



Эрик Фримен Элизабет Робсон

Руководство
пользователя
на понятном языке

Изучаем программирование на HTML5



Раскройте
секреты гуру HTML5



Узнайте, почему все,
что вашим друзьям
известно о видео,
может оказаться
ошибочным

Научитесь
избегать
досадных
проблем
с браузерной
поддержкой



Учебное руководство
по созданию веб-приложений
с использованием JavaScript



«Загрузите» HTML5
и JavaScript
прямо в свой мозг



Изучите
подводные камни,
связанные
с браузерами

O'REILLY®

ПИТЕР®

Отзывы о книге

«HTML5 является самой актуальной технологией создания веб-сайтов. Многие разработчики горят желанием воспользоваться ею для создания гибких, насыщенных медиа-сайтов, с которыми также будет удобно работать на планшетных компьютерах и смартфонах. Книга *„Изучаем программирование на HTML5“* — наилучший и самый увлекательный способ освоить эту восхитительную технологию. Очень рекомендую».

— **Мэриэни Марк, старший вице-президент по технологиям в Blue Nile Inc.**

«Являясь простой, информативной и увлекательной, книга *„Изучаем программирование на HTML5“* представляет собой настольное руководство для всех, кто желает изучить HTML5 или просто решил освежить свои знания по данной теме. Серия *„Head First“* помогает мне поддерживать знания на актуальном технологическом уровне, позволяя более эффективно управлять проектами и оказывать содействие моим разработчикам».

— **Тодд Гуилл, менеджер проектов, AllRecipes.com**

«Это вам не устаревающий DHTML! *„Изучаем программирование на HTML5“* рисует многообещающую и оптимистичную картину будущего Всемирной паутины через призму HTML5, давая вам шанс получить туда билет. Если вы ищете подробное, написанное доступным языком и порой довольно забавное руководство по HTML5, данная книга то, что вам нужно».

— **Мэини Отто, веб-разработчик и креативщик**

«Авторы данной книги попали в точку — навыки работы с JavaScript являются ключом к HTML5. Даже если вам никогда не доводилось писать JavaScript-программы, эта книга поможет быстро во всем разобраться благодаря наличию увлекательных и практических примеров».

— **Дэвид Пауэрс, автор книги «PHP. Создание динамических страниц» (PHP Solutions: Dynamic Web Design Made Easy)**

О других книгах серии *Head First*

«Будьте осторожны. Если кто-то из вас любит читать перед сном, то лучше отложите чтение книги „*Изучаем HTML, XHTML и CSS*“ (Head First HTML with CSS & XHTML) на дневное время. Эта книга будоражит мозг».

— Поулии Макнамара, Центр новых технологий и образования,
Фрибургский университет, Швейцария

«Книга „*Изучаем HTML, XHTML и CSS*“ представляет собой тщательно проработанное современное руководство по передовым практикам в области разметки и представления веб-страниц. Авторы предвидят, какие моменты могут вызвать у читателя замешательство, и своевременно разъясняют их. Использованный подход, в основе которого лежат обилие наглядных примеров и последовательность изложения, является оптимальным для читателя: он будет вносить небольшие изменения и наблюдать итоговый эффект в браузере, что позволит разобраться в назначении каждого нового элемента».

— Дэниел Гудмен, автор книги «*Динамический HTML: подробное руководство*»
(*Dynamic HTML: The Definitive Guide*)

«Книга „*Изучаем HTML, XHTML и CSS*“ с самого начала создает у читателя ощущение, что весь процесс обучения окажется простым и увлекательным. Освоение HTML при правильном объяснении не сложнее изучения основ родного языка, при этом авторы проделали отличную работу и приводят наглядные примеры по каждой концепции».

— Майк Дэвидсон, президент и исполнительный директор Newsvine, Inc

«Вместо изложения материала в стиле традиционных учебников „*Программируем для iPhone и iPad*“ предлагает читателю живую, увлекательную и даже приятную методику обучения программированию для iOS. Материал подобран умело и качественно: в книге рассматриваются многие ключевые технологии, включая Core Data, и даже такие важные аспекты, как проектирование интерфейса. И где еще можно прочитать, как UIWebView и UITextField беседуют у камина?»

— Шои Мерфи, проектировщик и разработчик приложений для iOS

«Книга „*Программируем для iPhone и iPad*“ объясняет принципы разработки приложений iOS с самого начала. Основные изменения по сравнению с первым изданием относятся к iOS 4, Xcode 4 и написанию приложений для iPad. Благодаря пошаговым описаниям с визуальным стилем изложения материала эта книга становится отличным средством изучения программирования для iPhone и iPad во всех аспектах, от простейших до нетривиальных».

— Рич Розен, программист и соавтор книги *Mac OS X for Unix Geeks*

«В отличие от многих невразумительных книг по программированию, насыщенных техническим жаргоном, руководства серии *Head First jQuery* помогают новичкам создавать их первые страницы jQuery на простом и доступном уровне».

— Линдси Скурас, юрист и программист-самоучка

Head First **HTML5** Programming



Wouldn't it be dreamy if there was an HTML5 book that didn't assume you knew what the DOM, events, and APIs were, all by page three? It's probably just a fantasy...

Ryan Benedetti
Ronan Cranley

O'REILLY®

Beijing • Cambridge • Köln • Sebastopol • Tokyo

Изучаем программирование на HTML5

Создание
веб-приложений
с использованием
JavaScript

Как было бы здорово, если бы существовала книга о HTML5, в которой не предполагается заранее, что читателю знакомы такие понятия, как объектная модель документа (DOM), события и API-интерфейсы. Об этом можно лишь мечтать...

Эрик Фримен
Элизабет Робсон



 **ПИТЕР®**

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

2013

ББК 32.988-02-018.1

УДК 004.43

Ф88

Фримен Э., Робсон Э.

Ф88 Изучаем программирование на HTML5. — СПб.: Питер, 2013. — 640 с.: ил.

ISBN 978-5-459-00952-1

Хотите создавать динамичные, интерактивные, насыщенные данными веб-страницы? Почему бы не использовать HTML5 для создания полнофункциональных веб-приложений? И почему бы не делать это с помощью современных методик, которые так же легко применимы к вашему настольному браузеру, как и к мобильным устройствам? Вам, конечно же, захочется использовать такие новейшие HTML5-технологии, как API-интерфейс Geolocation, элемент video, 2D-рисование, API-интерфейсы Web Storage и Web Workers и т. д. Пе так ли?

С помощью данной книги вы научитесь создавать веб-приложения с использованием современных стандартов и передовых методик завтрашнего дня. Вы изучите основы новых API-интерфейсов HTML5 и узнаете, как они взаимодействуют со страницами и приводятся в движение JavaScript-кодом, а также как использовать их для создания веб-приложений, которые впечатлят ваше начальство и изумят друзей.

ББК 32.988-02-018.1

УДК 004.43

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

© Authorized Russian translation of the English edition of Head First HTML5 Programming, ISBN 9781449390549 © 2011 Eric Freeman and Elisabeth Robson. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

ISBN 978-1449390549 (англ.)

© 2011 Eric Freeman and Elisabeth Robson. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

ISBN 978-5-459-00952-1

© Перевод на русский язык ООО Издательство «Питер», 2013

© Издание на русском языке, оформление ООО Издательство «Питер», 2013

Посвящается Стиву Джобсу, благодаря которому популярность HTML5 достигла таких высот, что данная книга должна разойтись огромным тиражом...

И еще раз посвящается Стиву Джобсу, потому что он наш герой.



←
Эрик Фримен



Элизабет Робсон

Эрик — один из основоположников серии «Head First». Кэти Сиерра так характеризует Эрика: «Один из редких людей, которые одинаково хорошо владеют языком, практическими навыками и знаниями культуры в разных областях, будь то сфера, в которой орудует хакер-хинстер, работает корпоративный вице-президент, проектировщик или эксперт-аналитик».

В профессиональном плане Эрик недавно подошел к почти десятилетней отметке в качестве должностного лица в медиа-компании: он занимает пост главного технического директора Disney Online & Disney.com в Walt Disney Company. В настоящее время Эрик занят WickedlySmart — стартапом, который он организовал совместно с Элизабет.

По образованию Эрик — ученый в области компьютерных наук, и ему довелось заниматься научными исследованиями с таким светилом, как Дэвид Геллернтер, во время его работы в качестве доктора философии в Йельском университете.

В свободное время Эрик серьезно увлекается музыкой; результат последнего проекта, над которым он работал совместно с пионером музыкального стиля «эмбиент» Стивом Роачем, имеется в электронном магазине приложений для iPhone и называется Immersion Station.

Пишите Эрику по адресу eric@wickedlysmart.com, а также посетите его сайт <http://ericfreeman.com>

Элизабет совмещает деятельность проектировщика программного обеспечения, носителя и инструктора. Она увлеклась технологиями еще во время учебы в Йельском университете, где получила степень магистра компьютерных наук и занималась разработкой языка параллельного визуального программирования и программной архитектуры.

Элизабет увлеклась созданием веб-приложений на самом раннем этапе развития Интернета. Она участвовала в создании заслужившего признание веб-сайта The Ada Project, который стал одним из нервных ресурсов, призванных помочь женщинам, занятым в сфере информатики. Она является одним из основателей WickedlySmart — образовательного онлайн-ресурса, посвященного веб-технологиям, на котором представлены ее книги, статьи, видео и прочее. Ранее, когда Элизабет была руководителем специальных проектов в O'Reilly Media, она лично проводила семинары и онлайн-лекции на разные технические темы, создавала образовательные ресурсы. До сотрудничества с O'Reilly Media Элизабет довелось поработать в Walt Disney Company, где она отвечала за руководство исследованиями и разработками в сфере цифрового медиа.

Пишите Элизабет на beth@wickedlysmart.com, посетите ее блог <http://elisabethrobson.com>

Содержание (сводка)

	Введение	21
1	Знакомство с HTML5. <i>Добро пожаловать в Вебвилль</i>	35
2	Знакомство с JavaScript и объектной моделью документа (DOM). <i>Немного кода</i>	69
3	События, обработчики и весь этот джаз. <i>Немного взаимодействия</i>	119
4	Функции и объекты JavaScript. <i>Серьезный JavaScript</i>	147
5	Создание HTML-страниц с поддержкой определения местоположения. <i>API-интерфейс Geolocation</i>	199
6	Общение с веб-службами. <i>Приложения-экстраверты</i>	247
7	Раскрываем в себе художника. <i>Элемент canvas</i>	315
8	Телевидение для нового поколения. <i>Элемент video... и наш особый гость – элемент canvas</i>	383
9	Сохраняем данные локально. <i>API-интерфейс Web Storage</i>	447
10	Применяем JavaScript на деле: <i>API-интерфейс Web Workers</i>	507
	Приложение. <i>Десять важных тем (которые мы не рассмотрели)</i>	565

Содержание (настоящее)

Введение

Ваш мозг думает о программировании на HTML5. Вы сидите за книгой и пытаетесь что-нибудь выучить, но ваш мозг считает, что вся эта писанина не нужна. Ваш мозг говорит: «Выгляни в окно! На свете есть более важные вещи, например сноуборд». Так как убедить свой мозг в том, что знание HTML5 и JavaScript не менее важно для вас?

Для кого написана эта книга?	22
Мы знаем, о чем вы думаете	23
И мы знаем, о чем думает ваш мозг	23
Метапознание: учимся учиться	25
Технические рецензенты	30
Благодарности	31
От издательства	33

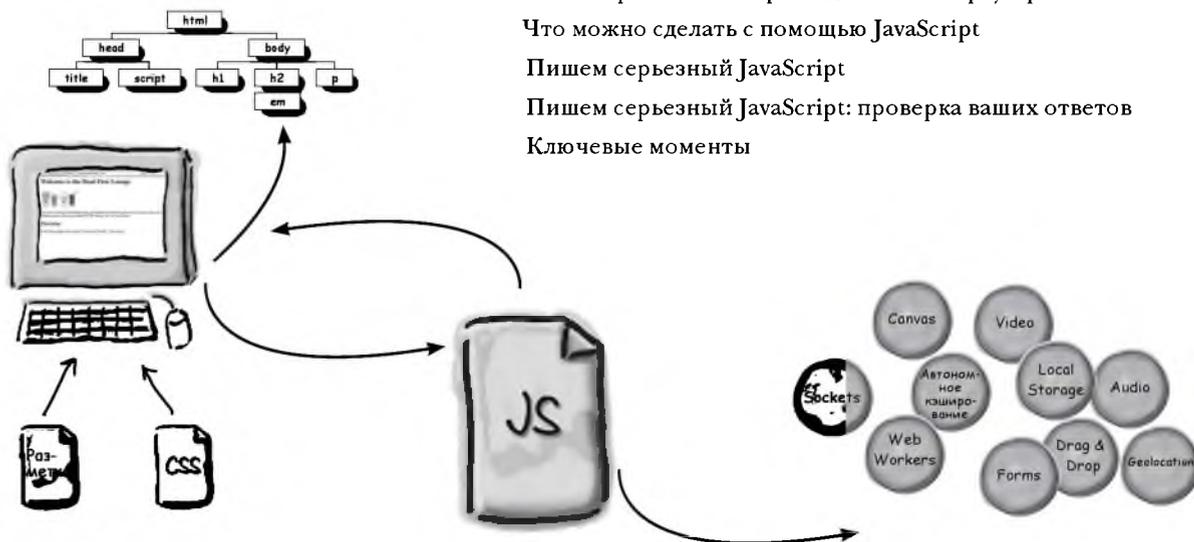
знакомство с HTML5

1

Добро пожаловать в Вебвилль

HTML стремительно развивается. Да, изначально HTML представлял собой простой язык разметки, однако с выходом новых версий он постепенно наращивал мускулы. В настоящее время мы располагаем версией HTML, заточенной под создание полноценных веб-приложений с поддержкой localStorage, 2D-рисования, автономного режима работы, сокетов, потоков и т. д. История развития HTML не всегда была радужной: она полна драматизма (об этом мы поговорим позже), а в этой главе мы для начала совершим увеселительный тур по Вебвиллю, чтобы вы могли разобраться во всем, что вкладывается в понятие «HTML5». Поэтому запрыгивайте к нам — мы отправляемся в Вебвилль, где за 3,8 страницы (ровно) пройдем путь от исходной точки до HTML5.

Переходите на HTML5 СЕГОДНЯ! Зачем ждать?	36
Представляем вам наш новый HTML5-модернизатор. Обновите свой HTML прямо сейчас	38
Вы ближе к HTML5-разметке, чем думаете!	41
Встречаем HTML5: Признания новой версии HTML	45
Просим встать НАСТОЯЩЕГО HTML5...	46
Как на самом деле работает HTML5...	48
Кто и что делает?	50
Ваша первая миссия: разведка в стане браузеров	51
Что можно сделать с помощью JavaScript	56
Пишем серьезный JavaScript	59
Пишем серьезный JavaScript: проверка ваших ответов	60
Ключевые моменты	65



2 знакомство с JavaScript и объектной моделью документа (DOM)

Немного кода

Благодаря JavaScript вы откроете для себя нечто новое. Вы уже все знаете о HTML-разметке (иначе называемой структурой) и CSS-стиле (также известном как представление), однако вам недостает знаний о JavaScript (или, как еще говорят, о поведении). Если ваш багаж знаний ограничивается лишь структурой и представлением, то вы, конечно же, сможете создавать прекрасно выглядящие страницы, однако они будут лишь простыми страницами. Но если вы добавите поведение, прибегнув к JavaScript, то сможете обеспечить для своих пользователей интерактивное взаимодействие; либо, что еще лучше, вы сможете создавать роскошные веб-приложения. Добавьте в свой инструментарий веб-разработчика наиболее интересные и универсальные знания о JavaScript и программировании!



Как работает JavaScript	70
Что можно сделать с помощью JavaScript?	71
Объявление переменной	72
Как присваивать имена переменным	74
Серьезное программирование	74
Выражения	77
Многократное выполнение одного и того же...	80
Принятие решений с использованием JavaScript	83
Принятие дополнительных решений... и добавление перехватывающего блока	84
Как и куда добавлять JavaScript в своих страницах	87
Как JavaScript взаимодействует с вашей страницей	88
Рецепт приготовления собственной объектной модели документа (DOM)	89
Первое испытание объектной модели документа (DOM)	90
Нельзя начинать взаимодействовать с DOM, пока веб-страница не загрузилась полностью	98
Для чего еще хорошо подходит DOM	100
Нельзя ли снова поговорить о JavaScript, или как осуществляется сохранение множественных значений при использовании JavaScript	101
Как создать массив	101
Phrase-O-Matic	105
Ключевые моменты	109

события, обработчики и Весь этот Джаз

3

Немного взаимодействия

Вам все еще не удается соприкоснуться с пользователем. Вы изучили основы JavaScript, однако могут ли ваши веб-страницы взаимодействовать с пользователями? Когда страницы откликаются на вводимые пользователем данные, они уже являются не простыми документами, а живыми, реагирующими приложениями. Из этой главы вы узнаете, как обрабатывать одну из форм ввода данных пользователем (извините за каламбур) и привязывать старомодный HTML-элемент `<form>` к современному коду. Это может показаться необычным, однако такой подход также эффективен. Пристегните ремни, поскольку наше путешествие по данной главе будет проходить на большой скорости: путь от простого приложения до интерактивного мы пройдем очень быстро.

Приготовьтесь к встрече с Wevville Tunes	120
Приступаем...	121
Когда я нажимаю кнопку Add Song (Добавить песню), ничего не происходит	122
Обработка событий	123
Составляем план...	124
Получение доступа к кнопке Add Song (Добавить песню)	124
Задание обработчика событий click для кнопки	125
Более пристальный взгляд на происшедшее...	126
Извлечение названия песни	128
Как добавить песню на страницу?	131
Как создать новый элемент	133
Добавление элемента в DOM	134
Соединяем все воедино...	135
...и проводим тест-драйв	135
Обзор того, что мы только что сделали	136
Как добавить приготовленный код...	139
Интегрирование приготовленного кода	140
Ключевые моменты	142



объекты и функции JavaScript

4

Серьезный JavaScript

Можете ли вы уже назвать себя создателем сценариев? Вполне возможно, поскольку вы уже многое знаете о JavaScript, однако кто захочет быть простым создателем сценариев, когда можно быть программистом? Пора проявить серьезность и поднять планку — настало время познакомиться с **функциями и объектами**. Они являются ключом к написанию более эффективного, хорошо организованного и легкого в сопровождении кода. Функции и объекты активно используются наряду с API-интерфейсами HTML5 JavaScript, поэтому чем лучше вы будете в них разбираться, тем быстрее сможете освоиться с тем или иным новым API-интерфейсом и начать его использовать. Пристегнитесь, поскольку эта глава потребует вашего всецелого внимания...



Расширяем ваш словарный запас	148
Как добавить свои собственные функции	149
Как работает функция	150
Локальные и глобальные переменные	157
Функции еще являются и значениями	162
Что можно сделать посредством функций как значений?	163
Как создать объект на JavaScript?	166
Что можно сделать с объектами	167
Поговорим о передаче объектов функциям	170
Наш следующий сеанс состоится в...	174
Объекты также могут обладать поведением...	176
Возвращаемся к приложению Webville Cinema...	177
Добавление ключевого слова this	179
Как создать конструктор	181
Воспользуемся нашим конструктором	182
Как на самом деле работает this?	183
Сразу же проведем тест-драйв нашего конструктора	187
Что такое объект window?	189
Более пристальный взгляд на window.onload	190
Еще один взгляд на объект document	191
Более пристальный взгляд на document.getElementById	191
Еще один объект, о котором нужно знать: объект элемента	192
Ключевые моменты	194

5

создание HTML-страниц с поддержкой определения местоположения

API-интерфейс Geolocation

Куда бы вы ни отправились, вас можно найти. Порой бывает так, что знание того, где вы находитесь, имеет существенное значение (особенно для веб-приложений). Из этой главы вы узнаете, как создавать веб-страницы с **поддержкой определения местоположения**. Иногда вы сможете определять местонахождение своих пользователей вплоть до угла, на котором они стоят, а иногда вам будет удаваться определить лишь район города, в котором они находятся (однако вы по-прежнему будете знать, какой это город!). Время от времени вы вообще не сможете получить хоть какую-нибудь информацию о местоположении пользователей в силу технических причин или просто потому, что им не нравится ваше чрезмерное любопытство. Да, представьте себе! Так или иначе, но в данной главе мы рассмотрим API-интерфейс JavaScript под названием Geolocation. Берите свое самое лучшее устройство с поддержкой определения местоположения (даже если это будет ваш настольный компьютер), и давайте приступим к работе.



Местоположение, местоположение...	200
Широта и долгота...	201
Как API-интерфейс Geolocation определяет местоположение пользователя	202
Так где же вы находитесь?	206
Как все это работает	210
Раскрываем наше тайное убежище...	213
Написание кода для определения расстояния	215
Отображение вашего местоположения на карте	216
Как добавить карту к странице	217
Прикалываем булавку на карту...	220
Прочие интересные вещи, которые можно сделать с использованием API-интерфейса Google Maps	222
Можем ли мы поговорить о точности?	225
“Wherever you go, there you are”	226
Приступаем к созданию приложения	227
Дорабатываем наш старый код...	228
Пора отправляться в путь!	230
Параметр positionOptions...	232
Мир параметров timeout и maximumAge...	233
Шлифуем наше приложение!	238
Интеграция нашей новой функции	239
Ключевые моменты	241

общение с Веб-службами

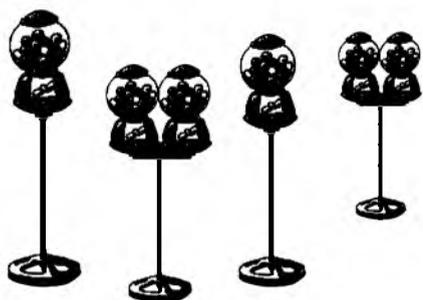
6

Приложения-экстраверты

Что-то вы слишком засиделись на своей странице. Настало время немного пообщаться с веб-службами с целью сбора данных и последующего возврата этой информации, что даст вам возможность создавать более эффективные веб-ресурсы, которые объединяют собираемые данные. Это важный момент в написании современных HTML5-приложений, и чтобы успешно им заниматься, вам необходимо знать, как происходит общение с веб-службами. Об этом мы и поговорим, а также научимся внедрять данные от веб-служб в свои страницы. Усвоив изложенный материал, вы сможете обращаться и взаимодействовать с любой веб-службой по своему выбору. Мы даже расскажем вам о новомодном жаргоне, которым следует пользоваться при общении с веб-службами. Вы познакомитесь с некоторыми новыми API-интерфейсами — так называемыми коммуникационными API-интерфейсами..



Осторожно, в этой главе вас ожидает неожиданный поворот!



Компании Mighty Gumball требуется веб-приложение	248
Подробнее о системе Mighty Gumball	250
Как выполняются запросы, адресованные веб-службам?	253
Как выполнять запросы из JavaScript	254
Подвинься, XML: встречайте JSON	260
Отображение данных об уровне продаж жвачки	264
Как установить собственный веб-сервер	265
Дорабатываем код с целью использования JSON	270
Переходим к использованию действующего сервера	271
Неожиданный поворот событий!	273
Помните, как мы столкнулись с неожиданным поворотом событий? Неполадки с приложением	276
Что за браузерная политика безопасности?	278
Какие у нас варианты?	281
Знакомство с JSONP	286
Что означает буква P в аббревиатуре JSONP?	287
Обновление кода веб-приложения Mighty Gumball	290
Дорабатываем приложение Mighty Gumball	298
Обновление URL-адреса JSON с целью включения lastreporttime	309
Ключевые моменты	311
Специальное сообщение из главы 7...	313

Раскрываем в себе художника

7

Элемент canvas

HTML больше не является просто языком «разметки». Благодаря новому HTML5-элементу `canvas` у вас появилась возможность собственноручно создавать и уничтожать *пиксели*, а также манипулировать ими. В этой главе мы воспользуемся элементом `canvas`, чтобы раскрыть таящегося в вас художника, — больше никаких разговоров о HTML, когда речь идет чисто о семантике и отсутствуют представления; используя `canvas`, мы начнем раскрашивать и рисовать цветом. Теперь все будет опираться на представления. Вы узнаете, как вставлять элемент `canvas` в свои страницы, как рисовать текст и графические изображения (естественно, используя JavaScript) и даже как поступать в случае с браузерами, не поддерживающими данный элемент. С чудесным элементом `canvas` вы встретитесь не только в этой, но и в других главах книги.

Новый HTML5-стартап только и ждет вашего внимания!



Наш новый стартап: TweetShirt	316
Взгляд на «оригинал-макет»	317
Как добавить <code>canvas</code> в свою веб-страницу	320
Как увидеть свой <code>canvas</code>	322
Рисование в элементе <code>canvas</code>	324
Выходим достойно из проблемной ситуации	329
TweetShirt: общая картина	331
Сначала напишем необходимый HTML	334
Теперь добавим <code><form></code>	335
Пришло время добавить JavaScript для вычислений	336
Написание функции <code>drawSquare</code>	338
Добавление вызова <code>fillBackgroundColor</code>	341
Тем временем, возвращаясь к TweetShirt.com...	343
Черчение контуров	345
Подробное исследование метода <code>arc</code>	348
Небольшой пример использования метода <code>arc</code>	350
Я говорю «градус», вы говорите «радиан»	351
Возвращаемся к написанию TweetShirt-кода для рисования кругов	352
Пишем функцию <code>drawCircle...</code>	353
Извлечение твитов	357
Завершаем написание функции <code>drawText</code>	365
Ключевые моменты	372

телевидение для нового поколения

8

Элемент `video`... и наш особый гость — элемент `canvas`

Нам больше не нужны какие-либо плагины. Элемент `video` отныне является полноценным членом HTML-семейства — просто вставьте его в свою страницу, и вы обеспечите прямую поддержку воспроизведения видео на большинстве устройств. Однако `video` — это нечто *значительно большее*, чем *просто элемент*: это API-интерфейс JavaScript, позволяющий управлять воспроизведением, создавать пользовательские видеointерфейсы и интегрировать видео с остальными HTML-элементами совершенно новыми способами. Говоря об *интеграции*, следует отметить, что между `video` и `canvas` *существует связь*, — вы увидите, что объединение этих элементов открывает новые широкие возможности по *обработке видео* в режиме реального времени. В этой главе мы сначала научимся внедрять элемент `video` в веб-страницу, а затем поговорим об использовании соответствующего API-интерфейса JavaScript. Вы будете поражены тем, что можно сделать с помощью небольшого количества разметки, JavaScript и элементов `video` и `canvas`.



↪ Настройте свой телевизор на Webville TV

Знакомство с Webville TV	384
Включите этот «телевизор» и протестируйте его...	385
Как работает элемент <code>video</code> ?	387
Пристальный взгляд на атрибуты элемента <code>video</code> ...	388
Что необходимо знать о форматах видео	390
Как жонглировать всеми этими форматами...	392
Я слышал, там будут API-интерфейсы?	397
Немного «программирования» содержимого Webville TV	398
Как написать обработчик «конца видео»	401
Как работает метод <code>canPlayType</code>	403
Распаковка демоблока	409
Рассмотрение остальной части «заводского» кода	410
Обработчики <code>setEffect</code> и <code>setVideo</code>	412
Реализация элементов управления видео	418
Реализация остальных элементов управления видео	419
Дорабатываем один нюанс...	420
Как происходит обработка видео	426
Как обрабатывать видео с использованием временного буфера	427
Реализация временного буфера в <code>canvas</code>	429
Займемся написанием эффектов	433
Как использовать события <code>eggot</code>	440

сохраняем данные локально

9

API-интерфейс Web Storage

Устали от того, что клиентские данные приходится втискивать в тесные шкафы файлы cookie? В 1990-е годы это не было проблемой, однако сейчас, в случае с веб-приложениями, запросы намного возросли. Как бы вы отнеслись к тому, если бы мы сказали, что у вас есть возможность выделять по 5 Мбайт на браузер каждого пользователя? Вы, вероятно, подумали бы, что мы шутим. Однако не стоит быть скептическими, поскольку API-интерфейс HTML5 Web Storage как раз и позволят делать это! Из данной главы вы узнаете обо всем, что необходимо для сохранения объектов локально на устройстве пользователя и использования их в работе ваших веб-приложений.

Трудно разобраться во всех своих делах, если после того, как их сделаешь, нельзя избавиться от клейких заметок, на которых они были записаны. Нельзя ли снабдить их функцией удаления?



Как работает браузерное хранилище (1995–2010)	448
Как работает Web Storage HTML5	451
Заметка для себя...	452
Были ли локальное хранилище и массив разделены при рождении?	458
Создание иптерфейса	463
Теперь добавим JavaScript	464
Завершаем создание интерфейса пользователя	465
Прервемся для небольшого запланированного мероприятия	468
Поддержка типа «сделай сам»	469
Дорабатываем наше приложение с использованием массива	474
Внесение изменений в createSticky с целью использования массива	475
Функция deleteSticky	483
Как выбрать заметку для удаления?	484
Как извлечь заметку для удаления, используя объект event	485
Удаление заметки также из DOM	486
Обновление интерфейса пользователя для выбора цвета заметок	487
Метод JSON.stringify – не только для массивов	488
Использование нового объекта stickyObj	489
Теперь, когда вы изучили localStorage, как вы собираетесь использовать его?	496

10

применяем JavaScript на деле

API-интерфейс Web Workers

Медленный сценарий — хотите продолжить его выполнение? Если вам доводилось достаточно тесно работать с JavaScript или путешествовать по Интернету, то вы, вероятно, сталкивались с диалоговым окном Slow Script (Медленный сценарий). Но как же сейчас, когда в компьютерах устанавливаются многоядерные процессоры, сценарии могут выполняться *слишком медленно*? Все потому, что JavaScript поддерживает выполнение только одного действия за раз. Однако с появлением HTML5 и Web Workers *все изменилось*. Теперь у вас есть возможность создавать множественные JavaScript-объекты `worker` для одновременного выполнения нескольких действий. Независимо от того, пытаетесь ли вы создать более отзывчивое приложение либо просто хотите по максимуму задействовать все возможности центрального процессора, API-интерфейс Web Workers придется кстати.

JavaScript-номок



Устрашающее диалоговое окно Slow Script (Медленный сценарий)	508
Как JavaScript проводит свое время	508
Когда однопоточность — это ПЛОХО	509
Поток веб-сценария worker	510
Как работают веб-сценарии worker	512
Ваш первый веб-сценарий worker...	517
Написание <code>manager.js</code>	518
Получение сообщений от веб-сценария worker	519
А теперь напишем worker	520
Захват виртуальных земель	528
Как вычисляется множество Мандельброта	530
Как задействовать сразу несколько веб-сценариев worker	531
Займемся созданием приложения Fractal Explorer	537
Создание веб-сценариев worker и раздача им задач...	542
Написание кода	543
Запуск веб-сценариев worker	544
Реализация кода worker	545
Возвращаемся к коду: как осуществляется обработка результатов работы worker	548
Подгоняем canvas под размеры окна браузера	551
Дотошный шеф-повар-программист	552
Время фипального тест-драйва!	553
Ключевые моменты	558

приложение: оставшиеся темы

Десять важных тем (которые мы не рассмотрели)

Мы рассмотрели массу различных тем, и вы почти закончили читать книгу. Нам грустно с вами расставаться, но прежде чем это сделать, нам необходимо поведать еще кое о чем, чтобы подготовить вас к свободному плаванию. В этот сравнительно небольшой раздел нельзя уместить все, что вам требуется знать. Вообще-то сначала мы включили все, что читателю следует знать о HTML5 (не рассмотренное в предыдущих главах), уменьшив размер шрифта до 0,00004. Все поместилось, однако текст стал слишком мелким и нечитаемым. Поэтому мы прилично сократили приложение и оставили только десять самых важных тем.



№ 1. Modernizr	566
№ 2. Элемент audio и API-интерфейс Audio	567
№ 3. jQuery	568
№ 4. XHTML мертв, да здравствует XHTML	570
№ 5. SVG	571
№ 6. Автономные веб-приложения	572
№ 7. API-интерфейс Web Sockets	573
№ 8. Дополнительно об API-интерфейсе Canvas	574
№ 9. API-интерфейс Selectors	576
№ 10. Однако есть даже еще кое-что!	577
HTML5-руководство по новым конструкциям	579
Вебвилльское руководство по семантическим элементам HTML5	580
Вебвилльское руководство по CSS3-свойствам	582

КАК ПОЛЬЗОВАТЬСЯ ЭТОЙ КНИГОЙ

Введение

Не могу поверить,
что они включили такое
в книгу о программирова-
нии на HTML5!



В этом разделе мы ответим на насущный вопрос:
«Так почему они включили ТАКОЕ в книгу о HTML5?»»

Для кого написана эта книга?

Если вы ответите утвердительно на все следующие вопросы:

- ① У вас есть компьютер с установленным **веб-браузером** и **текстовым редактором**?
- ② Вы хотите **узнать и разобраться**, как **создавать** веб-приложения, руководствуясь наилучшими методиками и самыми современными стандартами?
- ③ Вам больше по душе **оживленная интересная беседа**, нежели сухие и скучные академические лекции?

то эта книга — для вас.

Кому эта книга не подойдет?

Если вы ответите утвердительно на любой из следующих вопросов:

- ① Вы являетесь **абсолютным новичком** в создании веб-страниц?
- ② Вы уже заняты разработкой веб-приложений и ищете **справочник** по HTML5?
- ③ Вы **боитесь пробовать** что-то новое? Вы предпочитаете однообразный узор цветному рисунку? Вы полагаете, что техническая книга не может считаться серьезной, если в нее включены кадры из забавных образовательных фильмов 1950-х годов и наделенные человеческими качествами API-интерфейсы JavaScript?

Ознакомьтесь с книгой «Изучаем HTML, XHTML и CSS» (Head First HTML with CSS & XHTML), где вы найдете превосходное введение в мир веб-разработок, а затем возвращайтесь и присоединяйтесь к нам.

то эта книга — не для вас.

[Заметка от отдела продаж:
вообще-то эта книга для любого,
у кого есть деньги.]



Мы знаем, о чем вы думаете

«Как *это* можно назвать серьезной книгой по программированию на HTML5?»

«Почему здесь так много картинок?»

«Можно ли так чему-нибудь научиться?»

И мы знаем, о чем думает ваш мозг

Ваш мозг жаждет новизны. Он постоянно ищет, высматривает, ждет чего-то необычного. Таким его создала природа, и это помогает нам выжить.

Так что же ваш мозг делает со всеми рутинными, заурядными, обычными вещами, с которыми вы сталкиваетесь? Все, что он *может* сделать, это не позволить им помешать своей *истинной* работе, — фиксированию того, что *имеет значение*. Он не сохраняет в памяти скучную информацию; она никогда не пробьется через фильтр «это не является важным».

А как мозг *осознает*, что именно является важным? Допустим, вы вышли на прогулку, и вдруг перед вами выскакивает тигр. Что происходит у вас в мозге и в теле?

Активируются нейроны. Вспыхивают эмоции. Происходят химические реакции. И тогда ваш мозг *понимает*...

Конечно, это важно! Следует запомнить!

Но представим, что вы находитесь дома или в библиотеке. В безопасном, уютном месте, где не бывает никаких тигров. Вы что-то учите. Готовитесь к экзамену. Или пытаетесь освоить какой-то сложный технический материал, на что, как считает ваш босс, уйдет неделя или, самое большее, десять дней.

И тут возникает проблема. Ваш мозг будет пытаться оказать вам большую услугу. Он станет следить за тем, чтобы явно несущественные сведения не засорили ценные ресурсы памяти. Эти ресурсы лучше направить на сохранение действительно важной информации. Например, о том, что надо опасаться тигров, остерегаться огня или никогда больше не кататься на сноуборде в одних шортах.

Но при этом нельзя просто сказать своему мозгу: «Послушай, мозг, спасибо тебе, конечно, большое, однако неважно, насколько скучна книга или что она не вызывает во мне сильных эмоций в данный момент. Я в самом деле хочу запомнить информацию из нее».



Эта книга для тех, кто хочет учиться.

Как мы что-то узнаем? Сначала нужно это «что-то» *понять*, а потом не забыть. Затолкать в голову побольше фактов недостаточно. Согласно новейшим исследованиям в области когнитивистики, нейробиологии и психологии обучения, для *усвоения материала* требуется что-то большее, чем простой текст на странице. Мы знаем, как заставить ваш мозг работать.



Основные принципы серии «Head First»:

Наглядность. Графика запоминается гораздо лучше, чем обычный текст, и значительно повышает эффективность восприятия информации (до 89 % по данным исследований). Кроме того, материал становится более понятным. **Текст размещается на рисунках**, к которым он относится, а не под ними или на соседней странице.

Разговорный стиль изложения. Недавние исследования показали, что при разговорном стиле изложения материала (вместо формальных лекций) улучшение результатов на итоговом тестировании достигает 40 %. Рассказывайте историю, вместо того чтобы читать лекцию.

Не стоит быть слишком серьезным. Что вам самим больше по душе: вести оживленный разговор с интересным собеседником или слушать лектора?

Активное участие читателя. Пока вы не начнете напрягать извилины, в вашей голове ничего не произойдет. Читатель должен быть заинтересован в результате; он должен решать задачи, формулировать выводы и овладевать новыми знаниями. А для этого необходимы упражнения и каверзные вопросы, в решении которых задействованы оба полушария мозга и разные чувства.



Привлечение и удержание внимания читателя. Ситуация, знакомая каждому: «Я очень хочу изучить это, но засыпаю на первой странице». Мозг обращает внимание на интересное, странное, притягательное, неожиданное. Изучение сложной технической темы не обязано быть скучным. Интересное узнается намного быстрее.

Обращение к эмоциям. Известно, что наша способность запоминать в значительной мере зависит от эмоционального сопереживания. Мы запоминаем то, что нам небезразлично. Мы запоминаем, когда что-то чувствуем. Нет, сентименты здесь ни при чем: речь идет о таких эмоциях, как удивление, любопытство, интерес, и чувстве «Да, я крут!» при решении задачи, которую окружающие считают сложной, — или когда вы понимаете, что разбираетесь в теме лучше, чем всезнайка-Боб из технического отдела.

Метапознание: учимся учиться

Если вы действительно хотите быстрее и глубже усваивать новые знания — задумайтесь над тем, как вы задумываетесь. Учитесь учиться.

Мало кто из нас изучает теорию метапознания во время учебы. Нам *положено* учиться, но нас редко этому *учат*.

Но раз вы читаете эту книгу, то, вероятно, хотите научиться разрабатывать приложения, к примеру, для iPhone, и по возможности быстрее. Вы хотите *запомнить* прочитанное, а для этого абсолютно необходимо сначала *понять* прочитанное.

Чтобы извлечь максимум пользы из учебного процесса, нужно заставить ваш мозг воспринимать новый материал как *Не-что Важное*. Критичное для вашего существования. Такое же важное, как тигр. Иначе вам предстоит бесконечная борьба с вашим мозгом, который всеми силами уклоняется от запоминания новой информации.

Как же УБЕДИТЬ мозг, что знание HTML5 (и JavaScript) не менее важно, чем тигр?

Есть способ медленный и скучный, а есть быстрый и эффективный. Первый основан на тупом повторении. Всем известно, что даже самую скучную информацию *можно* запомнить, если повторять ее снова и снова. При достаточном количестве повторений ваш мозг кричит: «Вроде бы несущественно, но раз одно и то же повторяется *столько раз...* Ладно, уговорил».

Быстрый способ основан на **повышении активности мозга** и особенно на сочетании разных ее *видов*. Доказано, что все факторы, перечисленные на предыдущей странице, помогают вашему мозгу работать на вас. Например, исследования показали, что размещение слов *внутри* рисунков (а не в подписях, в основном тексте и т. д.) заставляет мозг анализировать связи между текстом и графикой, а это приводит к активизации большего количества нейронов. Больше нейронов — выше вероятность того, что информация будет сочтена важной и достойной запоминания.

Разговорный стиль тоже важен: обычно люди проявляют больше внимания, когда они участвуют в разговоре, так как им приходится следить за ходом беседы и высказывать свое мнение. Причем мозг совершенно не интересуется, что вы «разговариваете» с книгой! С другой стороны, если текст сух и формален, то мозг чувствует то же, что чувствуете вы на скучной лекции в роли пассивного участника, — его клонит в сон.

Но рисунки и разговорный стиль — это только начало.



Вот что сделали МЫ

Мы использовали **рисунки**, потому что мозг лучше приспособлен для восприятия графики, чем текста. С точки зрения мозга рисунок действительно стоит тысячи слов. А когда текст комбинируется с графикой, мы внедряем текст прямо в рисунки, потому что мозг при этом работает эффективнее.

Мы используем **пзбыточность**: повторяем одно и то же несколько раз, применяя разные средства передачи информации, обращаемся к разным чувствам — и все для новышения вероятности того, что материал будет закодирован в нескольких областях вашего мозга.

Мы используем конценции и рисунки несколько **пеожждапным** образом, потому что мозг лучше воспринимает новую информацию. Кроме того, рисунки и идеи обычно имеют **эмоциональное содержание**, потому что мозг обращает внимание на биохимию эмоций. То, что заставляет нас *чувствовать*, лучше запоминается, — будь то *шутка*, *удивление* или *интерес*.

Мы используем **разговорный стиль**, потому что мозг лучше воспринимает информацию, когда вы участвуете в разговоре, а не пассивно слушаете лекцию. Это происходит и при *чтении*.

В книгу включены многочисленные унаражения, потому что мозг лучше запоминает, когда вы что-то делаете. Мы постарались сделать их ненростыми, но интересными — то, что предпочитает большинство читателей.

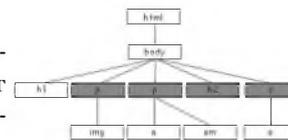
Мы совместили **несколько стилей обучения**, потому что одни читатели предпочитают ношаговые описания, другие стремятся сначала представить «общую картину», а третьим хватает фрагмента кода. Независимо от ваших личных предпочтений полезно видеть несколько вариантов представления одного материала.

Мы постарались задействовать **оба полушария вашего мозга**; это новышает вероятность усвоения материала. Пока одна сторона мозга работает, другая часть имеет возможность отдохнуть; это новышает эффективность обучения в течение продолжительного времени.

А еще в книгу включены **истории** и унаражения, отражающие другие точки зрения. Мозг глубже усваивает информацию, когда ему приходится оценивать и выносить суждения.

В книге часто встречаются **вопросы**, на которые не всегда можно дать простой ответ, потому что мозг быстрее учится и запоминает, когда ему приходится что-то делать. Невозможно накачать *мышцы*, наблюдая за тем, как занимаются *другие*. Однако мы позаботились о том, чтобы усилия читателей были приложены в *верном* направлении. Вам не придется ломать голову над невразумительными примерами или разбираться в сложном, перенасыщенном техническим жаргоном или слишком лаконичном тексте.

В историях, примерах, на картинках фигурируют **люди** — потому что вы тоже *человек*. И ваш мозг обращает на людей больше внимания, чем на неодушевленные *предметы*.



СТАНЬ браузером

**КЛЮЧЕВЫЕ
МОМЕНТЫ**



Кроссворды





Вырежьте и прикрепите
на холодильник.

Что можете сделать Вы, чтобы заставить свой мозг повиноваться

Мы свое дело сделали. Остальное за вами. Эти советы станут отправной точкой; прислушайтесь к своему мозгу и определите, что вам подходит, а что не подходит. Попробуйте новое.

- ① **Не торопитесь. Чем больше вы поймете, тем меньше придется запоминать.**
Просто читать недостаточно. Когда книга задает вам вопрос, не переходите к ответу. Представьте, что кто-то действительно задает вам вопрос. Чем глубже ваш мозг будет мыслить, тем скорее вы поймете и запомните материал.
- ② **Выполняйте упражнения, делайте заметки.**
Мы включили упражнения в книгу, но выполнять их за вас не собираемся. И не *разглядывайте* упражнения. Берите карандаш и пишите. Физические действия во время учения повышают его эффективность.
- ③ **Читайте врезки.**
Это значит: читайте все. *Врезки – часть основного материала!* Не пропусайте их.
- ④ **Не читайте другие книги после этой перед сном.**
Часть обучения (особенно перенос информации в долгосрочную память) происходит после того, как вы откладываете книгу. Ваш мозг не сразу усваивает информацию. Если во время обработки поступит новая информация, часть того, что вы узнали ранее, может быть потеряно.
- ⑤ **Пейте воду. И побольше.**
Мозг лучше всего работает в условиях «высокой влажности». Дегидратация (которая может наступить еще до того, как вы почувствуете жажду) снижает когнитивные функции.
- ⑥ **Говорите вслух.**
Речь активизирует другие участки мозга. Если вы пытаетесь что-то понять или лучше запомнить, произнесите вслух. А еще лучше, попробуйте объяснить кому-нибудь другому. Вы будете быстрее усваивать материал и, возможно, откроете для себя что-то новое.
- ⑦ **Прислушивайтесь к своему мозгу.**
Следите за тем, когда ваш мозг начинает уставать. Если вы начинаете поверхностно воспринимать материал или забываете только что прочитанное – пора сделать перерыв. С определенного момента попытки «затолкать» в мозг дополнительную информацию не только не ускоряют обучение, а скорее идут во вред ему.
- ⑧ **Чувствуйте!**
Ваш мозг должен знать, что материал книги действительно *важен*. Переживайте за героев наших историй. Придумывайте собственные подноски к фотографиям. Поморщиться над неудачной шуткой все равно лучше, чем не почувствовать ничего.
- ⑨ **Творите!**
Попробуйте применить новые знания в своей повседневной работе. Просто сделайте хоть что-нибудь, чтобы приобрести практический опыт за рамками упражнений. Все, что для этого нужно, – это карандаш и подходящая задача... задача, в которой изучаемые методы и инструменты могут принести пользу.

Примите к сведению

Это учебник, а не справочник. Мы намеренно убрали из книги все, что могло бы помешать изучению материала, над которым вы работаете. И при первом чтении книги начинать следует с самого начала, потому что книга предполагает наличие у читателя определенных знаний и опыта.

Мы предполагаем, что вы знаете HTML и CSS.

Если вы не знаете разметки HTML (то есть ничего о HTML-документах, об элементах, атрибутах, структуре свойств, структуре в сравнении с презентацией), то прочитайте книгу Э. Фримена, Э. Фримен «Изучаем HTML, XHTML и CSS» (СПб.: Питер, 2012) перед тем, как приступить к этой. Если же соответствующие знания у вас есть, то все в порядке.

От вас не требуется знание JavaScript.

Если у вас имеется какой-либо опыт программирования или написания сценариев (пусть и не на JavaScript), то это вам поможет. Однако в данной книге мы не ждем от вас знаний JavaScript; фактически, это издание является продолжением книги «Изучаем HTML, XHTML и CSS», где написание сценариев отсутствует.

Мы рекомендуем использовать разные браузеры.

Тестируйте страницы и веб-приложения, используя сразу несколько браузеров. Так вы узнаете, чем отличаются разные браузеры, и научитесь создавать страницы, нормально работающие в разных браузерах. Советуем использовать Google Chrome и Apple Safari, поскольку они наиболее соответствуют современным стандартам. Также рекомендуем вам применять для тестирования и новейшие версии других распространенных браузеров, таких как Internet Explorer, Firefox и Opera, а также мобильные браузеры на устройствах с операционными системами iOS и Android.

Упражнения ОБЯЗАТЕЛЬНЫ.

Не игнорируйте упражнения — они являются частью основного материала книги. Одни из них помогут вам запомнить материал, другие — понять его, а третьи — применить изученное на практике. Не пропускайте упражнения. Важны даже головоломки, поскольку они помогают мозгу усвоить концепции и дают возможность обдумывать изучаемые новые слова и термины в разном контексте.

Повторение применяется намеренно.

У книг этой серии есть одна принципиальная особенность: мы хотим, чтобы вы действительно хорошо усвоили материал. И чтобы вы запомнили все, что узнали. Большинство справочников не ставят своей целью успешное запоминание, но это не справочник, а учебник, поэтому некоторые концепции излагаются в книге по несколько раз.

Упражнения «Мозговой штурм» не имеют ответов.

В некоторых из них правильного ответа вообще нет, в других вы должны сами решить, насколько правильны ваши ответы (это является частью процесса обучения). В некоторых упражнениях «Мозговой штурм» приводятся подсказки, которые помогут вам найти нужное направление.

Системные требования

Для написания и работы с кодом на HTML5 и JavaScript вам понадобится текстовый редактор, браузер и, время от времени, веб-сервер (он может локально располагаться на вашем настольном компьютере).

В операционной системе Windows мы рекомендуем использовать такие текстовые редакторы, как PSPad, TextPad или EditPlus (но вы, если придется, можете пользоваться программой Notepad (Блокнот)). Для платформы Mac советуем TextWrangler, TextMate или TextEdit. Если вы работаете в Linux, то в вашем распоряжении масса встроенных текстовых редакторов, о которых, как мы полагаем, рассказывать не нужно.

Надеемся, что вы прочитали пункт о браузерах (см. предыдущую страницу) и установили как минимум два из них у себя в системе. Если нет, сделайте это. Вам также стоит уделить время и научиться работать с инструментами разработчика, встроенными в браузеры; в каждом из наиболее распространенных браузеров имеется встроенный инструментарий, который можно применять для инспектирования консоли JavaScript (вы сможете просматривать ошибки и вывод, генерируемый посредством `console.log`, что является альтернативой `alert`), использования веб-хранилища, объектной модели документа DOM, CSS-стиля, примененного к элементам, и многого другого. Некоторые браузеры даже поддерживают плагины, позволяющие добавлять новые инструменты разработчика. Чтобы освоить книгу, эти инструменты вам не потребуются, однако если у вас есть желание потратить время на изучение того, как ими пользоваться, то процесс разработки будет легче.

Некоторые параметры HTML5 или API-интерфейсы JavaScript требуют, чтобы файлы не просто загружались, а поступали при этом с реального веб-сервера (то есть URL-адрес должен начинаться с `http://`, а не с `file://`). В соответствующих местах книги мы указали, в каких случаях вам потребуется сервер, но если у вас возникло желание, рекомендуем установить веб-сервер у себя в системе прямо сейчас. В операционных системах Mac OS и Linux имеется встроенный сервер Apache, поэтому вам нужно лишь убедиться, что вы знаете, как получить к нему доступ и где размещать свои файлы, чтобы их можно было загружать, используя свой локальный сервер. В Windows вам потребуется установить Apache или IIS; если вы предпочтете Apache, то вам станет доступна масса инструментов с открытым исходным кодом, например WAMP и XAMPP, которые довольно просты в установке.

Вот и все! Удачного вам времяпровождения...

Технические рецензенты

Пол Барри ↷



Он не просто рецензент, а опытный автор, на счету которого такие книги, как «Изучаем Python» (Head First Python) и «Изучаем программирование» (Head First Programming)!

Лу Барр ↷



Мы пытались убедить ее, что нам нужна только помощь с графикой, но она не удержалась и выступила также техническим рецензентом.

Дэвид Пауэрс ↷



Наш главный технический рецензент.

Берт Бэйтс ↷



Не только занимается рецензированием, но также пишет книги! Как он только все успевает...

Ребекка Дан-Кэн ↷



Ребекка стала нашей лишней парой глаз; она помогала нам с кодом, выискивая в нем мелкие ошибки, которые никто не замечал (включая нас!)

Тревор Фарлоу ↷



Наш рецензент, выложившийся на 110%. Он даже совершал пробежки по ночам, тестируя наш геолокационный код.

Выражаем огромную признательность команде наших технических рецензентов. Своей работой они доказали, что мы не смогли бы обойтись без их технических знаний и опыта, а также внимательности к деталям. Дэвид Пауэрс, Ребекка Дан-Кэн, Тревор Фарлоу, Пол Барри, Луиза Барр и Берт Бэйтс приложили максимум усилий в своих рецензиях, благодаря чему эта книга стала намного-намного лучше. Вы молодцы, ребята!

Благодарности

За прекрасное техническое рецензирование:

Это уже превращается в традицию в наших книгах, и мы хотим еще раз горячо поблагодарить Дэвида Пауэрса, нашего уважаемого технического рецензента и автора многих изданий, включая книгу «PHP. Создание динамических страниц» (PHP Solutions: Dynamic Web Design Made Easy). Благодаря замечаниям Дэвида содержимое книг всегда значительно улучшается, и нам крепче спится по ночам от знания того, что если текст прошел через Дэвида, то в техническом плане там все отлично. Еще раз спасибо, Дэвид.

Группе O'Reilly:

На плечи Кортни Нэш легла тяжелая обязанность по руководству не только проектом книги «Изучаем программирование на HTML5», но и нами. Кортни расчистила все пути для нас и деликатно оказывала необходимое давление на каждого редактора, чтобы книга смогла выйти в свет. Однако главная заслуга Кортни заключается в том, что она обеспечила неопределимую обратную связь касательно книги и ее содержания, в результате чего текст подвергся ряду серьезных доработок. Благодаря усилиям Кортни эта книга стала намного лучше. Выражаем ей свою признательность.



↑ Лу Барр, снова! (И Тоби.)

Лу Барр также стала неотъемлемой частью процесса работы над книгой и внесла свой вклад, выступив в массе ролей: рецензента, графического дизайнера, главного художника, веб-дизайнера и мастера Photoshop. Спасибо тебе, Лу, без тебя у нас ничего бы не получилось!

← Заметка для редактора: можно ли застолбить этого парня на три наших следующие книги? Причем на эксклюзивных правах!



Кортни Нэш ↗

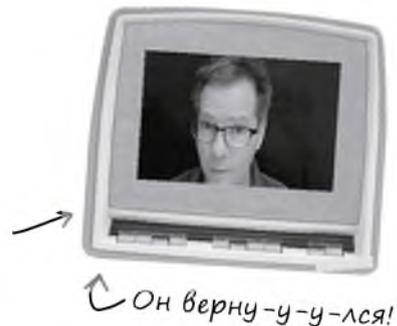
И спасибо всем остальным людям, благодаря труду которых эта книга увидела свет:

Благодарим остальных членов команды из O'Reilly за их разностороннюю поддержку. Это **Майк Хейдриксон, Майк Лукидес, Лоурел Рума, Карен Шейнер, Саидерс Кляйфельд, Кристен Борг, Карен Моитгомери, Рэйчел Моиагаи, Джули Хоукс и Нэнси Рэйхардт.**

И еще благодарности*

Также спасибо всем остальным

Джеймс Хейстридж написал оригинальный код, легший в основу приложения Fractal Explorer из главы 10, которое мы адаптировали в книге под свои нужды. Извиняемся, Джеймс, если приведенный в тексте код окажется не столь элегантен, как в исходном виде. Актер, художник и исполнительный директор Starbuzz **Лоуренс Заикоивски** принимал активное участие в создании книги и помогал тестировать видеоприложение из главы 8 (не пропустите!). **Городская ассоциация Бэйбридж Айленд** любезно разрешила нам использовать свой замечательный логотип, придуманный Дэниз Харрис, как символ штаб-квартиры WickedlySmart. Благодарим **Эитоии Виззари** и A&A Studios за возможность приводить фотоснимки их прекрасных фотокабинок. В нашем примере со стартапом TweetShirt вы увидите красивые иконки, предоставленные **ChethStudios.Net**. Выражаем признательность **Internet Archive** за кадры из фильмов, которые мы использовали для Webville TV. И спасибо **Дэниелу Штейнбергу**, который там работает и всегда готов нам помочь.



И наконец, выражаем благодарность **Кэти и Берту**

И последними, по пичуть не в меньшей степи, благодарим **Кэти Сиерру** и **Берта Бэйтса** — участников и **МОЗГОВОЙ ЦЕНТРА** всей пашей операции, которые к тому же являются основоположниками серии «Head First». Надеемся, что паша книга займет достойное место в этой серии.

Берт Бэйтс



Кэти Сиерра



Усердно проверяет на практике систему Парелли

*Так много благодарностей мы высказываем потому, что проверяем теорию о том, что каждый из упомянутых здесь людей захочет купить минимум один ее экземпляр (возможно даже больше, например, для своих родственников). Поэтому если вы желаете, чтобы мы упомянули вас в благодарностях в нашей следующей книге и у вас много родственников, пишите нам.

От издательства

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

Мы будем рады узнать ваше мнение!

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

1 знакомство с html5

Добро пожаловать в Вебвилль

Мы отправляемся в Вебвилль!
Там столько прекрасных домов в стиле HTML5, что просто безумие жить в каком-то другом месте! Давайте с нами, и мы покажем вам все местные достопримечательности.



Примечание:
язык XHTML
получил
«прощаль-
ное письмо»
в 2009 году,
о чем мы еще
поговорим.

HTML стремительно развивается. Да, изначально HTML представлял собой простой язык разметки, однако с выходом все новых версий он постепенно наращивал «мускулатуру». В настоящее время мы располагаем версией HTML, заточенной под создание полноценных веб-приложений с поддержкой localStorage, 2D-рисования, автономного режима работы, сокетов, потоков и т. д. История развития HTML не всегда была радужной: она полна драматизма (об этом мы поговорим позже), но в этой главе мы для начала совершим увеселительную поездку по Вебвиллю, чтобы вы могли разобраться во всем, что вкладывается в понятие «HTML5». Поэтому запрыгивайте к нам — мы отправляемся в Вебвилль, где за 3,8 страницы (ровно) пройдем путь от исходной точки до HTML5.



Переходите на HTML5 СЕГОДНЯ! Зачем ждгать?

Воспользуйтесь нашим новым

HTML5-модернизатором

и сделайте это всего за ТРИ ПРОСТЫХ ШАГА

Не раздумывайте! Акция действует в течение ограниченного времени. Мы возьмем ваши старые HTML-страницы и модернизируем их до HTML5 за ТРИ ПРОСТЫХ ШАГА.

Неужели все действительно так просто? Конечно! Мы даже подготовили для вас небольшую демонстрацию.

Взгляните на этот старый, потрепанный, выдавший лучшие дни HTML; мы превратим его в HTML5 прямо у вас на глазах:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <title>Head First Lounge</title>
    <link type="text/css" rel="stylesheet" href="lounge.css">
    <script type="text/javascript" src="lounge.js"></script>
  </head>
  <body>
    <h1>Добро пожаловать в Head First Lounge</h1>
    <p>
      
    </p>
    <p>
      Каждый вечер присоединяйтесь к нам для разговора
      за напитком <a href="elixirs.html">elixirs</a>,
      и, возможно, игры в two of Tap Tap Revolution.
      К вашим услугам беспроводной доступ; ЗСВС (Захватите Свой Собственный Веб-Сервер).
    </p>
  </body>
</html>
```

← Это вполне заурядный код веб-страницы Head First Lounge на HTML 4.01, который должен быть вам знаком по книге «Изучаем HTML» (Head First HTML) (а если и нет, не беспокойтесь, это абсолютно неважно).



МОЗГОВОЙ ШТУРМ

Вы увидите, насколько просто писать код на HTML5

Изучите приведенный выше пример кода, который написан на HTML 4.01 (это предыдущая версия языка). Взгляните на все строки и вспомните, что каждая из них делает. Отмечайте прямо на странице. Далее мы с вами разберемся в том, как переделать его в HTML5-код.

Возьми в руку карандаш

Внимательно взглянув на HTML-код на странице 36, заметили ли вы там какие-либо фрагменты разметки, которые могут претерпеть изменения при их переделке в HTML5? Или что бы вы сами там изменили? Дадим вам одну подсказку: определение `doctype`.

Это `doctype` для html — то, что нам нужно!
Это просто означает, что данный стандарт является открытым
В этой части говорится, что мы используем HTML версии 4.01 и что эта разметка написана на английском языке (EN)

```

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

Это указание на файл, который определяет соответствующий стандарт

Определение `doctype` относится к верхней части HTML-файла и сообщает браузеру, какого типа этот документ, в данном случае — HTML 4.01. Благодаря `doctype` браузер более точно может осуществлять интерпретацию и рендеринг страниц. Настоятельно рекомендуем вам использовать `doctype`.

Итак, что подсказывают ваши дедуктивные способности в плане того, как будет выглядеть определение `doctype` для HTML5? Напишите ответ здесь (вы сможете проверить его после того, как мы разберемся во всем чуть позже):

.....

.....

.....

.....

.....

Место для записи вашего ответа 

Представляем наш новый HTML5-модернизатор.

Обновите свой HTML прямо сейчас!



Шаг 1 удивит вас: мы начнем с верхней части HTML-страницы Head First Lounge и обновим определение `doctype`, придав ему новый блеск HTML5.

Вот как выглядела старая версия `doctype` в случае HTML 4.01:

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

Извиняемся перед теми, кто успел сделать себе татуировку `doctype 4.01` на память.

Вы, наверное, подумали, что в `doctype` мы собираемся заменить любое упоминание «4» на «5»? А вот и нет. Преимущество в том, что `doctype` для HTML5 выглядит совсем просто:

```
<!doctype html>
```

Вам больше не потребуется Google, чтобы узнать, как выглядит `doctype` для HTML5, как и не потребуется копировать и вставлять его из другого файла, поскольку это определение отличается крайней простотой и вам не составит труда его запомнить.

Однако постойте, есть еще кое-что...

Данное определение `doctype` подходит не только для HTML5, но и для всех *будущих версий* языка HTML. Другими словами, оно останется неизменным. Кроме того, оно будет работать и в старых версиях браузеров.

Разработчики стандартов W3C HTML пообещали, что на этот раз так и будет.



Если вы фанат телешоу «Фабрика красоты» (Extreme Makeovers) или «Потерявший больше всех» (The Biggest Loser), то шаг 2 вам понравится. Здесь у нас имеется `тег meta` с атрибутом `content...` впрочем, взгляните на картину «до» и «после»:

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

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

ДО (HTML 4)

ПОСЛЕ (HTML5)

Да, новый `тег meta` ~~значительно похудел~~ намного более прост. При использовании `тега meta` в HTML5 нужно лишь указать его наряду с кодировкой символов. Верите вы или нет, но все браузеры (новых и старых версий) уже понимают такое метаописание, поэтому его можно использовать в коде любой страницы — и оно *будет работать*.



А теперь шаг 3, заключительный. Здесь мы сосредоточим внимание на элементе `<head>` и модернизируем тег `link`. Вот что у нас имеется на текущий момент: тег `link` с атрибутом `type` со значением `text/css`, указывающий на каскадную таблицу стилей (значение `stylesheet`):

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

Код на старом HTML

Чтобы обновить этот код до HTML5, нужно просто убрать атрибут `type`. Почему? Потому что CSS объявлен стандартным языком стилей для HTML5 по умолчанию. Таким образом, после того как мы удалим атрибут `type`, наш обновленный тег `link` приобретет следующий вид:

```
<link rel="stylesheet" href="lounge.css">
```

HTML5



Поскольку вы работали быстро, у нас есть специальный бонус для вас. Мы еще больше облегчим вам жизнь, упростив тег `script`. JavaScript в случае HTML5 стал стандартным языком сценариев по умолчанию, поэтому вы можете удалить атрибут `type` также и из тегов `script`. Вот каким станет наш тег `script` без атрибута `type`:

```
<script src="lounge.js"></script>
```

Не беспокойтесь, если вы не слишком много знаете о тегах `script`, мы еще дойдем до него...

Либо, если вы будете иметь дело со встроенным кодом, можете просто написать свой сценарий следующим образом:

```
<script>
```

```
var youRock = true;
```

```
</script>
```

Весь ваш JavaScript будет здесь.

Подробнее о JavaScript мы поговорим позже.



Поздравляем, теперь вы сертифицированы модернизировать любой HTML-код до HTML-5!

Как продвинутый пользователь HTML5-модернизатора, вы обладаете всем необходимым для того, чтобы обновить любую нормальную HTML-страницу до HTML5. Дерзайте, пришло время реализовать полученные знания на практике!

Постойте-ка, столько суеты вокруг HTML5, и вдруг оказывается, что это все, что от меня потребуется? О чем же тогда оставшаяся часть книги?

Ладно, ладно, вы нас поймали. Пока мы вели речь о модернизации кода HTML-страниц, созданных с помощью старой версии этого языка, с целью поделить их всеми преимуществами, которые дает HTML5. И, как можно было убедиться, если вы знакомы с HTML 4.01, то это просто отлично, поскольку HTML5 — это расширенный HTML 4.01 (то есть практически все имеющееся в нем поддерживается и в HTML5), и вам потребуется лишь знать, как определять `doctype` и остальные теги в элементе `<head>`, чтобы начать программировать на HTML5.

Но вы правы в том, что мы повели себя несколько глупо, поскольку, конечно же, язык HTML5 — это нечто большее, чем просто модернизация нескольких элементов. Разработчиков в нем привлекает возможность создавать насыщенные, интерактивные страницы (или даже сложные веб-приложения), а также то, что он включает в себя целое семейство технологий, которые работают рука об руку с этим языком разметки.

Однако не будем спешить. Прежде чем двинуться дальше, сделаем еще кое-что, чтобы окончательно разобраться с нашей HTML-разметкой.





Возьми в руку карандаш

Вы ближе к HTML5-разметке, чем думаете!

Здесь приведена разметка на старом HTML, которая нуждается в обновлении. Осуществите HTML5-модернизацию и обновите этот написанный на HTML 4.01 код до HTML5. Не бойтесь делать пометки в книге, вычеркивайте приведенную здесь HTML-разметку и добавляйте новый код, который вам потребуется. Мы немного помогли вам и выделили области, нуждающиеся в изменениях.

Закончив, напечатайте код (или внесите изменения в файл упражнения, если вам так больше нравится), загрузите его в браузер и, откинувшись на спинку стула, насладитесь своим первым творением на HTML5. Ах да, наши ответы вы найдете на следующей странице.



Чтобы загрузить весь код и файлы примеров для этой книги, посетите страницу <http://wickedlysmart.com/hfhtml5>.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Head First Lounge</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
    <link type="text/css" rel="stylesheet" href="lounge.css">
    <script type="text/javascript" src="lounge.js"></script>
  </head>
  <body>
    <h1>Добро пожаловать в Head First Lounge</h1>
    <p>
      
    </p>
    <p>
      Каждый вечер присоединяйтесь к нам для разговора
      за напитком <a href="elixirs.html">elixirs</a>,
      и, возможно, игры в Tap Tap Revolution.
      Всегда к вашим услугам беспроводной доступ; ЗССВС (Захватите Свой
      Собственный Веб-Сервер).
    </p>
  </body>
</html>
```

Возьми в руку карандаш



Решение

Вы ближе к HTML5-разметке, чем вы думаете!

Здесь приведена разметка на старом HTML, которая нуждается в обновлении. Осуществите HTML5-модернизацию и обновите этот написанный на HTML 4.01 код до HTML5. Не бойтесь делать пометки в книге, вычеркивайте приведенную здесь HTML-разметку и добавляйте новый код, который вам потребуется. Мы немало помогли вам и выделили области, нуждающиеся в изменениях.

Вот наше решение.

Обновленный код будет выглядеть так:

```

<!doctype html> ← Определение doctype
<html>
  <head>
    <title>Head First Lounge</title>

    <meta charset="utf-8"> ← Tez meta
    <link rel="stylesheet" href="lounge.css"> ← Tez link
    <script src="lounge.js"></script> ← Tez script
  </head>
  <body>
    <h1>Добро пожаловать в Head First Lounge</h1>
    <p>
      
    </p>
    <p>
      Каждый вечер присоединяйтесь к нам для разговора
      за напитком <a href="elixirs.html">elixirs</a>,
      и, возможно, игры в Tap Tap Revolution.
      Всегда к вашим услугам беспроводной доступ; ЗССВС (Захватите Свой
      Собственный Веб-Сервер) .
    </p>
  </body>
</html>

```

Вот четыре строки, благодаря изменению которых наша веб-страница Head First Lounge получила официальное право называться HTML5-страницей.

Не верите нам? Зайдите по адресу <http://validator.w3.org/>, и вы убедитесь — она проходит валидацию как HTML5-страница. Без шуток!

Часть Задаваемые Вопросы

В: А как все это работает в старых версиях браузеров? Все эти новые `doctype`, `meta` и т. д... ведь как-то же могут старые браузеры работать с этим новым синтаксисом?

О: Да, могут. Взгляните на атрибуты `type` тегов `link` и `script`; сейчас имеет смысл избавиться от них в HTML5, поскольку CSS и JavaScript теперь являются стандартами (и, конечно же, технологиями по умолчанию для работы со стилями и написания сценариев соответственно). Как оказалось, браузерам уже заранее было известно, что CSS и JavaScript являются таковыми. Так, по стечению обстоятельств новый язык разметки HTML5 уже довольно долгое время поддерживается существующими браузерами. То же самое справедливо и в случае с `doctype` и тегом `meta`.

В: А как насчет нового `doctype`? Что-то с ним все стало слишком просто; у него нет даже версии или идентификатора DTD.

О: Да, кажется немного необычным, что после многих лет применения комплексных `doctype` теперь мы можем упростить их до фразы «мы используем HTML». А произошло вот что: HTML ранее основывался на стандарте SGML, который требовал как комплексных форм `doctype`, так и DTD. Новый стандарт отошел от SGML, преследуя этим цель сделать язык HTML проще и гибче. Таким образом, нужда в комплексных формах отпала. Здесь не обошлось без некоторой доли везения в том плане, что почти все браузеры просто ищут HTML в определении `doctype`, чтобы убедиться в том, что они осуществляют разбор именно HTML-документа.

В: Это было всерьез, когда вы говорили, что `doctype` останется неизменным? Как мне казалось, для браузеров важен контроль версий. Почему бы не использовать `<!doctype html5>`? Наверняка в будущем появится HTML6. Ведь так?

О: Подход к использованию `doctype` претерпевал изменения, и разработчики браузеров применяли его для того, чтобы дать указание своим браузерам осуществлять рендеринг в их собственном стандартном режиме. Теперь, когда у нас имеется намного более точный стандарт, `doctype` в HTML5 сообщает любому браузеру, что конкретный документ является стандартным HTML, будь он версии 5, 6 или любой другой.

В: Предполагаю, что разные браузеры пользователей будут поддерживать различающиеся наборы возможностей HTML5. Что делать в таком случае?

О: Да, это так, особенно пока HTML5 не обрел 100%-ную поддержку всеми браузерами. Об этих аспектах мы поговорим позже.

В: А почему вообще все это имеет значение? Я вот написал код веб-страницы без `doctype` и тега `meta`, и она отлично работает. Зачем мне лишняя головная боль, если в таком подходе нет абсолютно ничего неправильного?

О: Да, браузеры легко пропускают мелкие ошибки в HTML-файлах. Однако если вы включите соответствующие `doctype` и теги `meta`, то сможете быть уверены в том, что браузеры будут знать, что именно вы от них хотите, а не гадать об этом. Кроме того, в случае с людьми, пользующимися старыми версиями браузеров, указание нового `doctype` означает, что они будут использовать стандартный режим, что как раз вам и нужно. Стандартный режим — это режим, в котором браузер считает, что написанный вами HTML-код соответствует стандарту, поэтому он будет пользоваться правилами данного стандарта при интерпретации вашей страницы. Если вы не укажете `doctype`, то некоторые браузеры могут перейти в режим совместимости и посчитать, что ваша страница написана для старых версий браузеров, когда соответствующий стандарт еще не был на должной высоте, и неправильно интерпретировать страницу (или решить, что она просто некорректно написана).

В: А что случилось с XHTML? Ведь еще несколько лет назад казалось, что за ним будущее.

О: Да, так оно и было. Однако потом гибкость возобладали над строгим синтаксисом, и XHTML (XHTML 2, если быть точными) постепенно стал умирать, поскольку новый HTML5 оказался более либеральным в плане написания людьми веб-страниц (и осуществления их рендеринга браузерами). Но пусть вас это не смущает, так как знание XHTML лишь сделает из вас еще более успешного HTML5-разработчика. Кстати, если вы испытываете привязанность к XML, знайте, что существует также способ написания HTML5-кода в строгой форме. Подробнее об этом мы поговорим позже...

В: Что такое UTF-8?

О: UTF-8 — это кодировка символов, поддерживающая множество алфавитов, включая незападные. Вам, вероятно, доводилось сталкиваться с другими наборами символов, использовавшимися в прошлом, однако UTF-8 продвигается как новый стандарт. Она также быстрее и легче запоминается, чем предшествующие кодировки символов.



Мы все-таки не ожидаем, что вы знаете HTML5.

Даже если вы никогда ранее не занимались HTML5, это не проблема, однако у вас должен иметься опыт работы с HTML и знание таких базовых аспектов, как элементы, теги, атрибуты, вложения, понимание разницы между семантической разметкой и добавлением стиля и т. д.

Если вы не знакомы с данными аспектами, то мы возьмем на себя смелость дать вам небольшой совет (и бесстыдным образом кое-что прорекламировать): есть еще одна книга из этой серии, называющаяся «Изучаем HTML, XHTML и CSS», и вам следует ее прочитать. Если же вы отчасти знакомы с языками разметки, можете быстро ознакомиться с данным изданием или использовать его как справочник при чтении данной книги.



Мы также предусмотрели небольшое руководство по разметке HTML5 и CSS3 в приложении. Если вам нужен краткий обзор нововведений, то вы найдете его в конце книги.





ВСТРЕЧАЕМ HTML5

Интервью недели:

Признания новой версии HTML

Head First: Добро пожаловать, HTML5. Весь Интернет просто гудит от разговоров о Вас. Как Вам кажется, Вы во многом похожи на HTML 4. В чем же причина всеобщего ажиотажа вокруг Вас?

HTML5: Всеобщий ажиотаж объясняется тем, что я предоставляю возможности по созданию совершенно нового поколения веб-приложений и обеспечению качественного взаимодействия с ними.

Head First: Согласен, по почему HTML 4 или казавшийся многообещающим XHTML не сделали этого?

HTML5: XHTML 2 оказался тупиковой ветвью эволюции. Каждый, кому довелось писать код веб-страниц на этом языке, терпеть его больше не может. XHTML запово изобрел подход к написанию разметки веб-страниц, который уже и так используется, и не привнес в страницы ничего нового. Я сказал: «Постойте-ка, я ведь могу делать новые вещи и при этом заключаю в себе все те возможности, которые существовали до меня». Я имею в виду, что если что-то работает, то зачем запово изобретать колесо. Такова моя философия.

Head First: Но известно ли вам, что некоторые разработчики стандартов по-прежнему заявляют, что Интернету будет лучше, если он станет держиваться их «безупречных» стандартов?

HTML5: Знаете, мне все равно, что они там говорят. Я прислушиваюсь к людям, которые реально запыты написанием веб-страниц: как они используют меня, как я могу им помочь. Вторыми в моем списке идут создатели веб-браузеров. А разработчики стандартов стоят в этом списке последними. Я стану прислушиваться к их словам только при условии, что они не расходятся с мнением пользователей.

Head First: Почему же?

HTML5: Потому что если пользователи и создатели браузеров не согласны с разработчиками стандартов, то это паводит на мысли о правильности мпе-

ния последних. К счастью, с людьми, работающими над спецификациями HTML5, мы полностью сохдимся во взглядах.

Head First: Возвращаясь к предыдущей версии HTML, Вы отмечали, что являетесь расширением HTML 4.01. То есть Вы обратно совместимы с пей, правильно? Означает ли это, что Вам придется справиться с не всегда удачными в плане дизайна веб-страницами из прошлого?

HTML5: Обещаю, что приложу максимум усилий для того, чтобы справиться со всем, что мне подкинут из прошлого. Но отмечу, что это не значит, что со мной так и нужно обращаться. Я хочу, чтобы создатели веб-страниц приобщались к новейшим стандартам и использовали меня наилучшим образом. Благодаря этому они смогут получить максимальную отдачу от моего применения. Но, с другой стороны, я не спасу и смогу обеспечить отображение старой веб-страницы в силу своих возможностей, даже если она не была модернизировапа до HTML5.

Head First: Мой следующий вопрос звучит так...

HTML5: Стойте, стойте!!! Все эти вопросы касаются прошлого. Мы с Вами не говорим о том, что важно здесь и сейчас. Поскольку речь идет о моей разметке, хочу сказать, что моя персональная миссия заключается в том, чтобы охватить своими объятиями весь Интернет, внедрить новые структурные элементы, облегчающие жизнь веб-разработчикам, и помочь всем создателям браузеров поддерживать согласованную семантику вокруг разметки HTML5. Но на самом деле я здесь для того, чтобы рассказать Вам о своем дополнительном предпазачении: веб-приложе...

Head First: ...Жаль, HTML5, по паше время истекло. Спасибо, в следующем интервью мы обязательно поговорим обо всем, о чем Вы пожелаете.

HML5: Б-р-р-р, терпеть не могу, когда так случается!

Просим вставить НАСТОЯЩЕГО HTML5...

Итак, вы терпеливо выслушали наше шуточное повествование о «HTML5-модернизаторе», и мы уверены, что вы уже догадались о том, что HTML5 представляет собой нечто намного большее, чем там было сказано. Если поспрашивать разных людей, то в ответ можно услышать, что, по слухам, HTML5 устрояет необходимость в плагинах, может использоваться повсюду, начиная от простых страниц и заканчивая играми типа Quake, является кремом из взбитых сливок на десерте. Каждый по-разному представляет себе, что такое HTML5...



HTML5 — это все о мультимедиа, о том, как избавиться от плагинов, о возможностях собственной поддержки аудио и видео.



Нет, скорее, он об описательной разметке.



Вообще-то он касается создания насыщенных клиентских интернет-приложений. Вместо того чтобы создавать их с помощью инструментов вроде Flash, теперь я могу использовать элемент canvas, трансформации и JavaScript для генерирования классных интерфейсов и анимаций.



Замечательная вещь в HTML5 — хранение информации на стороне клиента и кэширование. Как вам возможность автономного доступа к данным из Интернета?



Так здорово, что я могу использовать Web Workers для повышения эффективности моего JavaScript и увеличения отзывчивости веб-страниц.



В CSS также есть масса всего нового, что можно использовать вместе с HTML5: продвинутые селекторы, анимации и — да, эффект отбрасывания тени!



И не забудьте о мобильных устройствах. Мне нравится возможность создавать страницы с поддержкой определения местоположения.

Хорошая новость заключается в том, что HTML5 действительно является всем тем, о чем сказано выше. Когда люди говорят о HTML5, они имеют в виду *семейство технологий*, которые в сочетании друг с другом образуют целую новую палитру для создания веб-страниц и приложений.

Как на самом деле работает HTML5...

Итак, как было сказано выше, в основе HTML5 лежит семейство технологий, по что это значит? Что ж, как вам уже известно, существует HTML-разметка сама по себе, которая была расширена с целью включения ряда новых элементов. Кроме того, с выходом CSS3 много нового появилось в каскадных таблицах стилей, что открыло еще более широкие возможности по стилизации веб-страниц. Также существует турбопапетатель под названием JavaScript и целый новый набор API-интерфейсов *JavaScript*, доступных вам.

В приложении вы найдете замечательное Вебвилльское руководство по новой HTML-разметке и свойствам CSS3.

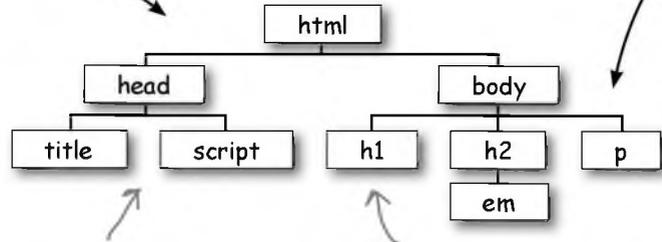
Давайте заглянем за кулисы и посмотрим, как все это объединяется.

1 Браузер загружает документ, включающий HTML-разметку и CSS-стили.

2 При загрузке страницы браузер также создает **внутреннюю модель документа**, которая будет содержать все элементы вашей HTML-разметки.



Для каждого элемента в вашей HTML-разметке браузер создает объект, который будет представлять соответствующий элемент, и размещает его в древовидной структуре со всеми остальными элементами...

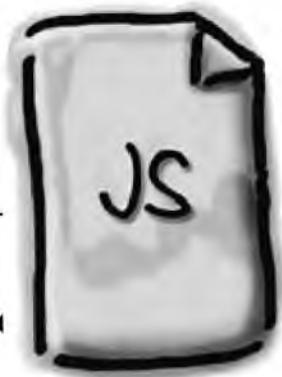


...Это дерево называется объектной моделью документа (Document Object Model, DOM). Вы будете часто сталкиваться с ней по ходу книги, поскольку данная модель играет важную роль в том, как мы добавляем поведение к веб-страницам с помощью JavaScript (подробнее в главе 2).

Стиль страницы (при наличии такового) берется из CSS3, который стал расширением CSS2 и включил в себя множество распространенных идиом, используемых в Интернете (например, отбрасывание тени и закругленные углы).

С появлением HTML5 разметка претерпела ряд усовершенствований, как вы могли убедиться в случае с тегами в элементе `<head>`. Кроме того, появились дополнительные элементы, которые вы можете использовать (с некоторыми из них вы столкнетесь в процессе чтения книги).

За сценой



3 При загрузке страницы браузер также загружает ваш **JavaScript-код**, выполнение которого обычно запускается сразу после окончания загрузки страницы.

Применяя JavaScript, вы сможете взаимодействовать со своей страницей путем манипулирования объектной моделью документа (DOM), обеспечивать реакцию на действия пользователя или генерируемые браузером события либо использовать весь набор новых API-интерфейсов.

JavaScript взаимодействует с вашей страницей посредством объектной модели документа (DOM).

API-интерфейсы, также известные как интерфейсы прикладного программирования (Application Programming Interfaces), обеспечат для вас набор объектов, методов и свойств, которые вы сможете использовать для доступа ко всей функциональности этих технологий. Многие из этих API-интерфейсов мы рассмотрим далее.

4 API-интерфейсы обеспечат вам доступ к элементам audio и video, 2D-рисованию с использованием canvas, к localStorage и прочим замечательным технологиям, необходимым для создания приложений. И не забывайте, что для использования всех этих API-интерфейсов необходим JavaScript.

Знакомьтесь: API-интерфейсы JavaScript



* КТО И ЧТО ДЕЛАЕТ? *

Мы с вами говорили о «семействе технологий» столько раз, что уже кажется, что мы сами одна семья. Однако мы по-настоящему так и не разобрались в том, что конкретно они собой представляют, поэтому, наконец, сделаем это. Ниже представлен перечень большинства членов этого семейства, посмотрите, сможете ли вы разобраться, кто есть кто. Мы вас опередили и в качестве примера соотнесли одно из описаний с нужной позицией. *Не беспокойтесь, мы знаем, что это ваша первая встреча с членами семейства HTML5, поэтому ответы вы найдете в конце главы.*

CSS3

Используя мейя, вы можете рисовать прямо на своей веб-странице. Благодаря мне можно рисовать текст, изображения, линии, круги, прямоугольники, узоры, применять градиенты. Я помогу вам раскрыть таящегося в вас художника.

Web Workers

Мейя использовали в HTML 4 для ввода информации, но я стал еще лучше в HTML5. Я могу требовать от вас заполнять все поля и способен проверять, ввели ли вы адрес электронной почты, URL-адрес или номер телефона туда, куда требуется.

Forms

Ранее для обеспечения функциональности, аналогичной нашей, вам приходилось использовать плагины, но теперь мы стали полноценными членами семейства элементов HTML. Хотите посмотреть или послушать что-либо? Тогда мы нужны вам.

Автономные Веб-приложения

Мы здесь для того, чтобы помочь вам со структурой и семантическим значением вашей страницы и обеспечить новые способы создания секций, заголовков, пикших колопититилов и навигации на ваших страницах.

Audio и Video

Я самый стильный во всем семействе. Знаете ли вы, что теперь я могу апимировать ваши элементы, придавать их углам закругленность и даже обеспечивать эффект отбрасывания тени?

Новые элементы разметки

Используйте мейя как часть локального хранилища в браузере любого пользователя. Вам необходимо сохранять установки, элементы, помещенные в электронную корзину, или, возможно, даже припрятать большой кэш для увеличения производительности? Тогда я нужен вам API-интерфейс.

Local Storage

Вам требуются приложения, способные работать даже тогда, когда отсутствует подключение к Сети? Я могу вам помочь.

Canvas

Я — API-интерфейс, который поможет вам определить свое местоположение, и отлично работаю с Картами Google.

Geolocation

Я потребуюсь, когда вам будет необходимо, чтобы несколько сценариев выполнялись параллельно в фоновом режиме, благодаря чему ваш интерфейс пользователя сможет оставаться отзывчивым.

ВАША МИССИЯ...

...если вы на нее согласны, заключается в проведении разведки в стане всех HTML-браузеров. Мы уверены, что вы слышали о том, что одни из них поддерживают HTML5, а другие нет. Вы должны тщательно во всем разобраться, поскольку истина где-то рядом...

знакомство с HTML5

ДЕЛО: HTML5

ВАША ПЕРВАЯ МИССИЯ: РАЗВЕДКА В СТАНЕ БРАУЗЕРОВ

**СОВЕРШЕННО
СЕКРЕТНО**

ВАМ ПОРУЧАЕТСЯ ОПРЕДЕЛИТЬ [REDACTED] ТЕКУЩИЙ УРОВЕНЬ ПОДДЕРЖКИ ПО КАЖДОМУ ИЗ ПРИВЕДЕННЫХ НИЖЕ БРАУЗЕРОВ (ПРИМЕЧАНИЕ: ВЫ СМОЖЕТЕ ОТЫСКАТЬ НУЖНУЮ ИНФОРМАЦИЮ ПО АДРЕСУ [HTTP://WICKEDLYSMART.COM/HTML5/BROWSERSUPPORT.HTML](http://wickedlysmart.com/html5/browsersupport.html), [REDACTED]. ПРИНИМАЙТЕ ВО ВНИМАНИЕ НОВЕЙШИЕ ВЕРСИИ БРАУЗЕРОВ. ПО КАЖДОМУ БРАУЗЕРУ, ПРИВЕДЕННОМУ В СТОЛБЦЕ «БРАУЗЕР/ВОЗМОЖНОСТЬ», ОТМЕТЬТЕ ВОЗМОЖНОСТИ, КОТОРЫЕ ОН ПОДДЕРЖИВАЕТ, А ЗАТЕМ ДАЙТЕ СВОЮ СУБЪЕКТИВНУЮ ОЦЕНКУ ТОГО, НАСКОЛЬКО ХОРОШО ОН СОВМЕСТИМ С HTML5 [REDACTED]. ПО ВОЗВРАЩЕНИИ ПРЕДСТАВЬТЕ ОТЧЕТ ДЛЯ ПОЛУЧЕНИЯ НОВОГО ЗАДАНИЯ!

Возможность Браузер	Video	Audio	Canvas	Web storage	Geolocation	Web Workers	Автономные веб-приложения
Firefox							
Safari							
Chrome							
Mobile WebKit							
Opera							
IE 6, 7							
IE 8							
IE 9							

Устройства под управлением операционных систем iOS и Android (среди прочих)

ВАША ПЕРВАЯ МИССИЯ: РАЗВЕДКА В СТАНЕ БРАУЗЕРОВ

**СОВЕРШЕННО
СЕКРЕТНО**

Мы немного схитрили при ответе и поставили флажки, ориентируясь на ситуацию, которая сложится к 2015 году. Но ваши ответы должны отражать положение вещей на момент чтения книги. Нам кажется, вам будет интересно заглянуть в будущее.

Браузер	Возможность						
	Video	Audio	Canvas	Web Storage	Geolocation	Web Workers	Автономные веб-приложения
Firefox	✓	✓	✓	✓	✓	✓	✓
Safari	✓	✓	✓	✓	✓	✓	✓
Chrome	✓	✓	✓	✓	✓	✓	✓
Mobile WebKit	✓	✓	✓	✓	✓	✓	✓
Opera	✓	✓	✓	✓	✓	✓	✓
IE 6, 7							
IE 8				✓			
IE 9	✓	✓	✓	✓	✓		

Даже если потребуется какое-то время па то, чтобы стапдарт HTML5 обрел повсеместную поддержку, вы будете знать, какие имеппо браузеры полностью его поддерживают, и пользоваться ими задолго до этого события. Фактически, мпогие из дапных возможностей уже поддерживаются современными браузерами. Вот почему пачать использовать HTML5 уже сейчас является отличной идеей. Кроме того, если вы пачпете прямо сейчас, то сможете про-извести впечатление па друзей и сослуживцев своими продвинутыми знаниями.

И не тяните, сделайте это поскорее!

Постойте-ка, если я начну использовать HTML5 прямо сейчас, разве не получится так, что я оттолкну пользователей старых браузеров? Или мне придется писать два варианта своих веб-страниц: один для браузеров с поддержкой HTML5, а другой — для старых версий браузеров?

Притормозите, сделайте глубокий вдох.

Прежде всего HTML5 — расширение предыдущей версии HTML, поэтому вам придется писать только один вариант своих веб-страниц. Вы правы в том, что возможности, поддерживаемые браузерами, могут отличаться в зависимости от того, насколько новой является версия веб-обозревателя и как часто ваши пользователи обновляют его. Таким образом, необходимо иметь в виду, что некоторые из новейших возможностей HTML5 могут не поддерживаться браузерами пользователей, и это возвращает нас к вопросу о том, что делать в таком случае.

Сейчас один из принципов HTML5-дизайна подразумевает, что своим страницам необходимо давать возможность плавно деградировать, то есть если браузер вашего пользователя не поддерживает какую-то новую функцию, вы должны позаботиться о достойной альтернативе для него. В книге мы покажем вам, как писать свои страницы с учетом этого.

Хорошая новость заключается в том, что все браузеры двигаются по направлению к стандарту HTML5 и связанным с ним технологиям (даже мобильные браузеры), в силу чего со временем плавная деградация станет скорее исключением, чем правилом (хотя вы всегда будете хотеть сделать все возможное, чтобы обеспечить для пользователей достойное взаимодействие со своими веб-страницами независимо от того, какие браузеры у них будут установлены).



Часть
Задаваемые
Вопросы

В: Я слышал, что стандарт HTML5 не получит статус финальной рекомендации до 2022 года! Это правда?

О: W3C — это организация по разработке стандартов, которая формально рекомендует стандарт HTML5. Входящие в W3C люди любят действовать осторожно, причем настолько, что предпочитают дожидаться, пока сменится несколько поколений HTML5-браузеров, прежде чем решиться сделать данный шаг. И это правильно, поскольку со стандартом все окончательно утрясется в ближайшие два года, а разработчики браузеров уверенно движутся по пути его реализации. Так что, да, может пройти какое-то время, прежде чем HTML5 обретет статус «финальной рекомендации». Ожидается, что он станет стандартом уже к 2014 году, и использовать HTML5 в практических целях следует начинать уже сейчас.

В: Что произойдет, когда все окончательно решится с HTML5?

О: Появится HTML6? Мы понятия не имеем, но, вероятно, что бы там ни было, оно придет к нам вместе с летающими автомобилями, ракетными костюмами и обедами в таблетках. Помните, что даже если мы перейдем на HTML6, `doctype` не изменится. Если предполагать, что W3C сдержит свое обещание и будущие версии HTML окажутся обратно совместимыми друг с другом, то мы сможем без проблем перейти на нечто новое, что будет следующим на очереди.

В: Chrome, Safari, Firefox, множество мобильных браузеров... вам не кажется, что ситуация лишь усугубляется? Будут ли наши страницы нормально работать во всех этих браузерах?

О: Несмотря на конкуренцию на рынке браузеров (для настольных компьютеров и мобильных устройств), на самом деле большинство из них основано всего на нескольких общих HTML-движках. Например, Chrome, Safari и мобильные браузеры для Android и iPhone базируются на WebKit, который представляет собой браузерный движок с открытым исходным кодом. Поэтому веб-страницы, по большей части, смогут успешно работать в разных браузерах.

В: А почему бы просто не использовать Flash, чтобы избежать проблем с межбраузерной поддержкой?

О: Flash — отличный инструмент, который получил повсеместное распространение в операционных системах и браузерах настольного сегмента. HTML5 со своим семейством технологий предлагает вам сделать с использованием открытых стандартов многое из того же, что и Flash. Что вам предпочтеть? Задумайтесь о том, какой объем инвестиций в технологии HTML5 вкладывают такие компании, как Google, Apple, Microsoft и др. В долгосрочной перспективе HTML5 станет крупным игроком, а в мобильном сегменте он уже является таковым. Выбор за вами, обе эти технологии будут в обиходе еще долгое время, индустрия движется по направлению к открытым стандартам.



HTML-археология



Мы провели раскопки и обнаружили код, вложенный в HTML-страницу. Надеемся, вы поможете нам взломать данный код и выяснить, что он означает. Мы не ожидаем от вас толкования этого кода, а просто пытаемся разогреть ваш мозг, заставив его немного порассуждать дедуктивным путем...

```
<script>
  var walksLike = "duck";
  var soundsLike = document.getElementById("soundslike");
  if (walksLike == "dog") {
    soundsLike.innerHTML = "Woof! Woof!";
  } else if (walksLike == "duck") {
    soundsLike.innerHTML = "Quack, Quack";
  } else {
    soundsLike.innerHTML = "Crickets...";
  }
</script>
```

Подсказка: `document` представляет целую HTML-страницу, а `getElementById`, возможно, имеет отношение к HTML-элементам и идентификаторам.

Я просто хочу сказать, что если вы намерены серьезно заняться созданием веб-приложений и использованием HTML5, то вам потребуются навыки работы с JavaScript.



У нас к вам есть разговор.

Если вы ранее прочитали книгу «Изучаем HTML, XHTML и CSS», то мы полагаем, что вы, вероятно, хорошо разбираетесь в использовании языков разметки и таблиц стилей для создания прекрасных веб-страниц. Ориентируясь в обеих этих технологиях, вы сможете преодолеть длинный путь...

С появлением HTML5 веб-страницы превращаются в насыщенные приложения, обладающие поведением, обновляются на лету и взаимодействуют с пользователями. Создание страниц подобного рода требует изрядного труда программиста, и если вы собираетесь писать код, который будет выполняться в браузере, то вам необходимо использовать *JavaScript*.

Если вам доводилось ранее заниматься программированием илиписанием простых сценариев, то вам это пригодится: JavaScript (несмотря на слухи) является фактически языком, и в этой книге мы поведаем вам обо всем, что необходимо знать для написания приложений. Если же у вас нет опыта программирования, мы сделаем все возможное, чтобы ввести вас в курс дела. В любом случае, одним из огромных преимуществ JavaScript является легкость его понимания для программистов-новичков.

Более увлекательного способа научиться программированию мы и представить не можем!

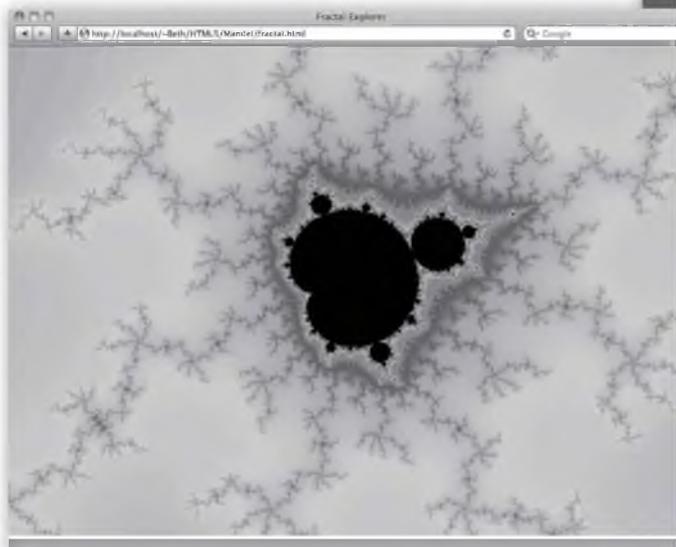
Итак, что теперь? Давайте кратко ознакомимся с JavaScript, а затем по-настоящему глубоко погрузимся в него в главе 2. Не старайтесь разобраться во всех деталях на протяжении нескольких следующих страниц — здесь вы должны лишь почувствовать, что такое *JavaScript*.

Что можно сделать с помощью JavaScript?

JavaScript открывает целый новый мир выражений и функциональностей для ваших веб-страниц. Давайте взглянем на ряд вещей, которые вы сможете сделать с помощью JavaScript и HTML5...

Используя HTML5 и JavaScript, вы сможете создать поддерживающую 2D-рисование поверхность прямо на своей странице, и при этом не потребуются никакие плагины.

Взаимодействуйте со своими страницами новыми способами, которые подходят как для настольного сегмента, так и для портативных устройств.



Внедрите в свои страницы поддержку определения местоположения, чтобы можно было узнать, где находятся ваши пользователи, показать им, что располагается поблизости, помочь отыскать то, что им нужно, задать им направление либо собрать людей с общими интересами в одном месте.

Используйте Web Workers, чтобы ускорить свой JavaScript-код и произвести важные вычисления либо сделать свои приложения более отзывчивыми. Вы даже можете еще эффективнее задействовать потенциал многоядерных процессоров, установленных на компьютерах ваших пользователей!



Получайте доступ к любой веб-службе и передавайте полученные от нее данные своему приложению почти в режиме реального времени.

Кэшируйте данные локально, используя браузерное хранилище, для ускорения работы мобильных приложений.

Для воспроизведения видео больше не нужны специальные плагины.

Интегрируйте свои страницы с Картами Google и дайте пользователям возможность отслеживать их собственное перемещение в режиме реального времени.

Создавайте собственные элементы управления воспроизведением, используя HTML и JavaScript.

Попрощайтесь с браузерными cookie-файлами и используйте локальное хранилище в браузере.

Используя JavaScript, вы сможете сохранять множество установок и данных для своих пользователей локально в браузере и даже сделать так, чтобы к ним имелся автономный доступ.

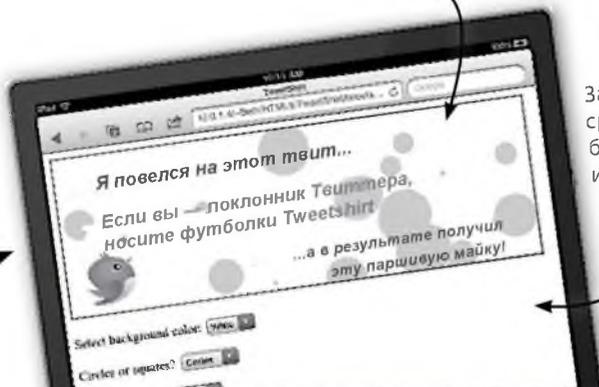
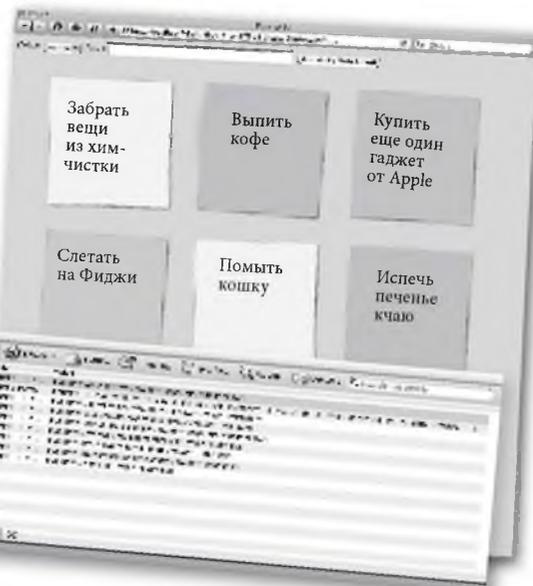
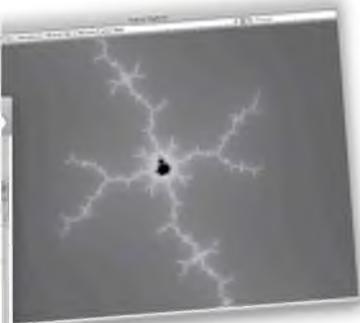
Браузер теперь не просто инструмент для просмотра скучных документов. Благодаря JavaScript вы сможете рисовать пиксели прямо в окне браузера.

Зарядите свои формы посредством JavaScript, чтобы обеспечить настоящую интерактивность.

Создавайте веб-страницы, которые по-новому объединяются с видео.

Используйте мощь JavaScript для тщательной обработки видео в своем браузере. Создавайте спец-эффекты и напрямую манипулируйте отдельными видеопикселями.

Вы, вероятно, подумали, что мы обшарили весь Интернет, чтобы найти наиболее захватывающие примеры. На самом деле это не так. Мы просто сделали скриншоты созданных нами примеров, с которыми вы будете сталкиваться в книге. Правда, они здорово выглядят? Итак, теперь, когда вы уже в Вебвилле, пришло время научиться местному языку — JavaScript. Что ж, приступим.





ВСТРЕЧАЕМСЯ С JAVASCRIPT

Интервью недели:
Признания языка сценариев

Head First: Добро пожаловать, JavaScript. Мы рады, что Вы смогли выкроить для нас время в своем плотном графике. Позвольте сразу спросить вот о чем: HTML5 превращается в эпатирующую, а как считаете Вас??

JavaScript: Я не стремлюсь быть в центре внимания, а остаюсь за кулисами. Я бы сказал, что небольшая часть похвалы, высказываемой в адрес HTML5, должна относиться ко мне.

Head First: Почему Вы так считаете?

JavaScript: Существует целое семейство технологий, которые делают работу «HTML5», куда, например, входят 2D canvas, localStorage, Web Workers и др. А правда заключается в том, что для того, чтобы действительно пользоваться ими, нужен я. Конечно, HTML5 позволяет создавать веб-страницы и представлять их вниманию пользователей, но без меня у людей не будет интереса к взаимодействию с ними вообще. Но все нормально. Желаю успеха HTML5, а я просто продолжу делать свою работу дальше.

Head First: Что бы Вы посоветовали разработчикам, решившим перейти на HTML5?

JavaScript: Ну, здесь все просто. Если вы действительно хотите овладеть HTML5, потратьте время на изучение JavaScript и всех библиотек, работающих с HTML5.

Head First: Знаете, у Вас не всегда была хорошая репутация. В одной из статей в 1998 году о Вас высказались так: «JavaScript — это незрелый, вычурный язык сценариев».

JavaScript: Это обидно. Может, я и не начал свою жизнь в безупречной академической среде многих языков программирования, но смог превратиться в один из наиболее широко используемых языков всех времен, поэтому на тот момент я бы не стал списывать меня со счетов столь опрометчиво. Кроме того, в то, чтобы сделать меня надежным и крайне эффективным языком, были вложены огромные ресурсы. Я стал быстрее по меньшей мере в 100 раз, чем был 10 лет назад.

Head First: Это впечатляет.

JavaScript: Да, и если Вы еще не слышали, то проинформирую Вас, что разработчики стандартов совсем недавно назвали меня языком сценариев по умолчанию для HTML5. Таким образом, я здесь, чтобы остаться надолго. Замечу, что программистам больше не нужно указывать JavaScript в своих тегах `<script>`. Может, меня и называли вычурным в 1998 году, но где теперь все эти JScript, VBScript, Java-апплеты и провалившиеся попытки с браузерными языками?

Head First: Что ж, Вы действительно являетесь ключом к созданию отличных HTML5-страниц. Но у Вас есть репутация языка, с которым возникает путаница.

JavaScript: Несмотря на слухи, я являюсь очень мощным языком, и чтобы успешно использовать меня, необходимо затратить некоторое время на изучение. С другой стороны, я популярен, потому что меня легко освоить. То есть я выбрал лучшее из обоих миров, как Вы считаете?

Head First: Похоже, что так оно и есть! Спасибо вам, JavaScript, за интервью.

JavaScript: Всегда пожалуйста.

Пишем серьезный JavaScript: проверка ваших ответов

Снова пройдитесь по коду и посмотрите, что привлекло ваше внимание. Вам нужно просто почувствовать этот код; далее пошагово разберемся во всех деталях.

```

var drink = "Energy Drink";

var lyrics = "";

var cans = 99;

while (cans > 0) {

    lyrics = lyrics + cans + " cans of "

        + drink + " on the wall <br>";

    lyrics = lyrics + cans + " cans of "

        + drink + "<br>";

    lyrics = lyrics + "Take one down,
        pass it around,<br>";

    if (cans > 1) {

        lyrics = lyrics + (cans-1) + " cans of "

            + drink + " on the wall <br>";

    }

    else {

        lyrics = lyrics + "No more cans of "

            + drink + " on the wall <br>";

    }

    cans = cans - 1;

}

document.write(lyrics);

```

Объявляем первую переменную и присваиваем ей значение "Energy Drink"

Объявляем вторую переменную и присваиваем ей пустое строковое значение.

Объявляем третью переменную и присваиваем ей числовое значение 99.

Это цикл while. Он говорит: «Пока количество банок превышает 0, выполнять все, что заключено в фигурные скобки. Остановиться, когда банок больше не останется»

Добавляем следующую строку песни в переменную lyrics с использованием оператора конкатенации строк «+»

Завершаем строку посредством разрыва строки HTML.

Повторяем снова.

Добавляем следующую строфу, снова с использованием конкатенации.

Если остались еще банки (то есть значение количества банок превышает 1)...

...то добавить последнюю строку.

в противном случае, когда банок не осталось...

... добавить "No more cans of " в конец lyrics.

Уменьшаем количество оставшихся банок на 1.

Мы сохранили все строки песни в переменной lyrics, поэтому теперь даем веб-странице команду записать ее. Это означает, что строка будет добавлена на страницу, в результате чего вы увидите текст песни.



ТЕСТ-АРАЙВ

Вы же не думали, что, провернув всю сложную работу по выполнению упражнения, вы так и не подвергнете практическому испытанию наш JavaScript-код? Вам необходимо взять код с предыдущей страницы и перенести его (вместе с HTML-разметкой, приведенной ниже) в файл (например, index.html), а затем загрузить в браузер. Результат можно увидеть внизу:

```

<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script>
    </script>
  </body>
</html>

```

← Введите этот код.

↑ Не забывайте, что для того, чтобы загрузить весь код и файлы примеров для книги, следует зайти по адресу <http://wickedlysmart.com/hfhtml5>.

← Теги <script> и </script> окружают JavaScript-код. Они говорят странице, что все, что в них заключено, является JavaScript-кодом, а не HTML.

← Сюда нужно вставить JavaScript-код с предыдущей страницы.

А вот результат нашего тестового прогона данного кода. Он генерирует текст лирической песни о 99 бутылках банках пива энергетического напитка на полке и записывает его в документ браузера.



Часть
Задаваемые
Вопросы

В: Почему в теле приведенного ранее HTML-документа нет ничего, кроме тегов `script`?

О: Мы решили начать с пустого элемента `body`, поскольку создали все содержимое данной страницы, используя JavaScript-код. Да, можно было бы просто внести текст лирической песни прямо в элемент `body` (и при этом нам бы пришлось долго печатать на клавиатуре), либо мы могли позволить коду выполнить всю тяжелую работу за нас (что мы и сделали), а затем дать ему команду вставить текст песни на страницу посредством `document.write`.

Имейте в виду, что здесь мы пока прощупываем почву, а по ходу книги будем тратить намного больше времени, рассматривая то, как можно динамически заполнять страницу содержимым с помощью кода.

В: Я понял, что мы сгенерировали весь текст лирической песни, но что конкретно сделал метод `document.write` и как текст попал в документ?

О: Метод `document.write` берет строку текста и вставляет ее в документ; фактически, он помещает ее точно туда, где располагается `ter script`. Таким образом, в данном случае `document.write` вставляет строку прямо в тело страницы.

Вскоре вы познакомитесь с более тонкими способами изменения текста живого документа с помощью JavaScript, а данный пример призван дать вам почувствовать, как код способен динамически вносить изменения в страницу.

В: Вы используете термины «веб-страница» и «веб-приложение»; под ними понимается что-то разное? Что именно делает нечто веб-приложением?

О: Это отличный вопрос, поскольку мы используем эти термины в широком смысле. Технически никакой разницы между двумя этими понятиями нет; другими словами, вам не нужно делать что-то особенное для того, чтобы превратить страницу, написанную на HTML, JavaScript и/или CSS, в веб-приложение. Появление различий — это скорее одна из возможных перспектив.

Когда у нас имеется страница, ведущая себя скорее как приложение, нежели как статичный документ, мы начинаем думать о ней больше как о веб-приложении и меньше как о веб-странице. Приложение мы рассматриваем как нечто такое, что обладает рядом особых качеств, например способностью поддерживать множество состояний, управлять более комплексными взаимодействиями с пользователем, отображать динамические и постоянно обновляемые данные без необходимости в обновлении всей страницы или даже выполнять более сложные задачи либо вычисления.

В: Весь этот JavaScript, конечно, замечательная вещь, но как насчет CSS? Мне не терпится воспользоваться преимуществами CSS3-нововведений, чтобы улучшить внешний вид своих страниц.

О: Да, CSS прошел долгий путь, и мы с воодушевлением смотрим на то, насколько хорошо он работает с HTML5. Несмотря на то что эта книга не о CSS, мы с вами обязательно воспользуемся преимуществами некоторых новых возможностей этого языка. Как вы, возможно, знаете, многие из трюков, к которым мы прибегаем для добавления закругленных углов и теней на изображениях при использовании HTML и создания простых анимаций на JavaScript, теперь могут быть с легкостью воспроизведены с помощью CSS. Так что в этой книге мы воспользуемся мощью CSS и обратим ваше внимание, когда это произойдет.



Мы поговорили о масса вещей, включая HTML-разметку, API-интерфейсы JavaScript, «семейство технологий» и CSS. А что именно представляет собой HTML5? Не может же он быть лишь простым языком разметки, который сумел пробудить всеобщий интерес...

Дадим вам неофициальный ответ:

Язык разметки +
API-интерфейсы JavaScript + CSS = $\overset{\text{HTML5}}{\underset{\uparrow}{\text{Суперклассная вещь}}}$

Отметим, что когда многие люди говорят о том, что такое HTML5, они подразумевают совокупность всех этих технологий. То есть у нас есть язык разметки для построения основной структуры страниц, JavaScript вместе со всеми своими API-интерфейсами для добавления нововедения и новой функциональности, а также CSS для стилизации страниц — все эти технологии мы будем использовать для создания веб-приложений завтрашнего дня.

Но почему же мы сказали «неофициальный»? Что ж, есть люди, которые любят проводить жесткие разграничительные линии между этими технологиями и говорить, к какому стандарту относится каждая из них. Это в порядке вещей и имеет место. Однако для нас важно вот что: какие технологии поддерживаются браузером и достаточно ли они проработаны для того, чтобы использоваться для создания наших страниц и приложений? На наш взгляд, HTML5 — это язык разметки + API-интерфейсы JavaScript + CSS, и мы считаем, что именно это, как правило, имеют в виду люди, когда говорят о HTML5 как о технологии.

↑ Если вам действительно интересно, как эти технологии объединяются в единый набор стандартов (а интересоваться это должно каждого), то советуем посетить ресурс w3.org для получения дополнительной информации по данному вопросу.

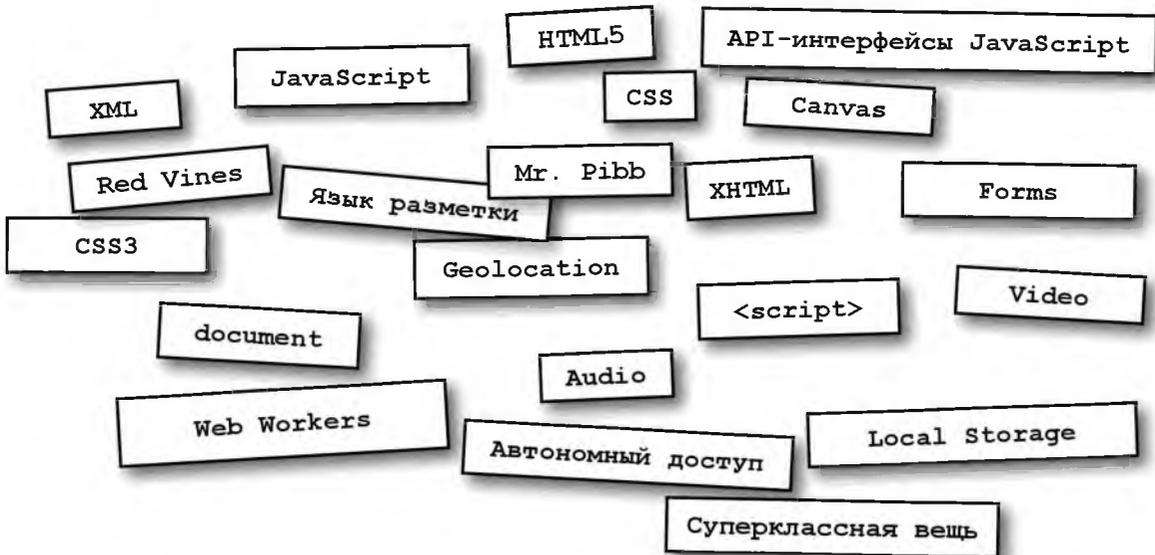


Поздравляем, вы закончили изучать главу 1 и написали свой первый код на HTML5!

← И свой первый код на JavaScript!

Прежде чем перейти к следующей главе, выполним еще одно задание на закрепление изученного материала. Используйте приведенные внизу карточки со словами для составления формулы, решающей уравнение «что такое HTML5?». Будьте внимательны, поскольку в эту кучу добавлены слова, которые могут сбить вас с толку. Справившись с задачей, немного отдохните и освежитесь, а затем приступайте к главе 2.

+ + =



КЛЮЧЕВЫЕ МОМЕНТЫ

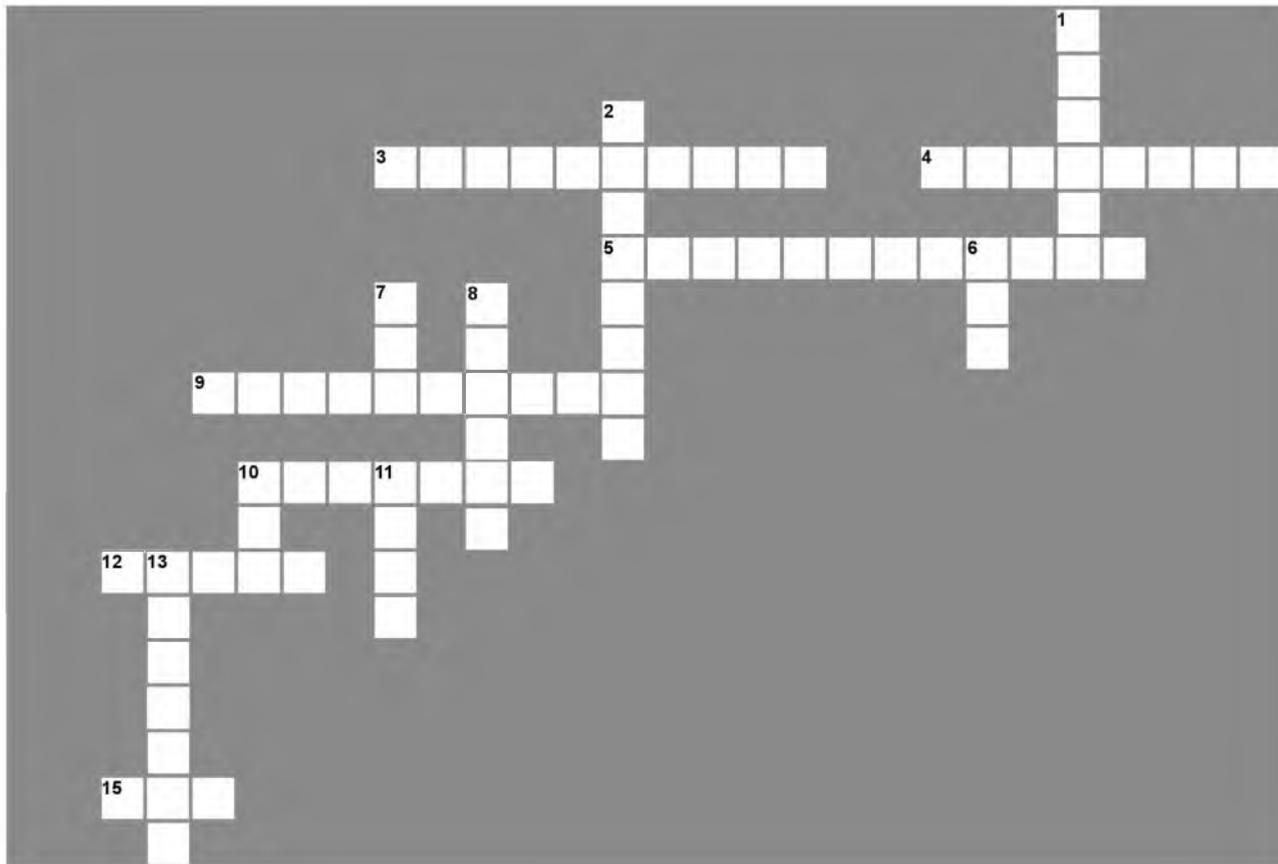


- HTML5 — это самая современная версия HTML. Она включает новые упрощенные теги, а также семантические и медиаэлементы, полагаются на набор JavaScript-библиотек, обеспечивающих функционирование веб-приложений.
- XHTML больше не является стандартом для веб-страниц. Вместо него разработчики и W3C решили продолжать расширять и совершенствовать HTML.
- Новое и более простое определение `doctype` в HTML5 поддерживается старыми версиями браузеров: когда они сталкиваются с ним, то переходят в стандартный режим.
- Атрибут `type` больше не требуется в теге `<script>` или в ссылке на таблицу стилей CSS. JavaScript и CSS стали языками по умолчанию для HTML5.
- Тег `<meta>`, используемый для указания набора символов, был упрощен и теперь включает только кодировку символов.
- UTF-8 сейчас является стандартной кодировкой символов, используемой в Интернете.
- Изменения в `doctype` и теге `<meta>` не скажутся отрицательно на страницах, загружаемых в старые версии браузеров.
- Совокупность новых элементов HTML5 представляет собой расширенный набор элементов HTML 4. Это означает, что старые страницы смогут нормально функционировать в современных браузерах.
- Работы над стандартом HTML5 не будут официально завершены до 2014 года, однако большинство современных браузеров станет поддерживать его задолго до этого (а многие поддерживают уже сейчас!).
- HTML5 включает элементы, которые привносят новую семантику в страницы, открывая перед вами больше возможностей, связанных с созданием структуры веб-страниц, чем было в HTML 4.01. Мы не будем рассматривать их в книге, однако в приложении вы найдете небольшое руководство по ним.
- Для использования многих возможностей HTML5 наилучшим образом вам потребуется JavaScript.
- Применяя JavaScript, вы сможете взаимодействовать с объектной моделью документа (Document Object Model, DOM).
- Объектная модель документа (DOM) — это браузерное внутреннее представление веб-страницы. Используя JavaScript, вы сможете получать доступ к элементам, изменять их, а также добавлять новые элементы в объектную модель документа.
- API-интерфейсы JavaScript (Application Programming Interface — интерфейс прикладного программирования) позволяют управлять всеми аспектами HTML5 (2D-рисованием, воспроизведением видео и др.).
- JavaScript является одним из наиболее популярных языков в мире. Реализации JavaScript значительно усовершенствовались за последние годы.
- Вы можете выяснять, поддерживается ли та или иная новая функция браузером, и обеспечивать плавную деградацию веб-страниц в случае отсутствия такой поддержки.
- CSS — это стандартный язык стилей для HTML5; многие люди вкладывают CSS в понятие «HTML5», когда используют его для описания семейства технологий, применяемых для создания веб-приложений.



HTML5-кроссворд

Настало время дать отдохнуть правому полушарию вашего мозга и заставить поработать левое. В приведенном ниже кроссворде все слова связаны с HTML5 и взяты из текущей главы.



По горизонтали

3. _____ реклама, также называемая спамом.
4. Ваша миссия заключалась в том, чтобы провести _____ в стане браузеров.
5. Инструмент, позволяющий обновить старый код до HTML5 за три шага _____.
9. Стандартный язык сценариев для HTML5.
10. Это определение теперь стало намного проще, чем было в версии HTML 4.01.
12. Этот язык получил «прощальное письмо» в 2009 году.
14. Используйте цикл _____ для генерирования вывода строк песни.
15. JavaScript стал быстрее в ___ раз, чем был 10 лет назад.

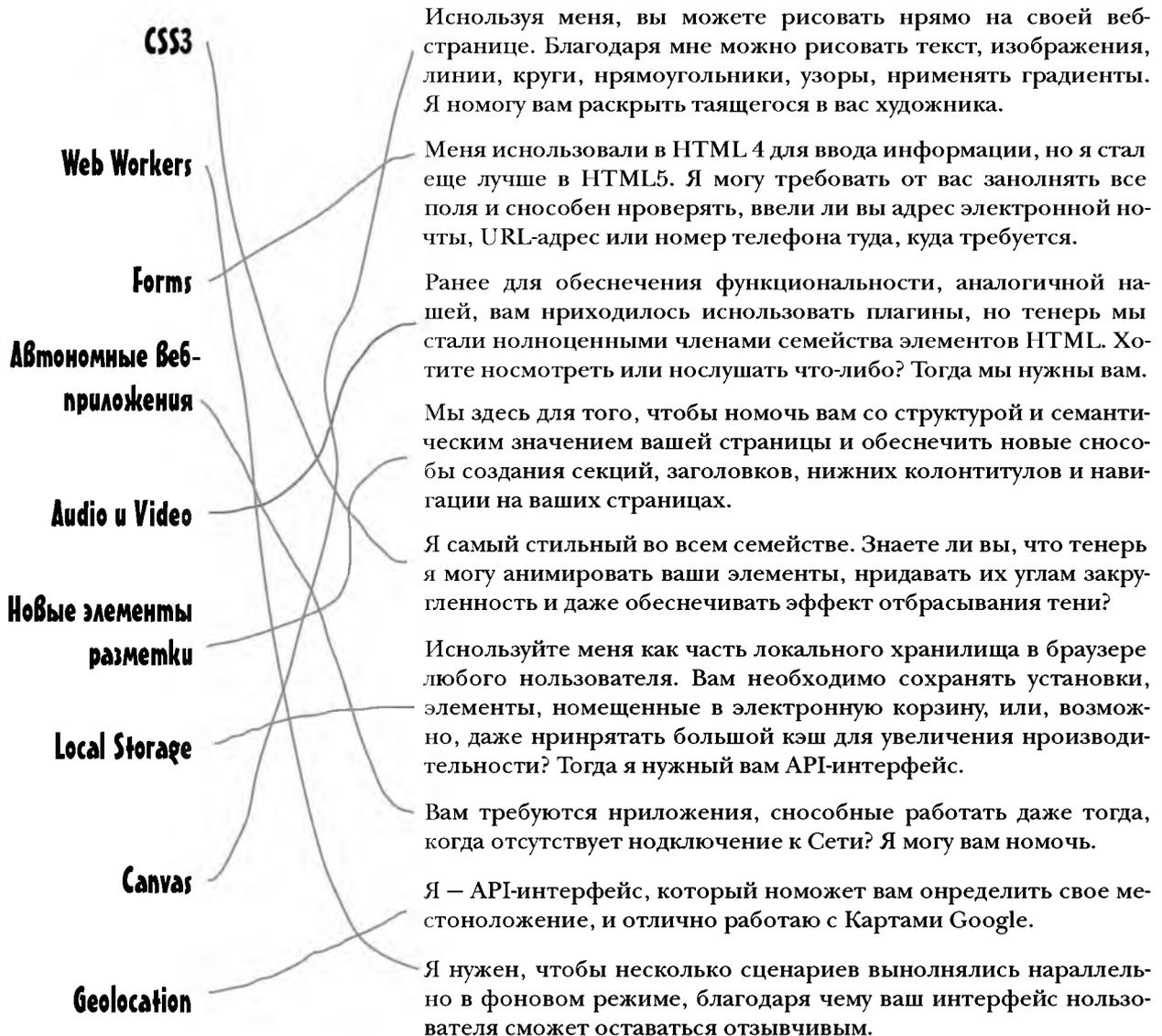
По вертикали

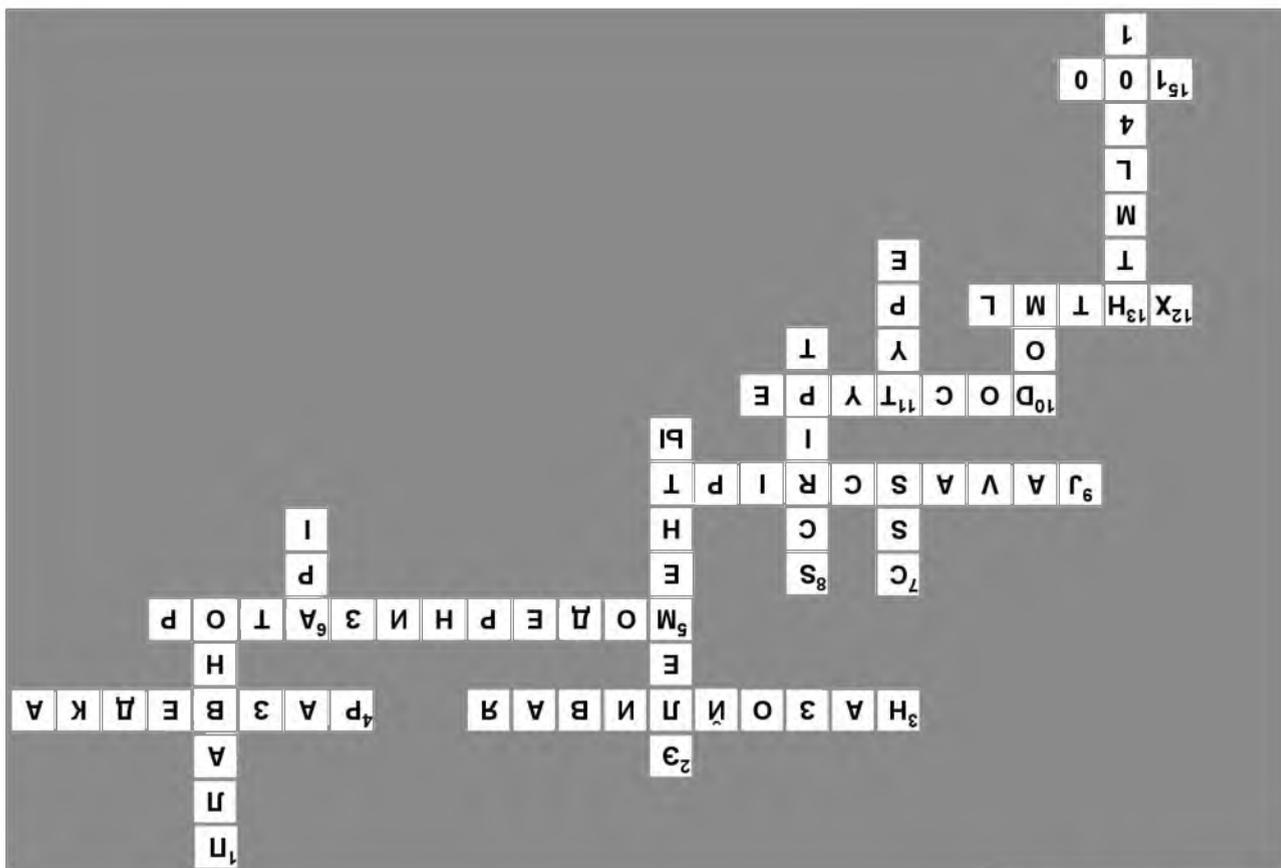
1. Нам необходимо, чтобы деградация наших веб-страниц происходила _____.
2. Новейшие _____ HTML5 привносят новую семантику и открывают дополнительные возможности, связанные с созданием структуры веб-страниц.
6. Истинная мощь HTML5 заключается в _____ JavaScript.
7. Стандартный язык стилей для HTML5.
8. Тег <_____> говорит браузеру, что все, что следует далее, является JavaScript-кодом, а не HTML.
10. _____ — это внутреннее представление веб-страницы.
11. Данный атрибут тегов `link` и `script` больше не требуется, если вы используете HTML5.
13. Версия HTML, предшествующая HTML5.

* КТО И ЧТО ДЕЛАЕТ? *

РЕШЕНИЕ

Мы с вами говорили о «семействе технологий» столько раз, что уже кажется, что мы сами одна семья. Однако мы по-настоящему так и не разобрались в том, что конкретно они собой представляют, поэтому, наконец, сделаем это. Ниже представлен перечень большинства членов этого семейства, посмотрите, сможете ли вы разобраться, кто есть кто. Не беспокойтесь, мы знаем, что это ваша первая встреча с членами семейства HTML5, поэтому приводим решения задания.





ИТМ5-кроссворд. Решение



2 знакомство с JavaScript и объектной моделью документа (DOM)

Немного кода



Благодаря JavaScript вы откроете для себя нечто новое. Вы уже все знаете о HTML-разметке (иначе называемой *структурой*) и CSS-стиле (также известном как *представление*), однако вам недостает знаний о JavaScript (или, как еще говорят, *поведении*). Если ваш багаж знаний ограничивается лишь структурой и представлением, то вы, конечно же, сможете создавать прекрасно выглядящие страницы, однако они будут лишь *простыми страницами*. Но если вы добавите поведение, прибегнув к JavaScript, то сможете обеспечить для своих пользователей интерактивное взаимодействие; либо, что даже еще лучше, вы сможете создавать роскошные веб-приложения. Приготовьтесь добавить в свой инструментарий веб-разработчика наиболее интересные и универсальные знания о JavaScript и программировании!

↑ А если вам нужна дополнительная мотивация, то и наиболее полезные!

Как работаем JavaScript

Наша цель заключается в том, чтобы написать JavaScript-код, который будет выполняться в браузере после загрузки веб-страницы. Данный код мог бы, например, реагировать на действия пользователя, обновлять или изменять страницу, общаться с веб-службами и, в целом, создавать ощущение, что ваша страница ведет себя скорее как приложение, нежели как обычный документ. Давайте посмотрим, как это все работает.

```
<html>
<head>
<script>
  var x = 49;
</script>
<body>
<h1>My first JavaScript</h1>
<p></p>
<script>
  x = x + 2;
</script>
</body>
</html>
```

Написание

1

Вы пишете свою разметку на HTML и код на JavaScript, а затем сохраняете их в файлах, скажем, `index.html` и `index.js` (либо и то и другое в одном HTML-файле).



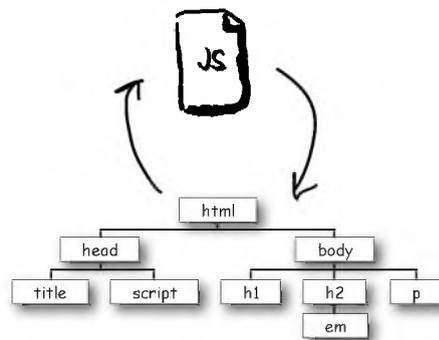
Загрузка

2

Браузер извлекает и загружает вашу страницу, осуществляя разбор ее содержимого сверху вниз.

Когда браузер обнаруживает JavaScript-код, он разбирает его и проверяет на правильность, после чего выполняет этот код.

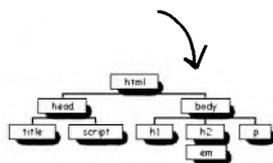
Браузер также создает внутреннюю модель HTML-страницы, называемую объектной моделью документа (DOM).



Выполнение

1

JavaScript-код продолжает выполняться, используя DOM для исследования страницы, ее изменения, получения от нее событий или запроса браузера на извлечение дополнительных данных с веб-сервера.



Что можно сделать с помощью JavaScript?

Если у вас имеется страница с элементом `<script>` (или со ссылкой на отдельный JavaScript-файл), то вы готовы приступить к написанию кода. JavaScript является полноценным языком программирования, с помощью которого можно сделать очень многое из того, что предоставляют другие языки, и даже еще больше, поскольку вы будете программировать внутри страницы!

JavaScript позволяет следующее.

1 Формировать операторы

Создавайте переменные и присваивайте им значения, складывайте числа, производите другие вычисления, используйте встроенную функциональность библиотек JavaScript.

```
var temp = 98.6;
var beanCounter = 4;
var reallyCool = true;
var motto = "I Rule";
temp = (temp - 32) * 5 / 9;
motto = motto + " and so do you!";
var pos = Math.random();
```

2 Делать что-то дважды, неоднократно

Выполнение операторов может осуществляться снова и снова, столько раз, сколько вам потребуется.

```
while (beanCounter > 0) {
    processBeans();
    beanCounter = beanCounter - 1;
}
```

2 Принимать решения

Пишите условный код, зависящий от состояния вашего приложения.

```
if (isReallyCool) {
    invite = "You're invited!";
} else {
    invite = "Sorry, we're at capacity.";
}
```



Объявление переменной

Переменные содержат данные. В случае с JavaScript они могут содержать массу различных вещей. Давайте объявим несколько переменных, содержащих данные.

```
var winners = 2;
var boilingPt = 212.0;
var name = "Dr. Evil";
var isEligible = false;
```

Целочисленные значения.

Или числовые значения с плавающей точкой.

Или строки символов (кратко «строки»).

Или логические значения (true или false).



Три шага по созданию переменной

① → ③ ②
 var scoops = 10;

- Первый шаг заключается в объявлении переменной, в данном случае scoops. Следует отметить, что JavaScript, в отличие от некоторых других языков, не требует указания типа переменной, а просто создает контейнер общего типа, в котором может содержаться масса вещей:



- Далее нам необходимо значение, которое будет размещаться в переменной. Указать значение можно несколькими способами:

```
var scoops = 10;
var scoops = totalScoops / people;
var scoops = Math.random() * 10;
```

Значение может быть литеральным, например числом или строкой.

Или значение может быть результатом оценки выражения.

Либо вы можете использовать внутренние функции из JavaScript-библиотек, например генератор случайных чисел, для создания значения. Более подробно об этой, а также о ваших собственных функциях мы поговорим позже.

Переменные — это контейнеры для значений. JavaScript-переменные не имеют строгих типов, поэтому любая из них может содержать числовое, строковое или логическое значение.

3 Наконец, у нас имеются переменная и значение (литеральное значение, например 10, или результат оценки выражения (вроде totalScoops / people). И все, что нам осталось сделать, — присвоить значение нашей переменной.



Создав переменную, вы, конечно же, сможете изменить ее значение в любой момент или даже присвоить ей значение, имеющее другой тип. Вот ряд примеров:

```
scoops = 5;
scoops = scoops * 10;
scoops = "Tired of being an integer";
```

Мы можем присвоить переменной scoops другое целочисленное значение. Или даже использовать scoops в выражении, которое изменит ее значение. В данном случае значение scoops будет равно 50.

В JavaScript даже есть значение, которое означает «значения нет». Оно называется null. О его использовании мы поговорим позже.

Либо мы можем изменить значение и тип переменной scoops, в данном случае — на строковое значение и соответственно тип. Но будьте осторожны, поскольку это может привести к большим проблемам в коде, если вы ожидаете, что scoops будет иметь числовое значение. Подробнее об этом — чуть позже.

Часто задаваемые вопросы

В: Каким будет значение моей переменной, если я просто напишу так:

```
var winner;
```

О: После выполнения данного оператора переменной winner будет присвоено значение undefined, которое является другим значением и типом JavaScript. О том, где и как его использовать, мы поговорим позже.



О синтаксисе

- Каждый оператор должен заканчиваться точкой с запятой.

```
x = x + 1;
```

- Однострочный комментарий должен начинаться двойным слешем. Комментарий — это просто заметки о коде, предназначенные для себя или для других разработчиков. Они никак не оцениваются.

```
// Это комментарий
```

- Пробелы неважны (почти везде).

```
x = 2233;
```

- Строки символов следует заключать в двойные кавычки.

```
"You rule!"
```

- Переменные необходимо объявлять с указанием var и имени. JavaScript не требует указывать тип в отличие от некоторых других языков программирования.

```
var width;
```

- Не заключайте в кавычки логические значения true и false.

```
rockin = true;
```

- Переменным необязательно присваивать значения при объявлении:

```
var width;
```

В: Мне доводилось иметь дело с другими языками программирования, в которых переменные объявляются с указанием типа. Например, int x или String y. Разве в JavaScript нет типов?

О: В JavaScript есть типы, однако, в отличие от других языков, он предусматривает динамическую типизацию, которая означает, что интерпретатор JavaScript сам определит, какой тип использовать.

Как присваивать имена переменным

У вас мог возникнуть вопрос: «А как выбирать имена для своих переменных?». Если вам доводилось присваивать имена идентификаторам в случае со своими HTML-элементами, то аналогичная процедура в отношении переменных покажется вам схожей. Есть только несколько правил при формировании имен переменных.

Правило 1. Начинайте имена своих переменных с буквы, символа подчеркивания или знака доллара.

При грамотном присваивании имен своим переменным необходимо не просто придумывать им выразительные названия, но также использовать букву (в нижнем или верхнем регистре), символ подчеркивания или знак доллара в начале их имен. Вот ряд примеров:



```
var thisIsNotAJoke;
var _myVariable;
var $importantVar;
```

↙
Правильно

Имя начинается с числа, а это неправильно.

Имя начинается с недопустимых символов (% и ~).

```
var 3zip;
var %entage;
var ~approx;
```

↘
Неправильно



Серьезное программирование

В JavaScript `number`, `string` и `Boolean` называются *примитивными типами*. В переменных также можно сохранять *объекты*. Об объектах мы поговорим довольно скоро, а пока вы можете рассматривать объект как коллекцию некоторых вещей, тогда как примитив — это просто вещь, которая не может быть разделена на что-либо еще.

Правило 2. Затем вы можете указывать любое количество букв, цифр, символов подчеркивания или знаков доллара.

Продолжайте использовать буквы, знаки доллара и символы подчеркивания при формировании имени своей переменной. После первого символа вы также при желании можете указывать числа:



```
var my3sons;
var cost$;
var vitaminB12;
```

↙
Правильно

Здесь имеется недопустимый пробел. Знаки «-» и «+» недопустимы и сильно сбивают с толку JavaScript.

```
var zip code;
var first-name;
var to+do;
```

↘
Неправильно



Правило 3. Избегайте всех зарезервированных слов JavaScript.

Язык JavaScript включает ряд зарезервированных слов, например `if`, `else`, `while`, `for` (это лишь часть из них), и не будет слишком любезен, если вы попытаетесь использовать их в качестве имен для своих переменных. Перечень зарезервированных слов JavaScript приведен ниже. Не нужно сию же минуту заново изучать их, но ходу освоения JavaScript у вас выработается понимание, что же они собой представляют. Но если когда-либо JavaScript станет «ругаться» на то, как вы объявили свои переменные, то вам следует мыслить следующим образом: «Хм, а может, слово, которое я пытаюсь использовать, является зарезервированным?».

abstract	delete	goto	null	throws
as	do	if	package	transient
boolean	double	implements	private	true
break	else	import	protected	try
byte	enum	in	public	typeof
case	export	instanceof	return	use
catch	extends	int	short	var
char	false	interface	static	void
class	final	is	super	volatile
continue	finally	long	switch	while
const	float	namespace	synchronized	with
debugger	for	native	this	
default	function	new	throw	



↪ Избегайте использования этих слов в качестве имен для переменных!

Часто задаваемые Вопросы

В: А что, если я использую зарезервированное слово как часть имени своей переменной? Могу ли я присвоить переменной, например, имя `ifOnly` (то есть в ее имени будет содержаться зарезервированное слово `if`)?

О: Конечно можете, просто избегайте точного совпадения имени переменной с зарезервированным словом. Также целесообразно писать понятный код, в котором, как правило, не используются слова вроде `elze`, которое можно перепутать с `else`.

В: Чувствителен ли JavaScript к регистру? Другими словами, есть ли разница между написанием `myvariable` и `MyVariable`?

О: Если вам доводилось иметь дело в основном с HTML-разметкой, то, возможно, вы привыкли к нечувствительным к регистру языкам, ведь, в конце концов, `<head>` и `<HEAD>` трактуются браузером как одно и то же. Однако в случае с JavaScript регистр имеет значение, поэтому `myvariable` и `MyVariable` будут рассматриваться как две разные переменные.

В: Как я понял, JavaScript позволяет присваивать нужное значение переменной (числовое, строковое и т. д.) в любой момент. Однако что произойдет, если я добавлю две переменные, одна из которых будет иметь числовое значение, а другая строковое, вместе?

О: JavaScript старается осуществлять умное преобразование типов по мере необходимости. Например, если вы добавите сразу две переменные с числовым и строковым значениями, то он, как обычно, попытается преобразовать числовое значение в строковое и конкатенировать их. Не во всех ситуациях это нужно. Держите эту мысль в уме, мы вернемся к ней совсем скоро.

Вебвилльское руководство по выбору оптимальных имен



Вам предоставляется немалая свобода при выборе имен для своих переменных, поэтому мы возьмем на себя смелость дать несколько советов, чтобы облегчить вам эту задачу.

Отдавайте предпочтение именам, которые что-то означают

Такие имена переменных, как, например, `_m`, `r` и `foo`, могут что-то означать для вас, однако в Вебвилле на них, как правило, смотрят неодобрительно. Не только потому, что вы сами, скорее всего, забудете их со временем, но и в силу того, что ваш код будет более удобочитаемым, если вы предпочтете использовать в нем переменные с именами вроде `angle`, `currentPressure` и `passed`.

При формировании имен переменных из нескольких слов используйте стиль CamelCase

В определенный момент у вас возникнет необходимость решить, как указать имя переменной, которая олицетворяет, скажем, огнедышащего двуглавого дракона. Просто используйте стиль CamelCase, который подразумевает, что первая буква в каждом новом слове должна указываться в верхнем регистре (кроме слова, идущего первым): `twoHeadedDragonWithFire`. Стиль CamelCase прост в использовании, широко распространен в Вебвилле и обеспечивает достаточную гибкость в формировании настолько специфических имен переменных, насколько вам потребуется. Существуют также и другие подходы, однако этот является одним из наиболее часто используемых (даже за пределами JavaScript).

Используйте переменные, имена которых начинаются с символов `_` и `$`, только по очень веской причине

Переменные, имена которых начинаются с `$`, обычно зарезервированы для JavaScript-библиотек и, несмотря на то что некоторые программисты используют переменные с именами, начинающимися с символа `_`, в силу различных соглашений, они не имеют широкого хождения. Мы рекомендуем держаться от них подальше, если только у вас не будет веской причины поступить иначе (если она появится, вы, естественно, будете это знать).

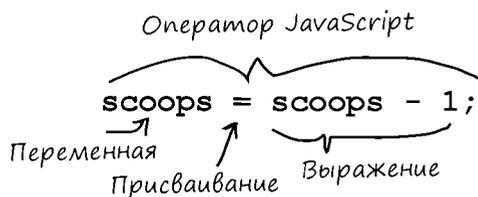
Будьте осторожны

Проявляйте осторожность при присваивании имен переменным. Далее в книге мы дадим еще ряд советов по этой теме, а пока знайте, что для переменных нужно выбирать понятные имена, избегать зарезервированных слов и всегда указывать `var` при объявлении переменной.



Выражения

Мы с вами уже видели, что представляют собой операторы JavaScript:



Однако давайте более пристально взглянем на выражения вроде того, которое приведено в данном операторе. Оказывается, выражения повсеместно встречаются в JavaScript, поэтому важно знать, какого рода вещи вы сможете выразить. Вот ряд примеров...

Вы можете писать выражения, результатом оценки которых будут числовые значения...

Числовые выражения

```
(9 / 5) * tempC + 32
x - 1
Math.random() * 10
2.123 + 3.2
```

Вы можете писать выражения, результатом оценки которых будут логические значения true или false (они соответственно называются логическими выражениями).

Логические выражения

```
2 > 3
tempF < 75
pet == "Duck"
startTime > now
level == 4
```

...а также писать выражения, результатом оценки которых будут строковые значения.

Строковые выражения

```
"super" + "cali" + youKnowTheRest
"March" + "21" + "st"
phoneNumber.substring(0, 3)
```

Также существуют другие типы выражений, о которых мы поговорим позже.

Прочие выражения

```
function () {...}
document.getElementById("pink")
new Array(10)
```

Внимательно относитесь к выражениям, представленным на нескольких следующих страницах (не говоря уже об оставшейся части книги), и вы поймете, как они используются для проведения вычислений, многократного повторения действий и принятия решений в вашем коде.



Упражнение

Выразите себя!

Ранее вы познакомились с различными типами выражений, которые можно использовать в JavaScript-коде. Теперь пора применить эти знания на практике и самим оценить несколько выражений. Свои ответы вы сможете проверить в конце главы.

```
(9 / 5) * tempC + 32
```

Каким окажется результат, если значением tempC будет 10?

```
"Number" + " " + "2"
```

Какова будет результирующая строка? _____

```
level >= 5
```

Каким окажется результат, если значением level будет 10?

А каким окажется результат, если значением level будет 5?

```
color != "pink"
```

Каким окажется результат, если значением color будет blue? _____

← Подсказка: «!»
означает «не».

```
(2 * Math.PI) * r
```

Каким окажется результат, если значением r будет 3?



Подсказка: Math.PI возвращает значение пи (оно, как вы знаете, равно 3,14...)



Так выражаться не стоит!

Возьми в руку карандаш



Взяв за основу свои текущие знания о переменных, выражениях и операторах JavaScript, посмотрите, сможете ли вы сказать, какие из этих операторов являются допустимыми, а какие приведут к выводу ошибки.

В приведенном ниже перечне обведите **допустимые** операторы.

```
var x = 1138;
var y = 3/8;
var s = "3-8";
x = y;
var n = 3 - "one";
var t = "one" + "two";
var 3po = true;
var level_ = 11;
var highNoon = false;
var $ = 21.30;
var z = 2000;
var isBig = y > z;
z = z + 1;
z--;
z y;
x = z * t;
while (highNoon) {
    z--;
}
```



Похоже, что все идет как по маслу, если я прибавляю числа к числам, а строки к строкам, но что, если я прибавлю число к строке или целочисленное значение к числу с плавающей точкой?

Помните, мы говорили, что программирование на JavaScript легко освоить? Одна из причин этого заключается в том, что JavaScript сам преобразует одни типы в другие по мере необходимости, чтобы выражения имели смысл.

В качестве примера допустим, что у вас имеется следующее выражение:

```
message = 2 + " if by sea";
```

Нам известно, что символ + может как использоваться для сложения чисел, так и выступать оператором конкатенации строк. Так какую же роль он выполняет в данном случае? Что ж, JavaScript знает, что строка " if by sea" никогда не станет похожей на число, поэтому он решает, что перед ним строковое выражение, преобразует 2 в строку "2" и присваивает переменной message значение "2 if by sea".

Либо, если у нас будет такой оператор:

```
value = 2 * 3.1;
```

JavaScript преобразует целочисленное значение 2 в число с плавающей точкой, а результат будет равен 6.2.

Как вы уже догадались, JavaScript не всегда делает то, что вам нужно, и в некоторых случаях ему требуется небольшая помощь в осуществлении преобразований. К этой теме мы вернемся немного позже.



Каков будет результат оценки следующих операторов со стороны JavaScript?

```
numORString1 = "3" + "4"
```

```
numORString2 = "3" * "4"
```

И почему?

```
while (juggling) {
    keepBallsInAir();
}
```



Множкратное выполнение одного и того же...

Если бы в JavaScript-программе все вынолялось лишь один раз, то она, скорее всего, была бы довольно скучной. Вы сами неоднократно делаете массу вещей: сноласкиваете голову водой, наносите шампунь, затем снова новторяете процедуру, пока ваши волосы не станут чистыми, либо продолжаете ехать до тех пор, пока не достигнете места назначения, либо непрерывно чернаете мороженое ложкой, пока не съедите его полностью. Для обработки подобных ситуаций JavaScript предусматривает несколько способов циклического выноления блоков кода.

Вы можете использовать JavaScript-цикл `while` для того, чтобы делать что-то до тех пор, пока удовлетворится соответствующее условие:

У нас имеется банка мороженого, в которой его осталось 10 ложек. Вот переменная, которую мы объявляем и инициализируем значением 10.

```
var scoops = 10;
```

```
while (scoops > 0) {
```

```
    alert("More icecream!");
```

```
    scoops = scoops - 1;
```

```
}
```

```
alert("life without ice cream isn't the same");
```

Цикл `while` использует логическое выражение, результатом оценки которого может быть `true` или `false`. В случае `true` код, расположенный далее, будет выполняться.

Пока ложек мороженого будет оставаться более нуля, мы продолжим выполнять все, что имеется в этом блоке кода.

При каждом выполнении цикла `while` мы уведомляем пользователя о том, что мороженое еще осталось, а затем удаляем одну ложку мороженого путем ее вычитания из общего числа ложек.

Когда значением условия (`scoops > 0`) становится `false`, цикл завершается, а выполнение кода продолжается здесь, независимо от того, какова будет следующая строка программы.



Таким образом, в примере с циклом `while` мы инициализируем значение, в данном случае – показатель количества оставшихся ложек мороженого, который *проверяется* циклом `while`, и если выдается `true`, то мы *выполняем* блок кода. В результате работы этого блока кода в определенный момент происходит *обновление* вовлеченного в проверку условия показателя до такого уровня, при котором *значением условия становится false* и цикл завершается.

```

var scoops = 10;
while (scoops > 0) {
    alert("More iccream!");
    scoops = scoops - 1;
}
alert("life without ice cream isn't the same");

```

← ИНИЦИАЛИЗИРОВАТЬ
 ← ОСУЩЕСТВЛЯТЬ ПРОВЕРКУ УСЛОВИЯ
 } ВЫПОЛНЯТЬ ДАННЫЙ БЛОК КОДА, ПОКА ПРОВЕРКА УСЛОВИЯ ВЫДАЕТ TRUE
 ← ОБНОВЛЯТЬ
 ← ПРОДОЛЖАТЬ ВЫПОЛНЕНИЕ КОДА ЗДЕСЬ ПОСЛЕ ТОГО, КАК ПРОВЕРКА УСЛОВИЯ ВЫДАСТ FALSE

JavaScript также предусматривает наличие цикла `for`, который еще больше формализует данную структуру. Вот как будет выглядеть наш код из примера с мороженым, переписанный с применением цикла `for`:

```

for (scoops = 10; scoops > 0; scoops--) {
    alert("There's more ice cream!");
}
alert("life without ice cream isn't the same");

```

ИНИЦИАЛИЗИРОВАТЬ
 ОСУЩЕСТВЛЯТЬ ПРОВЕРКУ УСЛОВИЯ
 ОБНОВЛЯТЬ
 ← ВЫПОЛНЯТЬ ДАННЫЙ БЛОК КОДА, ПОКА ПРОВЕРКА УСЛОВИЯ ВЫДАЕТ TRUE
 ← ПРОДОЛЖАТЬ ВЫПОЛНЕНИЕ КОДА ЗДЕСЬ ПОСЛЕ ТОГО, КАК ПРОВЕРКА УСЛОВИЯ ВЫДАСТ FALSE

Часто задаваемые Вопросы

В: Я не вижу никакой разницы между циклами `while` и `for`. Как понять, когда следует использовать каждый из них?

О: В целом, вы сможете выполнять одни и те же задачи, используя либо цикл `for`, либо цикл `while`. Однако, как вы могли убедиться в примере с мороженым, цикл `for` обеспечивает немного большую компактность, а `while` делает код более удобочитаемым. Как правило, циклы `for` чаще используются для совершения итераций по фиксированному количеству значений (например, по элементам, помещенным в электронную корзину в интернет-магазине), а `while` чаще применяются для циклического выполнения чего-либо до тех пор, пока удовлетворяется соответствующее условие (например, заставлять пользователя проходить тест до тех пор, пока он не сделает все правильно).

СТАНЬ браузером



Каждый из приведенных на этой странице JavaScript-фрагментов представляет собой отдельный блок кода. Ваша задача заключается в том, чтобы сыграть роль браузера и оценить все фрагменты кода для ответа на вопросы о результатах. Напишите свой ответ на каждый вопрос под соответствующим фрагментом.

Свои ответы вы сможете проверить в конце главы.

Фрагмент 2

```
var tops = 5;
while (tops > 0) {
  for (var spins = 0; spins < 3; spins++) {
    alert("Top is spinning!");
  }
  tops = tops - 1;
}
```

Сколько раз на экране появится диалоговое окно alert с сообщением "Top is spinning!"?

Фрагмент 4

```
for (scoops = 0; scoops < 10; scoop++) {
  alert("There's more ice cream!");
}
alert("life without ice cream isn't the same");
```

Сколько ложек мороженого вы съели?

Фрагмент 1

```
var count = 0;
for (var i = 0; i < 5; i++) {
  count = count + i;
}
alert("count is " + count);
```

Какой показатель общего количества будет отображен в диалоговом окне alert??

Фрагмент 3

```
for (var berries = 5; berries > 0; berries--) {
  alert("Eating a berry");
}
```

Сколько ягод вы съели?



Принятие решений с использованием JavaScript

Мы с вами использовали логические выражения в операторах `for` и `while` для проверки условий с целью решить, должно ли продолжаться циклическое выполнение. Их также можно использовать для принятия решений в JavaScript-коде. Рассмотрим пример:

```
if (scoops < 3) {
    alert("Ice cream is running low!");
}
```

Вот наше логическое выражение, используемое для проверки количества оставшихся ложек мороженого.

Когда останется менее трех ложек, мы будем выполнять соответствующий блок кода.

Мы также можем предусмотреть проведение более чем одной проверки:

```
if (scoops < 3) {
    alert("Ice cream is running low!");
} else if (scoops > 9) {
    alert("Eat faster, the ice cream is going to melt!");
}
```

Вы сможете добавить нужное количество проверок с использованием `else if`, каждая из которых будет ассоциирована со своим блоком кода, выполняемым, когда значением условия является `true`.

Принятие дополнительных решений... и добавление перехватывающего блока

Вы можете предусмотреть перехватывающий блок для своих операторов if – финальный else, который будет выполняться, если все прочие условия окажутся оценены как false. Давайте добавим еще несколько if/else, а также перехватывающий блок:

```

if (scoops == 3) {
    alert("Ice cream is running low!");
} else if (scoops > 9) {
    alert("Eat faster, the ice cream is going to melt!");
} else if (scoops == 2) {
    alert("Going once!");
} else if (scoops == 1) {
    alert("Going twice!");
} else if (scoops == 0) {
    alert("Gone!");
} else {
    alert("Still lots of ice cream left, come and get it.");
}
    
```

← Обратите внимание, что мы внесли изменение, из-за которого соответствующее сообщение будет выведено только тогда, когда количество ложек окажется равным именно 3.

← Мы добавили дополнительные условия для того, чтобы обратный отсчет шел до нуля ложек.

↑ А вот и наш перехватывающий блок; если ни одно из условий сверху не будет иметь значение true, то данный блок гарантированно выполнится.



Упражнение

Возьмите приведенный выше код и вставьте его в цикл while внизу. Пройдитесь по циклу while и напишите сообщения диалоговых окон alert в той последовательности, в которой они будут выводиться. Проверить свои ответы вы сможете в конце главы.

```

var scoops = 10;
while (scoops >= 0) {

    scoops = scoops - 1;

}
alert("life without ice cream isn't the same");
    
```

Вставьте сюда код, приведенный сверху.

↑ Генерируемый вывод напишите здесь.

Развлечения с магнитами



Данный код отображает известный палиндром в диалоговом окне alert. Проблема заключается в том, что часть кода находилась на магнитных табличках, прикрепленных к холодильнику, однако они упали на пол. Ваша задача заключается в том, чтобы восстановить целостность кода и отобразить палиндром. Будьте внимательны, поскольку на полу уже лежало несколько табличек, не имеющих отношения к данному коду, зато некоторые из табличек вам придется использовать более одного раза! Проверьте свои ответы в конце данной главы, прежде чем двинетесь дальше.

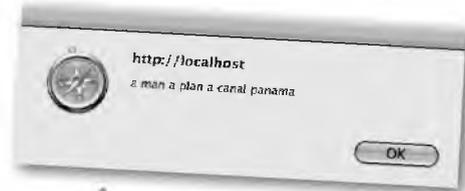
```

var word1 = "a";
var word2 = "nam";
var word3 = "nal p";
var word4 = "lan a c";
var word5 = "a man a p";

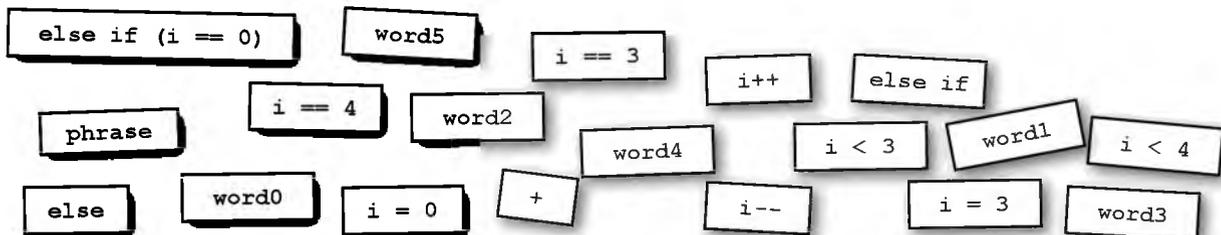
var phrase = "";

for (var i = 0; _____; _____) {
  if (i == 0) {
    phrase = _____;
  }
  else if (i == 1) {
    phrase = _____ + word4;
  }
  _____ (i == 2) {
    _____ = phrase + word1 + word3;
  }
  _____ (_____ ) {
    phrase = phrase + _____ + word2 + word1;
  }
}
alert(phrase);

```



Палиндром — это предложение, которое одинаково читается как слева направо, так и справа налево! Вот палиндром, который вы должны увидеть, если правильно разместите все таблички на магнитах по местам.





Мне сказали,
что мы будем добавлять
JavaScript в свои веб-страницы.
Когда мы, наконец, займемся этим?
Или так и будем ходить вокруг да около,
разбираясь в JavaScript?

Да, в том-то и дело. Для начала вам необходимо было изучить основы. Как раз этим мы и занимались до сих пор: теперь вы знаете, как объявлять и использовать переменные JavaScript, а также как осуществляется формирование базовых операторов и выражений. Кроме того, вы узнали, как использовать их все вместе для написания условного кода с операторами `if/else`, не говоря уже о циклическом выполнении с применением операторов `while` и `for`.

Вооружившись этими знаниями, теперь вы можете переходить к изучению того, как добавлять JavaScript в свои веб-страницы и, что более важно, как JavaScript взаимодействует с ними. То есть вы узнаете, как определять, что имеется на вашей странице, как ее изменять и, чуть нозже, как нисать код, реагирующий на то, что происходит на ваших страницах.

Таким образом, несмотря на то что мы еще не закончили с JavaScript, вашему ожиданию пришел конец. Настало время взглянуть, как разметка и новедение работают сообща...

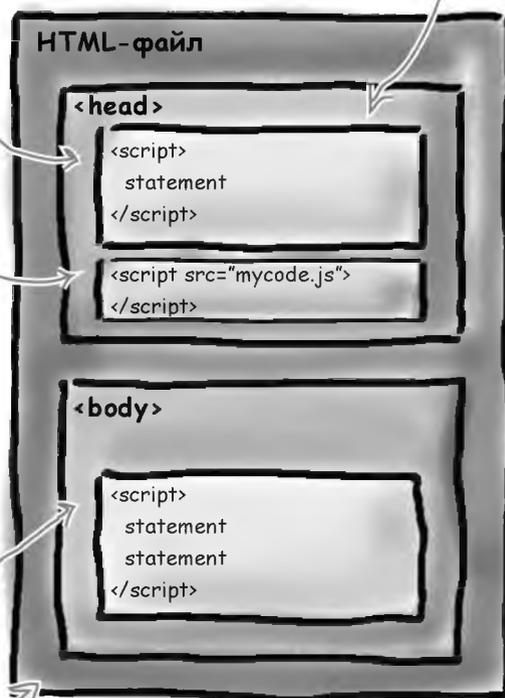
Как и куда добавлять JavaScript в своих страницах

Чтобы использовать JavaScript, его необходимо добавить в страницу. Но куда именно и как? Вы уже знаете, что существует такой элемент, как `<script>`, поэтому давайте посмотрим, где мы можем использовать его и как это влияет на выполнение JavaScript-кода на ваших страницах. Рассмотрим разные способы добавления кода в страницу.

Поместите элементы `<script>` в `<head>` своей HTML-страницы, чтобы их выполнение осуществлялось до загрузки страницы.

Вы можете внести требуемый код прямо в код своей веб-страницы либо внедрить ссылку на отдельный JavaScript-файл, воспользовавшись атрибутом `src` тега `script`

Или вы можете поместить свой код (либо ссылку на него) в элемент `<body>`. Данный код будет выполняться, когда загрузится тело документа.



Чаще всего код добавляется в `<head>` страницы. Добавление кода в `<body>` страницы дает небольшие преимущества в плане производительности, однако поступать так следует только в том случае, если вам действительно необходимо оптимизировать производительность своей страницы.

Разместите свой `<script>` как встроенный элемент в `<head>`.

Наиболее распространенный способ добавления кода в страницу заключается в помещении элемента `<script>` в `<head>` страницы. Если вы добавите JavaScript-код в элемент `<head>`, то он будет выполняться, как только браузер осуществит разбор `<head>` (а делает он это в первую очередь!), но до того, как разбору подвергнется остальная часть страницы.

Добавьте свой `<script>`, воспользовавшись ссылкой на отдельный JavaScript-файл.

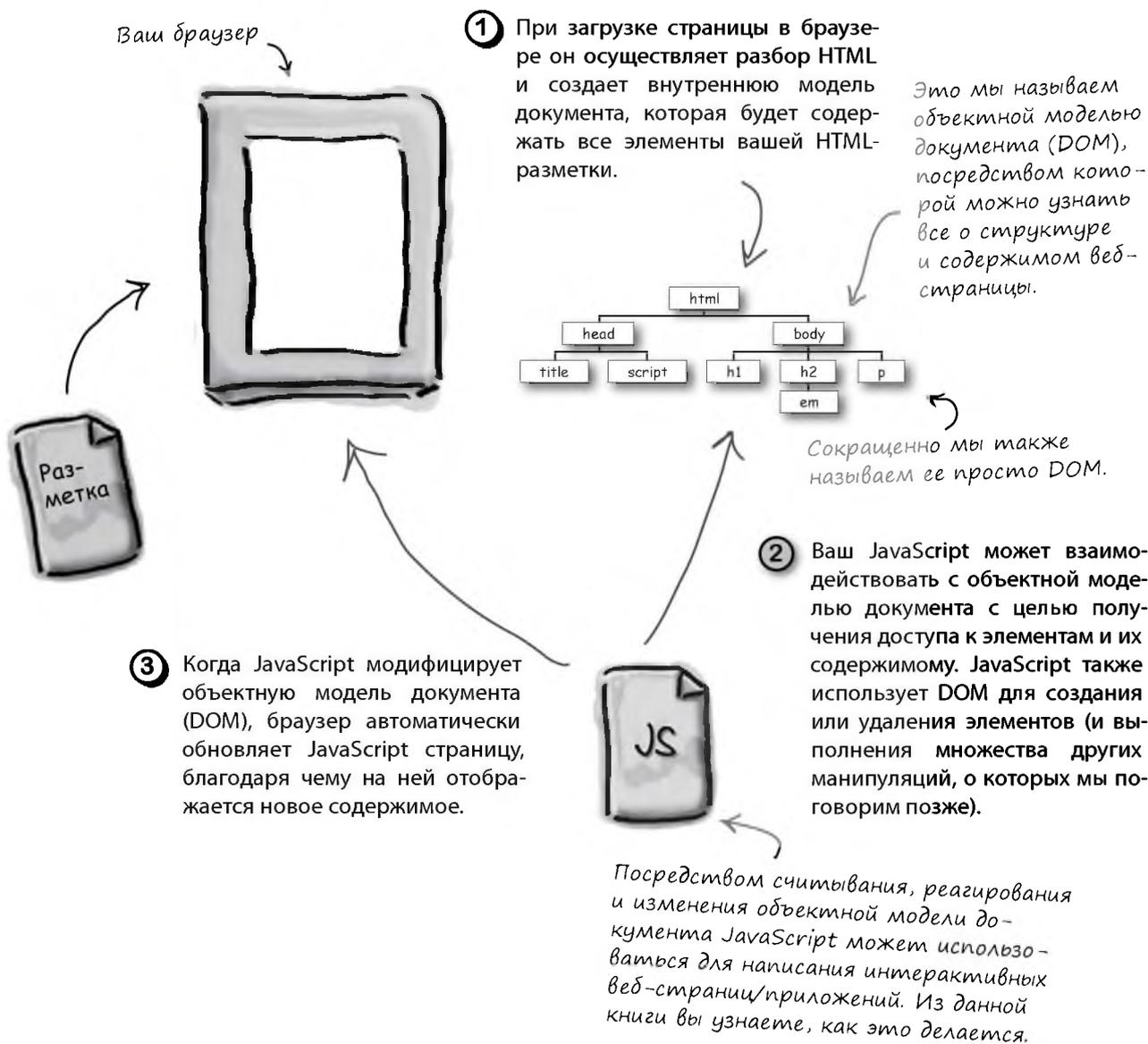
Вы также можете указать ссылку на отдельный файл, содержащий JavaScript-код. Поместите URL-ссылку на файл в атрибут `src` открывающего тега `<script>` и убедитесь, что вы закрыли элемент `script` с помощью `</script>`. Если вы указываете ссылку на файл, расположенный в том же каталоге, то можете просто использовать имя этого файла.

Добавьте свой код в элемент `<body>` документа либо как встроенный, либо посредством ссылки на отдельный файл.

Кроме того, вы можете поместить свой код прямо в `<body>` своей HTML-страницы. Опять-таки, заключите свой JavaScript-код в элемент `<script>` (или укажите ссылку на отдельный файл в атрибуте `src`). JavaScript в `<body>` вашей страницы станет выполняться, когда браузер будет осуществлять разбор тела страницы (обычно он делает это сверху вниз).

Как JavaScript взаимодействует с вашей страницей

JavaScript и HTML — это две разные вещи. HTML представляет собой разметку, а JavaScript — код. Так как же заставить JavaScript взаимодействовать с разметкой на своей странице? Для этого необходимо воспользоваться объектной моделью документа (DOM).



Рецепт приготовления собственной объектной модели документа (DOM)

Давайте возьмем разметку и создадим для нее объектную модель документа (DOM). Рецепт здесь прост.

Ингредиенты

Одна нормальная HTML5-страница

Один веб-браузер или более



Порядок действий

1. Сначала создайте узел `document`, который будет располагаться в самом верху.

document

2. Затем возьмите элемент верхнего уровня вашей HTML-страницы (в нашем случае это элемент `<html>`), который будет именоваться *текущим элементом*, и добавьте его в качестве дочернего элемента по отношению к `document`.

document

html

3. В случае с каждым элементом, вложенным в текущий элемент, добавьте соответствующий элемент в качестве дочернего по отношению к текущему в объектной модели документа (DOM).

document

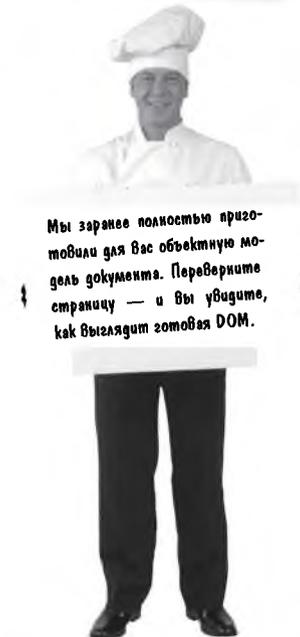
html

head

body

4. Вернитесь к каждому из только что добавленных вами элементов и повторите процедуру (шаг 3), пока не добавите все необходимые элементы.

```
<!doctype html>
<html lang="en">
<head>
  <title>My blog</title>
  <meta charset="utf-8">
  <script src="blog.js"></script>
</head>
<body>
  <h1>My blog</h1>
  <div id="entry1">
    <h2>Great day bird watching</h2>
    <p>
      Today I saw three ducks!
      I named them
      Huey, Louie, and Dewey.
    </p>
    <p>
      I took a couple of photos...
    </p>
  </div>
</body>
</html>
```



Первое испытание объектной модели документа (DOM)

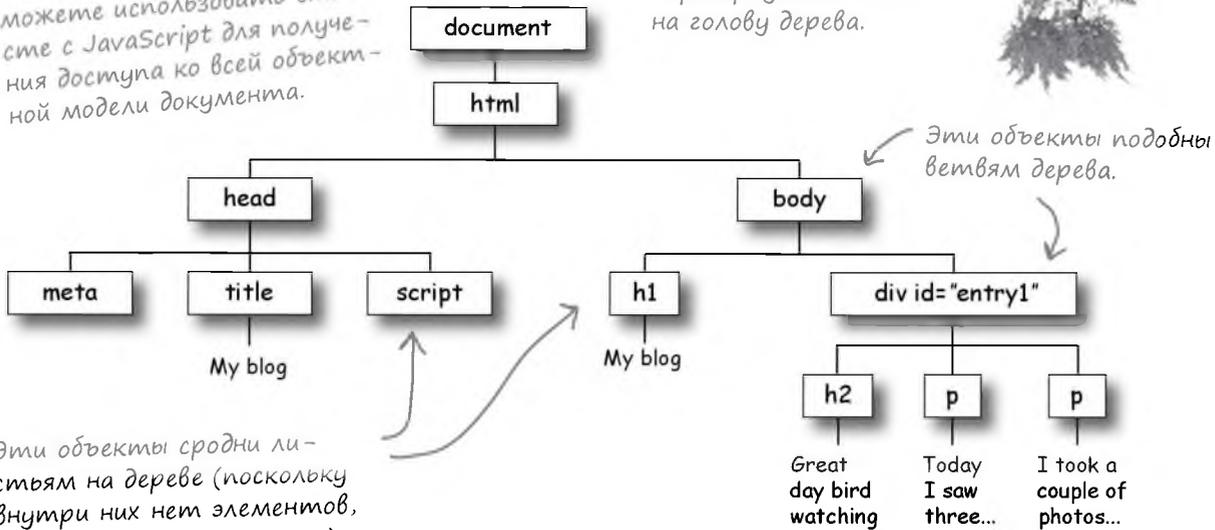
Вся прелесть объектной модели документа заключается в том, что она обеспечивает для нас согласованный между всеми браузерами способ получения доступа к структуре и содержимому HTML из кода. И это здорово. А через несколько мгновений мы с вами посмотрим, как все это работает...

Вернемся к примеру. Если вы последуете приведенному ранее рецепту приготовления объектной модели документа, то в результате у вас получится структура, изображенная чуть ниже. Каждая DOM включает объект document, располагающийся вверху, после чего общее дерево дополняют ветви и узлы-«листья» для каждого элемента в HTML-разметке. Давайте взглянем на них более пристально.

Мы сравнили данную структуру с деревом, поскольку «дерево» — это структура данных, которая берет свое начало в информатике.

Наверху всегда располагается document. Это особая часть дерева, которую вы можете использовать вместе с JavaScript для получения доступа ко всей объектной модели документа.

Объект document — это словно корень перевернутого с ног на голову дерева.



Эти объекты подобны ветвям дерева.

Эти объекты сродни листьям на дереве (поскольку внутри них нет элементов, а имеется только текст).

Объектная модель документа включает содержимое страницы, а также элементы (мы не всегда показываем текстовое содержимое, когда приводим рисунки DOM, однако оно там присутствует).

Теперь, когда у нас имеется объектная модель документа (DOM), мы можем исследовать и изменять ее так, как нам потребуется.



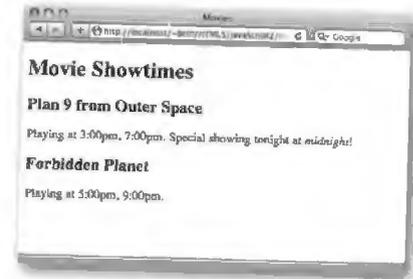


СТАНЬ браузером

Ваша задача заключается в том, чтобы сыграть роль браузера. Вам необходимо осуществить разбор HTML-разметки и создать на ее основе свою собственную объектную модель документа (DOM). Поэтому — вперед, произведите разбор HTML, который находится справа, а DOM нарисуйте внизу. Началом объектной модели документа мы уже нарисовали, а Вам предстоит ее завершить.



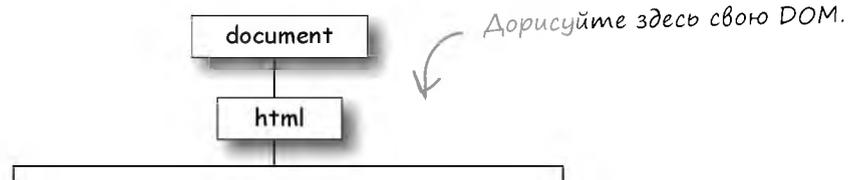
Проверьте свои ответы, посмотрев решение этого задания в конце главы, прежде чем двинетесь дальше.



```

<!doctype html>
<html lang="en">
  <head>
    <title>Movies</title>
  </head>
  <body>
    <h1>Movie Showtimes</h1>
    <h2 id="movie1" >Plan 9 from Outer Space</h2>
    <p>Playing at 3:00pm, 7:00pm.
    <span>
      Special showing tonight at <em>midnight</em>!
    </span>
    </p>
    <h2 id="movie2">Forbidden Planet</h2>
    <p>Playing at 5:00pm, 9:00pm.</p>
  </body>
</html>

```





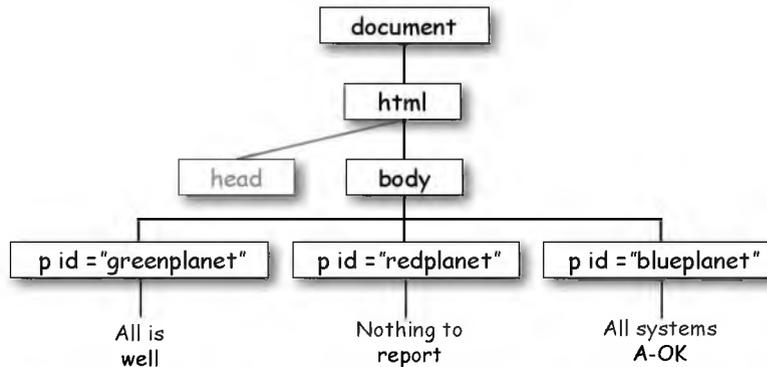
Или история о том, как две абсолютно разные технологии смогли работать сообща.

HTML и JavaScript, несомненно, прилетели с двух разных планет. Доказательства? ДНК HTML состоит из декларативной разметки, позволяющей описывать набор вложенных элементов, входящих в состав веб-страницы. JavaScript, с другой стороны, сделан из чисто алгоритмического генетического материала, призванного описывать вычисления.

Настолько ли они разные, что даже не способны взаимодействовать друг с другом? Конечно же, это не так, поскольку у них есть кое-что общее — объектная модель документа (DOM). Посредством DOM JavaScript может взаимодействовать с веб-страницами, и наоборот. Существует несколько путей сделать так, чтобы это произошло, однако мы пока сконцентрируемся на одном из них — на своего рода небольшой червоточине, позволяющей JavaScript получать доступ к любому элементу, и называется она `getElementById`.

Давайте посмотрим, как она работает...

Давайте начнем с DOM. Чуть ниже приведен пример простой объектной модели документа. Здесь имеется несколько HTML-параграфов (<p>), каждый из которых обладает id со значением соответственно greenplanet, redplanet и blueplanet. Все параграфы также содержат текст. Конечно, здесь есть и элемент <head>, однако мы отбросили детали в целях простоты.



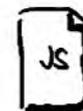
Теперь давайте задействуем JavaScript, чтобы стало интереснее. Допустим, нам необходимо изменить текст параграфа с id в виде greenplanet с "All is well" на "Red Alert: hit by phaser fire!". В будущем вам может потребоваться нечто подобное, в зависимости от действий, предпринимаемых пользователем, или от данных веб-службы. Обо всем этом мы еще поговорим, а пока обновим текст параграфа с id в виде greenplanet. Вот код, который позволит нам это сделать:

Как вы помните, document представляет всю страницу в браузере и целиком содержит объектную модель документа, поэтому мы можем «попросить» его что-либо сделать: например, найти элемент с определенным значением id.

Здесь мы просим document отыскать элемент, значение id которого соответствует заданному.

```
document.getElementById("greenplanet");
```

getElementById("greenplanet") возвращает элемент <p>, значение id которого соответствует "greenplanet"...



...после чего JavaScript-код сможет сделать с ним много чего интересного.

Как только `getElementById` возвратит требуемый элемент, вы сможете сделать с ним что-нибудь (например, изменить его текст на "Red Alert: hit by phaser fire!"). Для этого обычно требуется присвоить элемент переменной, благодаря чему на него можно будет ссылаться повсюду в своем коде. Давайте сделаем это, а затем изменим текст:

Мы присваиваем элемент переменной с именем `planet`.

Здесь мы вызываем `getElementById`, который отыщет и возвратит элемент "greenplanet".

```
var planet = document.getElementById("greenplanet");
```

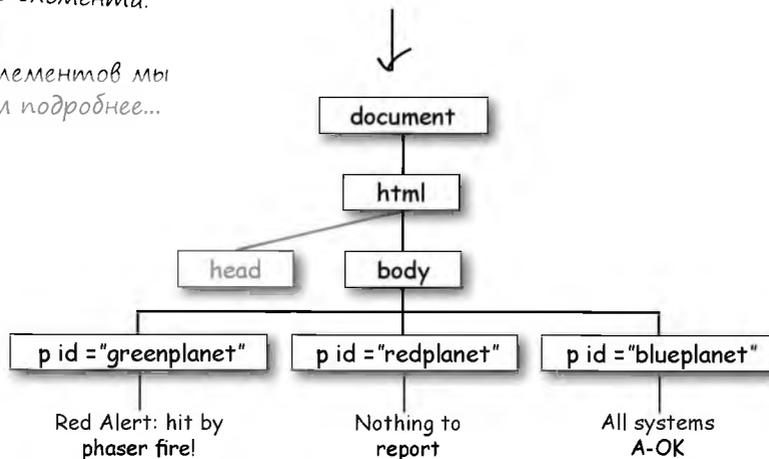
Теперь мы можем использовать в своем коде переменную `planet` для ссылки на наш элемент.

```
planet.innerHTML = "Red Alert: hit by phaser fire!";
```

Мы можем использовать свойство `innerHTML` нашего элемента `planet` для изменения содержимого требуемого элемента.

Мы заменяем содержимое элемента `greenplanet` на наш новый текст... в результате чего объектная модель документа (и веб-страница) будет обновлена с использованием этого нового текста.

О свойствах элементов мы вскоре поговорим подробнее...



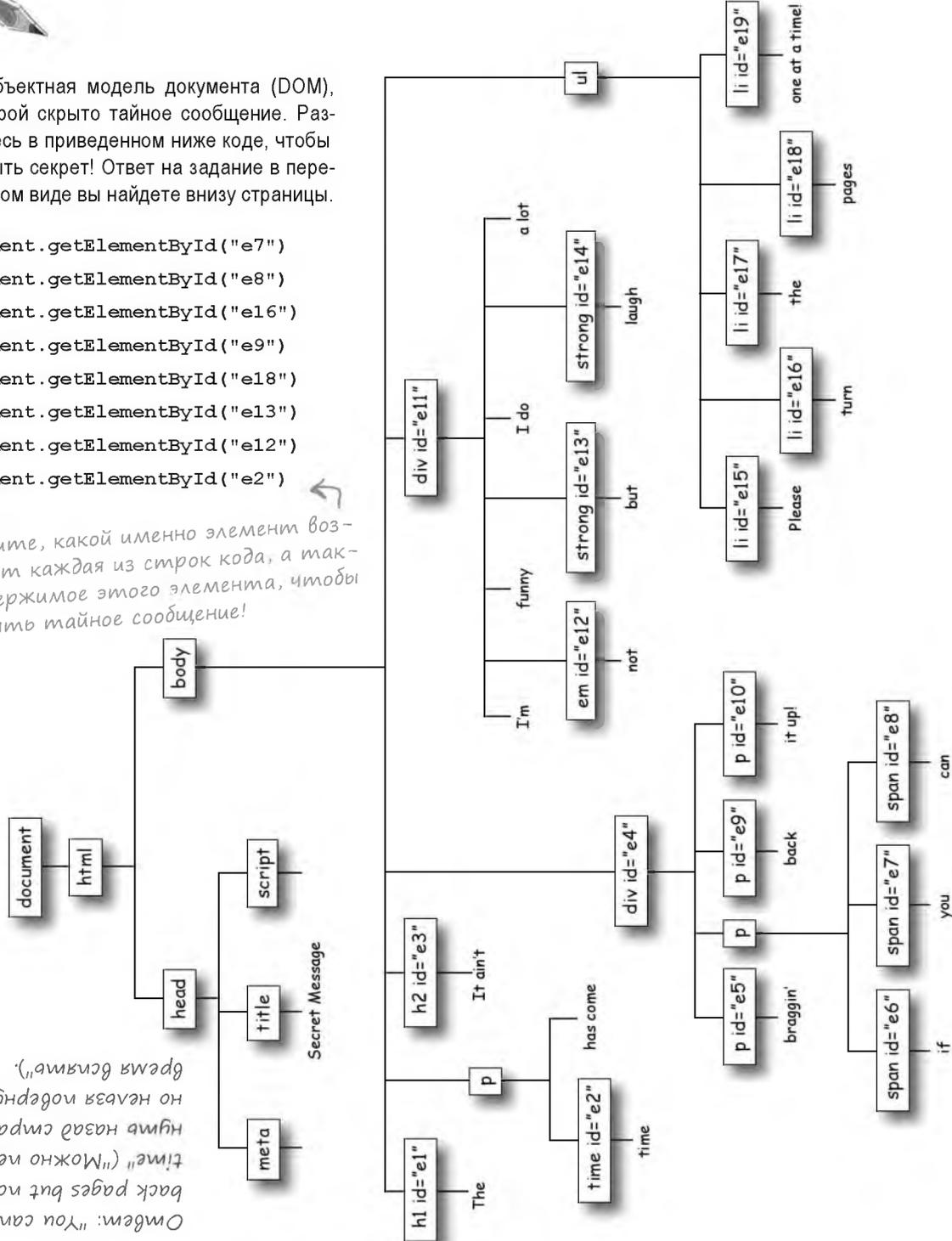
Любые изменения в объектной модели документа отражаются на том, как браузер осуществляет рендеринг страницы, поэтому вы увидите, что содержимое параграфа стало другим!

Вот объектная модель документа (DOM), в которой скрыто тайное сообщение. Разберитесь в приведенном ниже коде, чтобы раскрыть секрет! Ответ на задание в перевернутом виде вы найдете внизу страницы.

```
document.getElementById("e7")
document.getElementById("e8")
document.getElementById("e16")
document.getElementById("e9")
document.getElementById("e18")
document.getElementById("e13")
document.getElementById("e12")
document.getElementById("e2")
```

Напишите, какой именно элемент возвращает каждая из строк кода, а также содержимое этого элемента, чтобы раскрыть тайное сообщение!

Омдем: "You can turn back pages but not time" ("Можно перевернуть назад страницы, но нельзя повернуть время вспять").



Тест-драйв планет



Ранее вы уже видели, как использовать `document.getElementById` для получения доступа к элементу, а `innerHTML` — для изменения его содержимого. Теперь давайте сделаем это по-настоящему.

Ниже приведена HTML-разметка веб-страницы `Planets`; здесь у нас имеется элемент `<script>` в `<head>`, куда мы будем помещать код, и три параграфа со значениями `id` соответственно `greenplanet`, `redplanet` и `blueplanet`. Если вы еще этого не сделали, то добавьте HTML и JavaScript для обновления объектной модели документа (DOM):

```

<!doctype html>
<html lang="en">
<head>
  <title>Planets</title>
  <meta charset="utf-8">
  <script>
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
  </script>
</head>
<body>
  <h1>Green Planet</h1>
  <p id="greenplanet">All is well</p>
  <h1>Red Planet</h1>
  <p id="redplanet">Nothing to report</p>
  <h1>Blue Planet</h1>
  <p id="blueplanet">All systems A-OK</p>
</body>
</html>

```

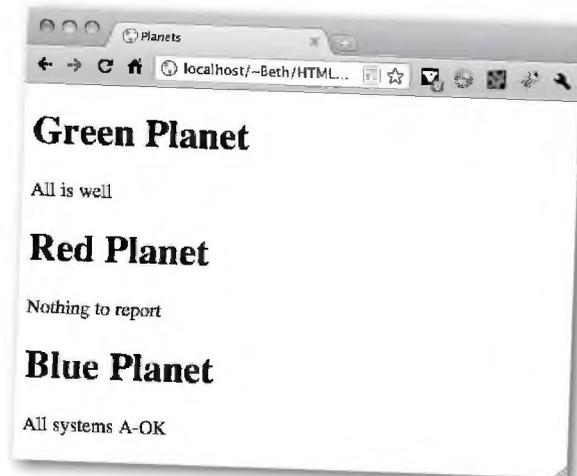
Мы добавили JavaScript в `<head>` страницы.

Точно так же, как и раньше, мы извлекаем элемент `<p>` с `id` в виде "greenplanet" и изменяем его содержимое.

Элемент `<p>`, содержимое которого мы изменяем с помощью JavaScript.

Добавив требуемый код, загрузите страницу в своем браузере, после чего вы увидите, какие волшебные метаморфозы произошли с параграфом, имеющим `id` в виде `greenplanet`, в объектной модели документа (DOM).

О! Хьюстон, у нас проблема: в параграфе с `id` в виде `greenplanet` по-прежнему отображается "All is well". В чем дело?





Я трижды перепроверил свою разметку и код, но все равно ничего не получается. Я не вижу никаких изменений на своей странице.

Ах да, мы забыли упомянуть об одной вещи.

Чаще всего имеет смысл начинать выполнение своего JavaScript-кода *после* того, как страница полностью загрузится. Почему? Если же вы не станете ждать, пока закончится загрузка страницы, объектная модель документа не успеет полностью сгенерироваться, когда ваш код начнет выполняться. В нашем случае выполнение JavaScript-кода начинается после того, как браузер сначала загрузит элемент `<head>` страницы, по до того, как будет загружена остальная часть страницы, поэтому объектная модель документа окажется еще не полностью сгенерированной. А если DOM не готова, то и элемент `<p id="greenplanet">` не будет существовать!

И что же тогда происходит? Вызов `getElementById` с целью поиска элемента с `id` в виде `greenplanet` не приведет к возврату чего-либо, поскольку соответствующий элемент будет отсутствовать, из-за чего браузер просто продолжит двигаться дальше и так или иначе произведет рендеринг страницы после того, как ваш код выполнится. Поэтому в результате вы увидите страницу, прошедшую рендеринг, по текст в параграфе с `id` в виде `greenplanet` останется прежним.

Нам необходимо как-то сообщить браузеру следующее: «Выполнять мой код после того, как страница полностью загрузится и будет создана объектная модель документа». Посмотрим, как это сделать.

Нельзя начинать взаимодействовать с DOM, пока веб-страница не загрузилась полностью

Но как приказать браузеру выполнять ваш код только после того, как страница загрузится? Чтобы дать указание браузеру ожидать, прежде чем начинать выполнение кода, мы воспользуемся двумя JavaScript-инструментами, с которыми мы вас еще не успели познакомиться: это объект `window` и функция. Подробнее о них мы поговорим позже, а пока просто воспользуемся ими, чтобы добиться нужного эффекта.

Обновите свой JavaScript, как показано ниже

```
<script>
function init() {
    var planet = document.getElementById("greenplanet");
    planet.innerHTML = "Red Alert: hit by phaser fire!";
}
window.onload = init;
</script>
```

Сначала создайте функцию с именем `init` и поместите в нее уже имеющийся у вас код.

Обратите внимание на то, что ваш код должен располагаться между открывающей и закрывающей фигурными скобками.

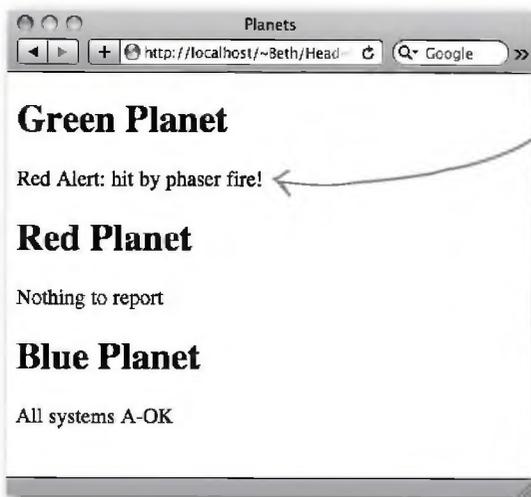
Здесь мы задаем значение свойства `window.onload` в виде имени нашей функции.

Здесь говорится: «когда страница полностью загрузится, выполнить код, расположенный в `init`».

Перезагрузите страницу



Теперь перезагрузите страницу и посмотрите, все ли встало на свои места:



Да! Теперь в элементе `<p>` с `id` в виде `greenplanet` отобразилось новое содержимое. Здорово, не правда ли?

Что ж, на самом деле здорово то, что теперь вы знаете, как приказать браузеру ждать, пока не будет полностью сгенерирована объектная модель документа, прежде чем выполнять код, который обращается к элементам.

Возьми в руку карандаш



```
<!doctype html>
```

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

```
<head>
```

```
<title>My Playlist</title>
```

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

```
<script>
```

```
_____ addSongs() {
```

```
var song1 = document._____("_____");
```

```
var _____ = _____("_____");
```

```
var _____ = _____.getElementById("_____");
```

```
_____.innerHTML = "Blue Suede Strings, by Elvis Pagely";
```

```
_____ = "Great Objects on Fire, by Jerry JSON Lewis";
```

```
song3._____ = "I Code the Line, by Johnny JavaScript";
```

```
}
```

```
window._____ = _____;
```

```
</script>
```

```
</head>
```

```
<body>
```

```
<h1>My awesome playlist</h1>
```

```
<ul id="playlist">
```

```
<li id="song1"></li>
```

```
<li id="song2"></li>
```

```
<li id="song3"></li>
```

```
</ul>
```

```
</body>
```

```
</html>
```

HTML для веб-страницы.

Ниже приведена HTML-разметка веб-страницы My Playlist, на которой отображается список песен для воспроизведения (плейлист), однако он пока пуст. Вам нужно завершить JavaScript-код, чтобы песни добавились в список. Заполните пробелы в JavaScript-коде, который необходим для выполнения данной задачи. Проверьте свои ответы, посмотрев решение этого задания в конце главы, прежде чем двинетесь дальше.

Это наш JavaScript. Данный код будет обеспечивать заполнение списка песнями, указанными внизу, в ``.

Заполните пробелы недостающим кодом, чтобы песни были добавлены в список.

Пустой список песен. Приведенный чуть выше код должен добавлять содержимое в каждый `` плейлиста.

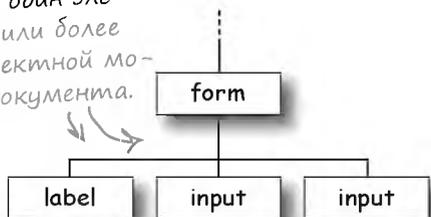
Если вы корректно заполните пробелы в JavaScript-коде, то веб-страница после загрузки будет выглядеть так.



Для чего еще хорошо подходит DOM

Объектная модель документа (DOM) позволяет делать намного больше, чем мы видели до сих пор, и вы будете использовать ее функциональность далее в процессе чтения книги, а пока давайте просто выполним краткий обзор, чтобы эти сведения отложились у вас в подсознании.

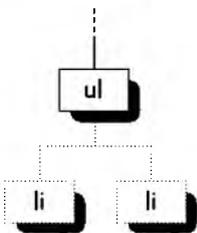
Ищите и извлекайте один элемент или более из объектной модели документа.



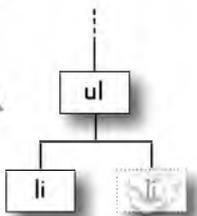
Создавайте новые элементы..



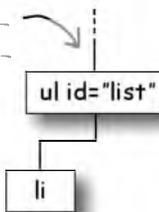
...и добавляйте их в DOM путем присоединения к другому элементу в дереве.



Удаляйте существующие элементы.



Получайте доступ и настраивайте атрибуты элементов, например `id` или `class`.



Извлекайте элементы из объектной модели документа.

Конечно, вы уже знаете, что это можно делать, поскольку мы с вами использовали `document.getElementById`. Однако существуют и другие способы извлечения элементов. Вы можете использовать имена тегов, имена классов и атрибуты для извлечения не только какого-то одного элемента, а целого набора элементов (например, всех элементов, имеющих в классе `"on_sale"`). Кроме того, вы можете извлекать значения формы, введенные пользователем (например, текст элемента ввода).

Создавайте и добавляйте элементы в объектную модель документа.

Вы можете создавать новые элементы и добавлять их в DOM. Естественно, любые изменения, вносимые вами в объектную модель документа, будут незамедлительно проявляться по мере того, как браузер будет осуществлять ее рендеринг (что просто замечательно!).

Удаляйте элементы из объектной модели документа.

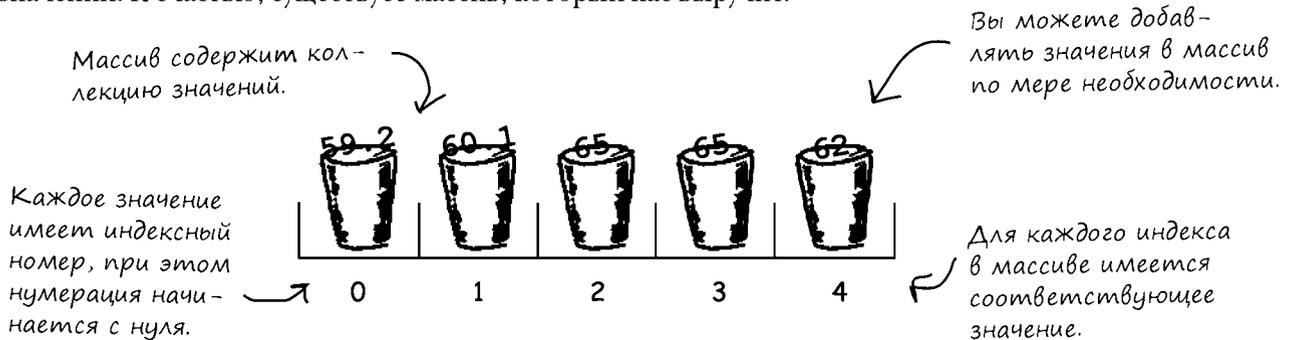
Вы также можете удалять элементы из DOM, для чего необходимо взять родительский элемент и удалить какой-либо из его дочерних элементов. Опять-таки, в окне браузера вы увидите, что элемент удален, как только он будет убран из объектной модели документа.

Извлекайте и настраивайте атрибуты элементов.

Ранее мы с вами получали доступ только к текстовому содержимому элементов, однако вы также можете получить доступ к их атрибутам. Например, вам может потребоваться узнать, каков класс конкретного элемента, а затем «на лету» изменить класс его принадлежности.

Нельзя ли снова поговорить о JavaScript, или как осуществляется сохранение множественных значений при использовании JavaScript

Мы уже достаточно поговорили о JavaScript и объектной модели документа (DOM). Прежде чем дать вам немного отдохнуть и расслабиться, мы хотим рассказать еще об одном JavaScript-типе, который вы будете постоянно использовать. Это массив (Array). Допустим, вам необходимо сохранить названия 32 сортов мороженого, или номера всех элементов в электропочке вашего пользователя, или, возможно, почасовые показатели уличной температуры. Чтобы сделать это с использованием простых переменных, потребуется масса времени, особенно если нужно сохранить десятки, сотни или тысячи значений. К счастью, существует массив, который нас выручит.



Как создать массив

Чтобы использовать массив, его сначала нужно создать, а также присвоить этот массив переменной, чтобы у нас было нечто такое, посредством чего мы будем ссылаться на него в своем коде. Давайте создадим массив вроде того, который показан выше и содержит почасовые показатели температуры (по Фаренгейту):

Наша переменная для массива...

...создание нового пустого массива

```
var tempByHour = new Array();
tempByHour[0] = 59.2;
tempByHour[1] = 60.1;
tempByHour[2] = 63;
tempByHour[3] = 65;
tempByHour[4] = 62;
```

Мы вернемся к этому синтаксису в главе 4, а пока просто знайте, что он создает новый массив.

Для добавления новых значений в массив мы просто ссылаемся на индексный номер элемента массива и присваиваем ему значение.

Как и в случае с переменными JavaScript, вы можете присвоить любое значение (или тип значения) индексу массива.

Индекс

JavaScript предусматривает также более быстрый способ создания и инициализации массива (мы называем его «литеральным массивом») с использованием значений:

```
var tempByHour = [59.2, 60.1, 63, 65, 62];
```

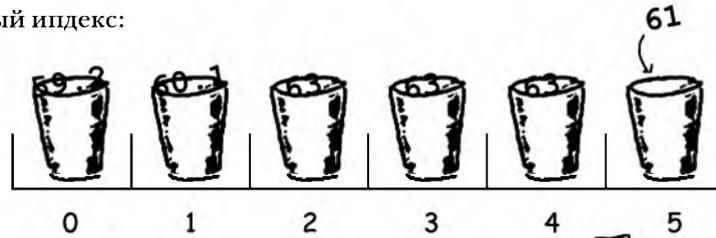
Данный код создает тот же массив, что и код, приведенный чуть выше, однако имеет меньший объем.

Добавление нового элемента в массив

Вы сможете в любой момент добавить новый элемент в свой массив, просто используя следующий незапятанный индекс:

```
tempByHour[5] = 61;
```

↑
Используя новый
порядковый индекс,
мы добавляем новый
элемент в массив.



Использование элементов своего массива

Вы можете извлечь значение элемента массива путем ссылки на перемешную массива с использованием индекса:

```
var message = "The temperature at 5 was " + tempByHour[5];  
alert(message);
```

↑
Для доступа к значению
температуры с индексом 5
мы просто ссылаемся
на массив с индексом 5.



Знайте размер своего массива, иначе...

Вы можете легко узнать размер своего массива путем ссылки на свойство массива, называемое length:

```
var numItems = tempByHour.length;
```

←
Подробнее о свойствах мы поговорим
в следующей главе, а пока просто
знайте, что каждый массив обладает
свойством length, которое позволяет
узнать количество элементов в нем.

Теперь, когда вы уже знаете, как выяснить длину массива, давайте посмотрим, можно ли объединить ваши знания о циклах с массивами...

Возьми в руку карандаш



Внизу вы найдете веб-страницу со списком пустых элементов, готовых к тому, чтобы ваш JavaScript заполнил их температурными показателями. Мы привели большую часть кода; вам необходимо устранить имеющиеся в нем пробелы, чтобы он смог задать содержимое для каждого элемента списка в виде соответствующего показателя температуры из массива (например, элемент списка с `id = "temp0"` получит температурный показатель с индексом 0 в массиве и т. д.). Таким образом, элемент списка с `id = "temp3"` будет гласить: "The temperature at 3 was 65". Попробуйте сделать так, чтобы элемент списка с `id = "temp0"` гласил "The temperature at noon was 59.2" вместо "The temperature at 0 was 59.2".

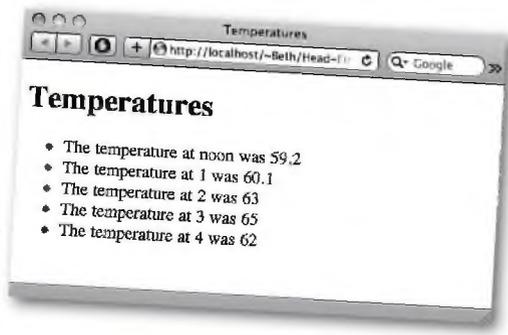
```

<!doctype html>
<html lang="en">
<head>
<title>Temperatures</title>
<meta charset="utf-8">
<script>
function showTemps() {
    var tempByHour = new _____;
    tempByHour[0] = 59.2;
    tempByHour[1] = 60.1;
    tempByHour[2] = 63;
    tempByHour[3] = 65;
    tempByHour[4] = 62;
    for (var i = 0; i < _____; ____ ) {
        var theTemp = _____[i];
        var id = "_____ " + i;
        var li = document._____ (id);
        if (i == ____ ) {
            li._____ = "The temperature at noon was " + theTemp;
        } else {
            li.innerHTML = "The temperature at " + _____ + " was " + _____;
        }
    }
}
window.onload = showTemps;
</script>
</head>
<body>
<h1>Temperatures</h1>
<ul>
    <li id="temp0"></li>
    <li id="temp1"></li>
    <li id="temp2"></li>
    <li id="temp3"></li>
    <li id="temp4"></li>
</ul>
</body>
</html>
    
```

← Это HTML

Здесь мы комбинируем циклы и массивы. Видите, как мы получаем доступ к каждому элементу массива с использованием индекса переменной?

Код, приведенный выше, будет заполнять каждый элемент списка содержимым в виде фразы, включающей словосочетание The temperature at...



пример приложения *phrase-o-matic*

Изучите данный код новенького приложения *Phrase-o-Matic* и посмотрите, сможете ли вы понять, что он делает, прежде чем двинетесь дальше...



```
<!doctype html>
<html lang="en">
<head>
  <title>Phrase-o-matic</title>
<meta charset="utf-8">
<style>
body {
  font-family: Verdana, Helvetica, sans-serif;
}
</style>
<script>
function makePhrases() {
  var words1 = ["24/7", "multi-Tier", "30,000 foot", "B-to-B", "win-win"];
  var words2 = ["empowered", "value-added", "oriented", "focused", "aligned"];
  var words3 = ["process", "solution", "tipping-point", "strategy", "vision"];

  var rand1 = Math.floor(Math.random() * words1.length);
  var rand2 = Math.floor(Math.random() * words2.length);
  var rand3 = Math.floor(Math.random() * words3.length);

  var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
  var phraseElement = document.getElementById("phrase");
  phraseElement.innerHTML = phrase;
}
window.onload = makePhrases;
</script>
</head>
<body>
  <h1>Phrase-o-Matic says:</h1>
  <p id="phrase"></p>
</body>
</html>
```

Испробуйте наше новое приложение *Phrase-o-Matic*, и вы превратитесь в блестящего оратора, как ваш босс или ребята из отдела маркетинга.



Вам показалось, что наше серьезное бизнес-приложение из главы 1 было недостаточно серьезным? Ладно. Тогда воспользуйтесь тем, которое приведено здесь, если вам нужно что-то показать боссу.

Phrase-O-Matic

Надеемся, вы догадались, что данный код представляет собой отличный инструмент для генерирования маркетинговых слоганов, отображаемых на веб-странице. В прошлом он уже генерировал такие удачные фразы, как *Win-win value-added solution* («Бесприигрышное эффективное решение») и *24/7 empowered process* («Процесс, идущий 24 часа в сутки, 7 дней в неделю»), и мы очень надеемся, что и будущие слоганы окажутся не хуже. Давайте посмотрим, как он работает.

- 1 Сначала мы определяем функцию `makePhrases`, которая будет выполняться после полной загрузки страницы, благодаря чему мы знаем, что сможем благополучно получить доступ к объектной модели документа (DOM):

Мы определяем функцию с именем `makePhrases`, которую будем вызывать позже.

```
function makePhrases() {
  }
window.onload = makePhrases;
```

Весь код для `makePhrases` будет размещаться здесь, и до него мы дойдем через несколько мгновений...

Мы будем выполнять `makePhrases`, как только закончится загрузка страницы.

- 2 Выполнив предыдущий шаг, мы можем приступить к написанию кода для функции `makePhrases`. Начнем с создания трех массивов. Каждый из них будет содержать слова, которые мы станем использовать для генерирования фраз. Мы применим быстрый способ создания этих массивов:

Мы создаем переменную с именем `words1`, которую сможем использовать для ссылки на первый массив.

```
var words1 = ["24/7", "multi-Tier", "30,000 foot", "B-to-B", "win-win"];
var words2 = ["empowered", "value-added", "oriented", "focused", "aligned"];
var words3 = ["process", "solution", "tipping-point", "strategy", "vision"];
```

Помещаем пять строк в массив. Но вы свободно можете заменить их какими-нибудь модными звучными словами.

Здесь вы можете видеть два дополнительных массива, присвоенных двум новым переменным с именами `words2` и `words3`.

- 3 Итак, у нас есть три новых массива, содержащих красивые звучные слова. Теперь мы будем осуществлять случайную выборку по слову из каждого массива, которые затем объединим в одну фразу.

Вот как производится выборка по одному слову из каждого массива:

Мы генерируем по одному случайному числу для каждого массива и присваиваем его новой переменной (rand1, rand2 и rand3 соответственно).

```
var rand1 = Math.floor(Math.random() * words1.length);  
var rand2 = Math.floor(Math.random() * words2.length);  
var rand3 = Math.floor(Math.random() * words3.length);
```

Данный код генерирует случайное число исходя из количества элементов в каждом массиве (в нашем случае их пять, однако вы свободно можете добавлять элементы в любой массив, и код по-прежнему сможет генерировать числа, беря за основу уже новое количество элементов).

- 4 Теперь сформируем отличный маркетинговый слоган, для чего возьмем все случайно выбранные слова и конкатенируем их друг с другом в одну фразу, разделив пробелами для удобочитаемости:

Мы определяем еще одну переменную для размещения фразы.

Каждое случайное число мы используем как индекс в массивах слов...

```
var phrase = words1[rand1] + " " + words2[rand2] + " " + words3[rand3];
```

- 5 Мы почти подошли к финалу: у нас есть фраза, которую теперь необходимо отобразить. Вы уже знаете, как мы будем действовать: применим getElementById для поиска нашего элемента <p>, а затем используем его свойство innerHTML для того, чтобы поместить в него новую фразу

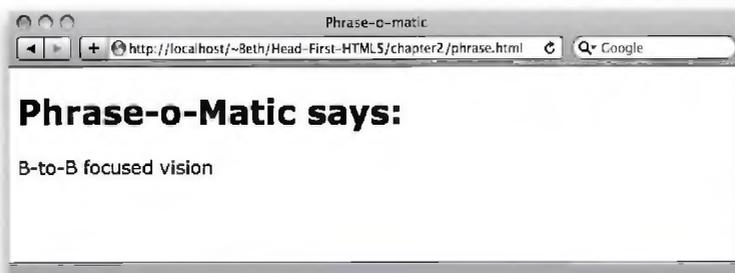
```
var phraseElement = document.getElementById("phrase");  
phraseElement.innerHTML = phrase;
```

Извлекаем элемент <p> с id в виде "phrase".

Затем задаем нашу фразу в качестве содержимого элемента <p>.

- 6 Итак, нанечатайте последнюю строку кода, еще раз окиньте все взглядом и ощутите удовлетворение от того, что вы довели дело до конца, прежде чем перейдете к загрузке страницы в своем браузере. Проведите ее тест-драйв и полюбуйтесь на генерируемые фразы.

А вот как выглядит наша фраза!



Простая перезагрузка страницы открывает перед вами возможности почти бесконечного генерирования все новых фраз. Незамысловатый код позволяет делать интересные вещи!

Часть Задаваемые Вопросы

В: Что именно представляет собой `Math` и что делают `Math.random` и `Math.floor`?

О: `Math` — это встроенная JavaScript-библиотека, содержащая набор математических функций. `Math.random` генерирует случайное число в промежутке между 0 и 1. Мы умножаем его на количество элементов в массиве (которое извлекаем посредством свойства `length` массива), чтобы получить число в промежутке между 0 и значением `length` массива. В результате, скорее всего, получится число с плавающей точкой (например, 3.2), поэтому мы применяем `Math.floor`, чтобы получить целое число, которое сможем использовать в качестве индекса в массиве для выборки случайного слова. Все, что делает `Math.floor`, — это отбрасывает цифры, идущие после знака десятичной дроби в числе с плавающей точкой. Например, `Math.floor(3.2)` возвращает результат 3.

В: Где можно найти документацию к `Math`?

О: В отличном справочнике по JavaScript под авторством Дэвида Флэнагана «JavaScript. Подробное руководство» (JavaScript: The Definitive Guide).

В: Ранее вы отмечали, что примитивы (значения `number`, `string` и `boolean`) можно сохранять в переменных или объектах. Но мы сохраняем массивы в переменных. Так чем является массив: примитивом или объектом?

О: Отличный вопрос! Массив — это особый род объекта, который встроен в JavaScript. Особый он потому, что его можно использовать для доступа к значениям, располагающимся в массиве, чего нельзя сделать с помощью остальных объектов (немассивов) или объектов, которые вы создаете сами. О создании своих собственных объектов мы поговорим в главе 4.

В: Что будет, если я попытаюсь получить доступ к индексу массива, который не существует? Например, если бы у меня было пять сохраненных слов

в `myWords`, а я при этом попытался бы получить доступ к `myWords[10]`.

О: Будет возвращено значение `undefined`, являющееся значением переменной, которой еще не было присвоено некое значение.

В: Можно ли удалить элемент из массива? Если да, то что в таком случае произойдет с индексами других элементов?

О: Удалить элемент из массива можно двумя разными способами. Вы можете задать значение для массива с соответствующим индексом в виде `null` (например, `myArray[2] = null`). Однако это будет означать, что длина массива останется прежней. Либо вы можете удалить требуемый элемент полностью (с помощью функции `splice`). В данном случае индексы элементов, идущих после удаленного вами элемента, сдвинутся вниз на единицу. Таким образом, если `myArray[2] = "dog"` и `myArray[3] = "cat"`, а вы удалите "dog", то `myArray[2] = "cat"`, а длина вашего массива станет короче на единицу.

Изучение языка является непростой задачей, и важно, чтобы ваш мозг не только трудился, но и отдыхал. Поэтому, закончив читать данную главу, устройте себе передышку, перекусите немного и, прежде чем перейти к следующей главе, ознакомьтесь с рубрикой «Ключевые моменты» и решите кроссворд, чтобы закрепить изученный материал.



Мы еще не научились преобразовывать цифровые изображения с едой в нечто вещественное, так что вам самим придется позаботиться о закусках.

КЛЮЧЕВЫЕ МОМЕНТЫ

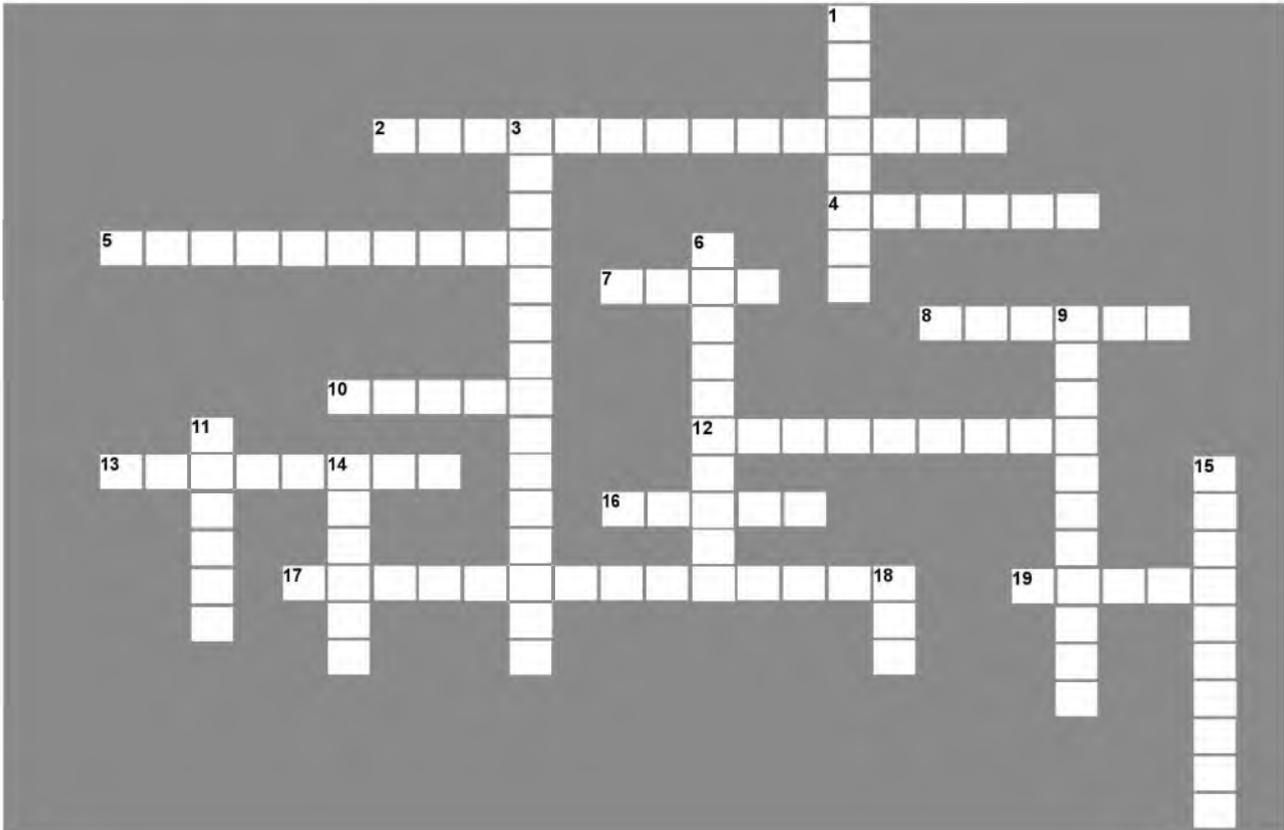


- Объявлять JavaScript-переменную необходимо с использованием `var`.
- `boolean`, `number` и `string` — это примитивные типы.
- Логическими значениями являются `true` и `false`.
- Числовыми значениями могут быть целочисленные величины или числа с плавающей точкой.
- Неинициализированная переменная имеет значение `undefined`.
- `undefined` и `null` — это два разных значения. Первое означает, что переменной еще не было присвоено значение, а `null` — что переменная имеет значение, под которым подразумевается «значения нет».
- Результатом оценки числовых, логических и строковых выражений будут соответственно числовые, логические и строковые значения.
- Для повторяющегося выполнения блоков кода следует использовать цикл `while`.
- Циклы `for` и `while` позволяют выполнять одни и те же действия; используйте тот из них, который наилучшим образом подходит в вашей ситуации.
- Чтобы выполнение цикла `for` или `while` завершилось, проверка условия в некий момент должна выдать `false`.
- Операторы `if/else` могут использоваться для принятия решений в зависимости от результатов проверок условий.
- Проверки условий являются логическими выражениями.
- Вы можете добавлять JavaScript-код в `<head>` или `<body>` своей веб-страницы либо размещать его в отдельном файле, внедрив ссылку на него в страницу.
- JavaScript-код (или ссылку на него) необходимо заключать в элемент `<script>`.
- Когда браузер загружает веб-страницу, он создает объектную модель документа (DOM), которая является внутренним представлением этой веб-страницы.
- Сделать свои веб-страницы интерактивными можно путем исследования и изменения DOM с использованием JavaScript.
- Для получения доступа к требуемому элементу на своей веб-странице необходимо воспользоваться `document.getElementById`.
- `document.getElementById` использует значение `id` элемента для поиска этого элемента в объектной модели документа.
- Для изменения содержимого элемента следует использовать свойство `innerHTML` этого элемента.
- Если вы попытаетесь получить доступ или изменить элементы до того, как веб-страница полностью загрузится, появится сообщение об ошибке JavaScript и ваш код не сработает.
- Присвойте функцию свойству `window.onload`, чтобы выполнение кода, имеющегося в этой функции, осуществлялось после того, как браузер завершит загрузку веб-страницы.
- Используйте массив для сохранения более одного значения.
- Для доступа к значению в массиве необходимо использовать индекс. Индекс — это целое число, означающее позицию элемента в массиве (нумерация при этом начинается с нуля).
- Свойство `length` массива позволяет узнать количество элементов в массиве.
- Комбинируя циклы и массивы, вы сможете последовательно получать доступ к каждому элементу массива.
- `Math` — это JavaScript-библиотека, включающая набор математических функций.
- `Math.random` возвращает число с плавающей точкой в промежутке между 0 и 1 (но никогда ровно 1).
- `Math.floor` преобразует число с плавающей точкой в целое число, отбрасывая все цифры, идущие после знака десятичной дроби.



HTML5-крсворд

Настало время заставить поработать левое полушарие вашего мозга нутем решения крсворда. Удачи!



По горизонтали

2. Если написать `3 + "Stooges"`, то JavaScript осуществит _____ `3` в строку.
4. _____ используется для извлечения значения из массива.
5. `5 < 10` — это _____ выражение.
7. Вы можете добавлять свой JavaScript-код в _____ или `body` HTML-документа.
8. Количество элементов в массиве можно узнать с помощью свойства _____.
10. Имена переменных могут начинаться с _____, знака `$` или символа подчеркивания.
12. Выбирайте подходящие имена для переменных и используйте стиль _____ для указания имен, состоящих из нескольких слов.
13. _____ — это корень дерева DOM.
16. Повторяйте выполнение одних и тех же блоков кода посредством цикла _____.
17. В JavaScript `document._____` позволяет извлечь требуемый элемент из объектной модели документа (DOM).
19. Сохраняйте названия сортов мороженого все вместе в одном _____.

По вертикали

1. Объектная модель документа (DOM) — это внутреннее представление веб-_____.
3. Браузер создает _____ документа при загрузке страницы.
6. Добавьте его в свои веб-страницы, чтобы сделать их интерактивными.
9. Значение `id` элемента `<p>`, содержащего текст "Red Alert: hit by phaser fire!".
11. Заключайте свой JavaScript-код в тег `<_____>`, если помещаете его в HTML-страницу.
14. Если вы почти закончили, то выпейте чаю, а если, как в условии _____, конец работы еще не близок, продолжайте дальше!
15. Циклы `while` и `for` используют _____ выражение в качестве проверки условия.
18. С _____ нельзя начинать взаимодействовать, пока страница не загрузилась полностью.



Задание
Решение

Выразите себя!

Ранее вы познакомились с различными типами выражений, которые можно использовать в JavaScript-коде. Теперь пора применить эти знания на практике и самим оценить несколько выражений. Приведем решение этого задания.

`(9 / 5) * tempC + 32`

Каким окажется результат, если значением `tempC` будет 10? 50

`"Number" + " " + "2"`

Какова будет результирующая строка? Number 2

`level >= 5`

Каким окажется результат, если значением `level` будет 10? true

А каким окажется результат, если значением `level` будет 5? true

↪ `>=` означает «больше либо равно»

`color != "pink"`

Каким окажется результат, если значением `color` будет `blue`? true

↪ `color` «не равен» `pink`

`(2 * Math.PI) * r`

Каким окажется результат, если значением `r` будет 3? 18.84

↪ `Math.PI` возвращает значение `pi` (оно, как вы знаете, равно 3,14...)

↪ Приблизительно!



↪ Так выражаться не стоит!

Возьми в руку карандаш Решение

Взяв за основу свои текущие знания о переменных, выражениях и операторах JavaScript, посмотрите, сможете ли вы сказать, какие из этих операторов являются допустимыми, а какие приведут к выводу ошибки.

В приведенном ниже перечне обведите допустимые операторы.

`var x = 1138;`

`var y = 3/8;`

`var s = "3-8";`

`x = y;`

`var n = 3 - "one";`

↪ Технически данный оператор является допустимым, однако получаемое в результате значение нельзя использовать.

`var t = "one" + "two";`

`var 3po = true;` недопустимый!

`var level_ = 11;`

`var highNoon = false;`

`var $ = 21.30;`

`var z = 2000;`

`var isBig = y > z;`

`z = z + 1;`

`z--;`

`z y;` недопустимый!

`x = z * t;`

```
while (highNoon) {
    z--;
}
```

СТАНЬ браузером. Решение



Каждый из приведенных на этой странице JavaScript-фрагментов представляет собой отдельный блок кода. Ваша задача заключается в том, чтобы сыграть роль браузера и оценить все фрагменты кода для ответа на вопросы о результатах. Напишите свой ответ на каждый вопрос под соответствующим фрагментом

Фрагмент 2

```
var tops = 5;
while (tops > 0) {
  for (var spins = 0; spins < 3; spins++) {
    alert("Top is spinning!");
  }
  tops = tops - 1;
}
```

15

Внешний цикл while выполняется пять раз, а внутренний цикл for — три раза при каждом выполнении внешнего цикла, поэтому получается $5 * 3$, или 15!

Сколько раз на экране появится диалоговое окно alert с сообщением "Top is spinning!"?

Здесь мы начинаем с 5 и выполняем цикл до тех пор, пока количество ягод не станет равным 0, ведь отсчет в убывающем порядке при каждом заходе (а не в возрастающем).

Фрагмент 4

```
for (scoops = 0; scoops < 10; scoop++) {
  alert("There's more ice cream!");
}
alert("life without ice cream isn't the same");
```

10

Сколько ложек мороженого вы съели?

Фрагмент 1

```
var count = 0;
for (var i = 0; i < 5; i++) {
  count = count + i;
}
alert("count is " + count);
```

Какой показатель общего количества будет отображен в диалоговом окне alert?

10

При каждом выполнении цикла мы добавляем значение i к показателю общего количества, а значение i все время увеличивается, поэтому при каждом выполнении цикла мы добавляем к показателю общего количества не 1, а 0, 1, 2, 3 и 4.

Фрагмент 3

```
for (var berries = 5; berries > 0; berries--) {
  alert("Eating a berry");
}
```

Сколько ягод вы съели?

5

Здесь все просто: цикл выполняется 10 раз, поэтому вы съели 10 ложек!



Решение

Возьмите приведенный выше код и вставьте его в цикл while внизу. Пройдитесь по циклу while и напишите сообщения диалоговых окон alert в той последовательности, в которой они будут выводиться. Вот наше решение этого задания.

```
var scoops = 10;
```

```
while (scoops >= 0) {
```

```
  if (scoops == 3) {
```

```
    alert("Ice cream is running low!");
```

← Вставленный код
← Это сообщение выводится один раз, когда значение scoops равно 3.

```
  } else if (scoops > 9) {
```

```
    alert("Eat faster, the ice cream is going to melt!");
```

← Это тоже выводится один раз, когда значение scoops равно 10.

```
  } else if (scoops == 2) {
```

```
    alert("Going once!");
```

← Каждое из этих сообщений выводится один раз, когда значение scoops равно соответственно 2, 1 и 0.

```
  } else if (scoops == 1) {
```

```
    alert("Going twice!");
```

```
  } else if (scoops == 0) {
```

```
    alert("Gone!");
```

← А это выводится всякий раз, когда ни одно из прочих условий не имеет значения true, то есть когда значение scoops равно 9, 8, 7, 6, 5 и 4.

```
  } else {
```

```
    alert("Still lots of ice cream left, come and get it.");
```

```
  }
```

```
  scoops = scoops - 1;
```

← Мы вычитаем по одной ложке при каждом выполнении цикла.

```
}
```

```
alert("Life without ice cream isn't the same.");
```

← Это сообщение выводится, когда выполнение цикла завершено.

Сообщения, выводимые в диалоговых окнах alert:

```

    Eat faster, the ice cream is going to melt!
    Still lots of ice cream left, come and get it.
    Still lots of ice cream left, come and get it.
    Still lots of ice cream left, come and get it.
    Still lots of ice cream left, come and get it.
    Still lots of ice cream left, come and get it.
    Still lots of ice cream left, come and get it.
    Still lots of ice cream left, come and get it.
    Ice cream is running low!
    Going once!
    Going twice!
    Gone!
    Life without ice cream isn't the same.
  
```



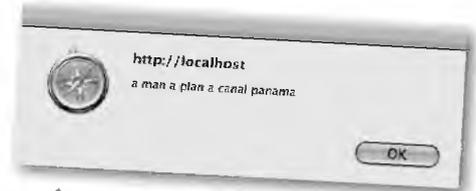
Развлечения с магнитами. Решение

Данный код отображает известный палиндром в диалоговом окне alert. Проблема заключается в том, что часть кода находилась на магнитных табличках, прикрепленных к холодильнику, однако они упали на пол. Ваша задача заключается в том, чтобы восстановить целостность кода и отобразить палиндром. Будьте внимательны, поскольку на полу уже лежало несколько табличек, не имеющих отношения к данному коду, зато некоторые из табличек вам придется использовать более одного раза! Приведем решение этого задания.

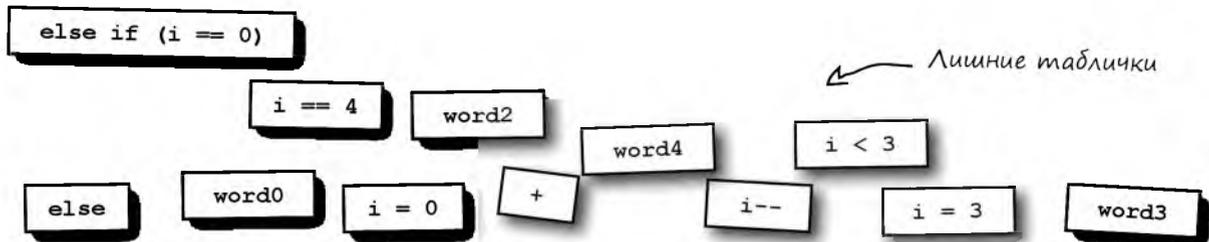
```
var word1 = "a";
var word2 = "nam";
var word3 = "nal p";
var word4 = "lan a c";
var word5 = "a man a p";
```

```
var phrase = "";
```

```
for (var i = 0; i < 4; i++) {
  if (i == 0) {
    phrase = word5;
  }
  else if (i == 1) {
    phrase = phrase + word4;
  }
  else if (i == 2) {
    phrase = phrase + word1 + word3;
  }
  else if (i == 3) {
    phrase = phrase + word1 + word2 + word1;
  }
}
alert(phrase);
```



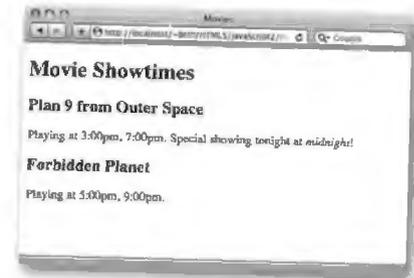
Палиндром — это предложение, которое одинаково читается как слева направо, так и справа налево! Вот палиндром, который вы должны увидеть, если правильно разместите все таблички на магнитах по местам.





СТАНЬ браузером. Решение

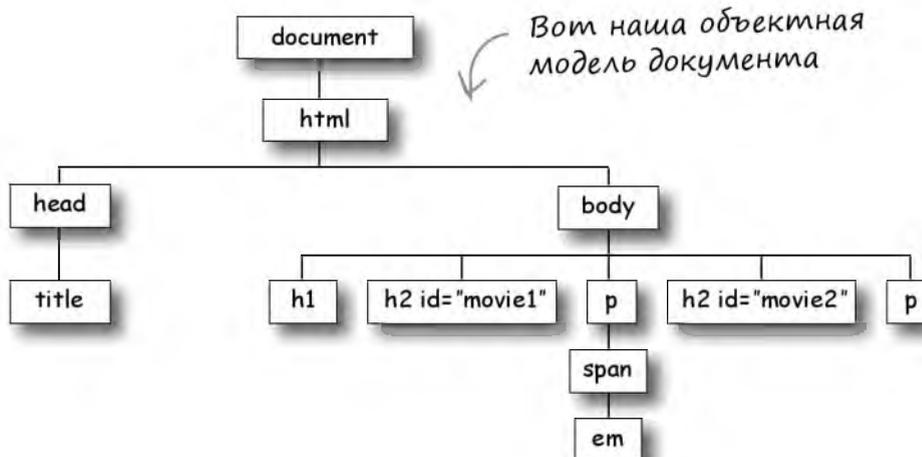
Ваша задача заключается в том, чтобы сыграть роль браузера. Вам необходимо осуществить разбор HTML-разметки и создать на ее основе свою собственную объектную модель документа (DOM). Поэтому сначала проведите разбор HTML, который находится справа, а DOM нарисуйте внизу. Начало объектной модели документа мы уже нарисовали, а Вам предстоит ее завершить.



```

<!doctype html>
<html lang="en">
  <head>
    <title>Movies</title>
  </head>
  <body>
    <h1>Movie Showtimes</h1>
    <h2 id="movie1" >Plan 9 from Outer Space</h2>
    <p>Playing at 3:00pm, 7:00pm.
      <span>
        Special showing tonight at <em>midnight</em>!
      </span>
    </p>
    <h2 id="movie2">Forbidden Planet</h2>
    <p>Playing at 5:00pm, 9:00pm.</p>
  </body>
</html>

```



Возьми в руку карандаш

Решение

Ниже приведена HTML-разметка веб-страницы My Playlist, на которой отображается список песен для воспроизведения (плейлист), однако пока он пуст. Вам нужно завершить JavaScript-код, чтобы песни добавились в список. Вот наше решение этого задания.

```

<!doctype html>
<html lang="en">
<head>
  <title>My Playlist</title>
  <meta charset="utf-8">
  <script>
    function addSongs () {
      var song1 = document.getElementById (" song1 ");
      var song2 = document.getElementById (" song2 ");
      var song3 = document .getElementById (" song3 ");

      song1 .innerHTML = "Blue Suede Strings, by Elvis Pagely";
      song2.innerHTML = "Great Objects on Fire, by Jerry JSON Lewis";
      song3. innerHTML = "I Code the Line, by Johnny JavaScript";
    }
    window. onload = addSongs ;
  </script>
</head>
<body>
  <h1>My awesome playlist</h1>
  <ul id="playlist">
    <li id="song1"></li>
    <li id="song2"></li>
    <li id="song3"></li>
  </ul>
</body>
</html>

```



Если вы корректно заполните пробелы в JavaScript-коде, то веб-страница после загрузки будет выглядеть так.

Код, благодаря которому будет заполнен наш плейлист.

Вы можете свободно подставлять сюда названия своих любимых песен!

Приведенный чуть выше код задает содержимое для этих элементов `` путем извлечения каждого элемента из DOM и присваивания свойству `innerHTML` значения в виде названия соответствующей песни.

Возьми в руку карандаш



Решение

Внизу вы найдете веб-страницу со списком пустых элементов, готовых к тому, чтобы ваш JavaScript заполнил их температурными показателями. Мы привели большую часть кода; вам необходимо устранить имеющиеся в нем пробелы, чтобы он смог задать содержимое для каждого элемента списка в виде соответствующего показателя температуры из массива. Рассмотрим решение этого задания.

```

<!doctype html>
<html lang="en">
<head>
<title>Temperatures</title>
<meta charset="utf-8">
<script>
function showTemps() {
    var tempByHour = new Array();
    tempByHour[0] = 59.2;
    tempByHour[1] = 60.1;
    tempByHour[2] = 63;
    tempByHour[3] = 65;
    tempByHour[4] = 62;
    for (var i = 0; i < tempByHour.length; i++) {
        var theTemp = tempByHour[i];
        var id = "temp" + i;
        var li = document.getElementById(id);
        if (i == 0) {
            li.innerHTML = "The temperature at noon was " + theTemp;
        } else {
            li.innerHTML = "The temperature at " + i + " was " + theTemp;
        }
    }
}
window.onload = showTemps;
</script>
</head>
<body>
<h1>Temperatures</h1>
<ul>
    <li id="temp0"></li>
    <li id="temp1"></li>
    <li id="temp2"></li>
    <li id="temp3"></li>
    <li id="temp4"></li>
</ul>
</body>
</html>

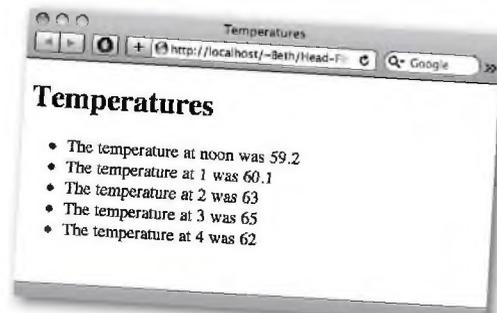
```

Создаем новый массив для размещения температурных показателей.

Комбинируем циклы и массивы. Обратите внимание на то, что мы используем значение *i* в качестве индекса в массиве, поэтому получаем доступ к каждому элементу по мере того, как значение *i* увеличивается при каждом выполнении цикла.

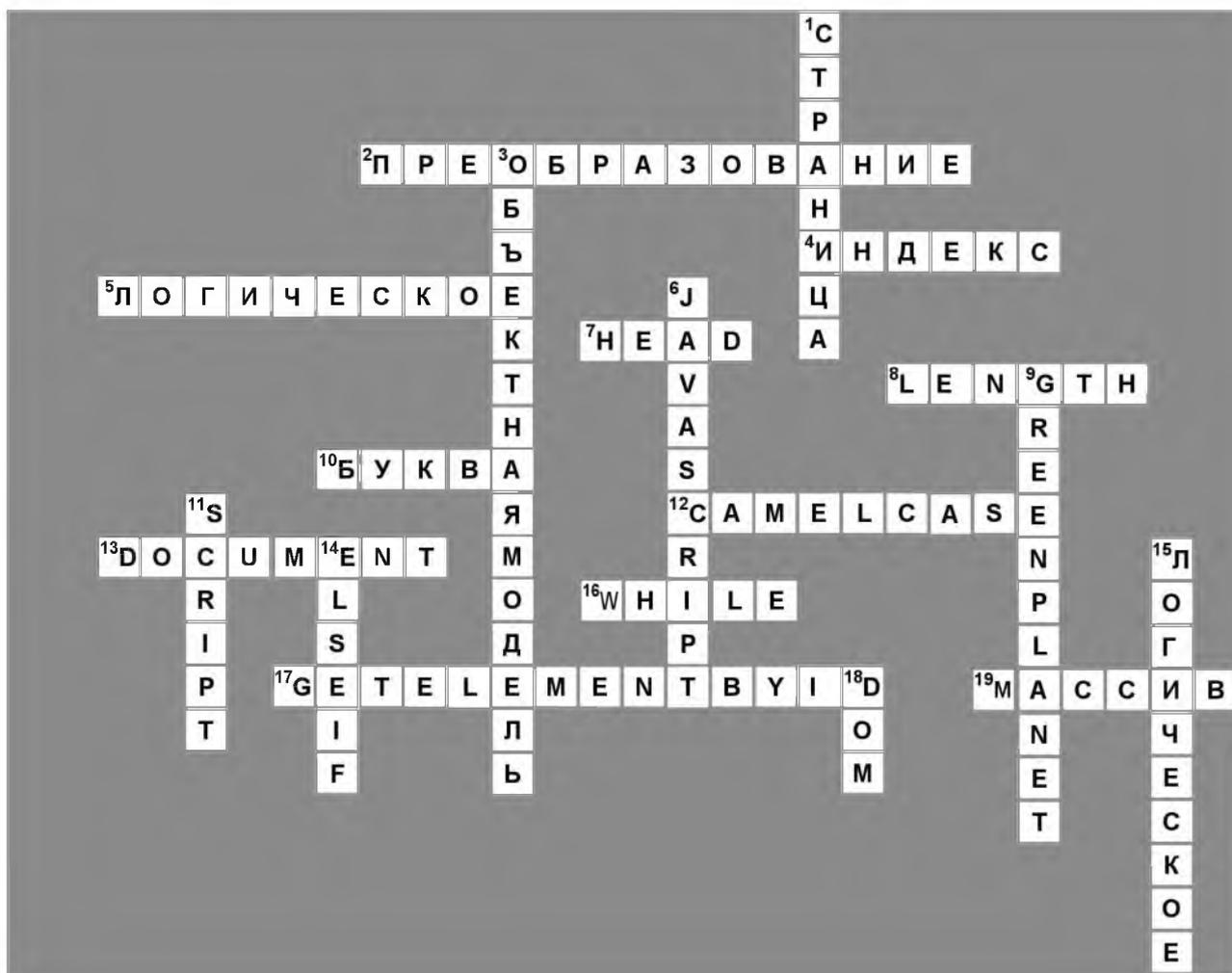
Генерируем строку, которая будет использоваться, для чего задействуем переменные *i* и *theTemp*.

А вот наши результаты!





HTML5-кроссворд. Решение



3 события, обработчики и Весь этот Джаз

Немного взаимодействия

Он, несомненно, выглядит прекрасно, но наши взаимоотношения были бы более увлекательными, если бы он действительно делал что-то время от времени.

Человек или манекен?
Решать вам.



Вам все еще не удастся соприкоснуться с пользователем. Вы изучили основы JavaScript, однако могут ли ваши веб-страницы взаимодействовать с пользователями? Когда страницы откликаются на вводимые пользователем данные, они уже являются не простыми документами, а живыми, реагирующими приложениями. В этой главе вы узнаете, как обрабатывать одну из форм ввода данных пользователем (извините за каламбур) и привязывать старомодный HTML-элемент `<form>` к современному коду. Это может показаться необычным, однако такой подход также эффективен. Пристегните ремни, поскольку наше путешествие по этой главе будет проходить на большой скорости: путь от простого до интерактивного приложения мы пройдем очень быстро.

Приготовьтесь к встрече с Webville Tunes

Итак, ранее в процессе чтения книги вы узнали массу всего об основах JavaScript, и, несмотря на то что мы с вами много говорили о создании веб-приложений, реальные примеры их создания мы еще кристально не рассматривали. Поэтому давайте теперь проявим серьезность (на самом деле, на этот раз без шуток!) и создадим реальное веб-приложение.

Пусть это будет менеджер плейлистов. Мы дадим ему какое-нибудь оригинальное название, например... скажем, Webville Tunes.



Добавляйте новые песни в любой момент.



Все ваши любимые песни будут отображаться прямо в окне браузера.

Вот что мы будем создавать.

Приложение будет полностью браузерным. Код на стороне сервера не потребуются.



Если вам известно, для чего нужен этот код:

```
window.onload = init;
```

то что, как вам кажется, делает вот этот код?

```
button.onclick = handleButtonClick;
```

Приступаем...

Для начала нет необходимости создавать большую, комплексную веб-страницу. На самом деле мы можем начать очень просто. Давайте создадим HTML5-документ с формой и элементом списка, где будет содержаться плейлист:

```

<!doctype html>
<html lang="en">
<head>
  <title>Webville Tunes</title>
  <meta charset="utf-8">
  <script src="playlist.js"></script>
  <link rel="stylesheet" href="playlist.css">
</head>
<body>
  <form>
    <input type="text" id="songTextInput" size="40" placeholder="Song name">
    <input type="button" id="addButton" value="Add Song">
  </form>
  <ul id="playlist">
  </ul>
</body>
</html>

```

Стандартные HTML5-элементы head и body.

Весь JavaScript-код мы поместим в файл playlist.js.

Мы включили таблицу стилей, чтобы придать красивый внешний вид нашему приложению.*

Все, что нам нужно, это простая форма. Вот она, с текстовым полем для ввода названий песен. Мы используем HTML5-атрибут placeholder, который демонстрирует пример того, что можно печатать в поле ввода.

А вот button с id в виде "addButton" для отправки новых дополнений в плейлист.

Мы будем использовать список для размещения песен. Он пока пуст, однако скоро мы это исправим с помощью JavaScript-кода...

Проведите тест-драйв



Напечатайте приведенный выше код, загрузите его в своем любимом браузере и полюбуйтесь на результат, нрежде чем перейдете к следующей странице.

Вот что вы должны увидеть.



* Не забывайте, что использованную в этом примере таблицу стилей (и весь код) вы можете загрузить на свой компьютер, посетив страницу <http://wickedlysmart.com/hfhtml5>.

Когда я нажимаю кнопку Add Song (Добавить песню), ничего не происходит

Что ж, и да, и нет. Вам кажется, что ничего не происходит, однако браузеру становится известно, что вы щелкнули на кнопке (кроме того, в зависимости от используемого браузера вы также увидите, как кнопка «отожмется» назад после ее нажатия).

На самом деле вопрос заключается в том, как заставить кнопку делать что-то, когда вы щелкаете на ней. Точнее, вопрос состоит в том, как будет происходить вызов необходимого JavaScript-кода, когда вы нажимаете кнопку?

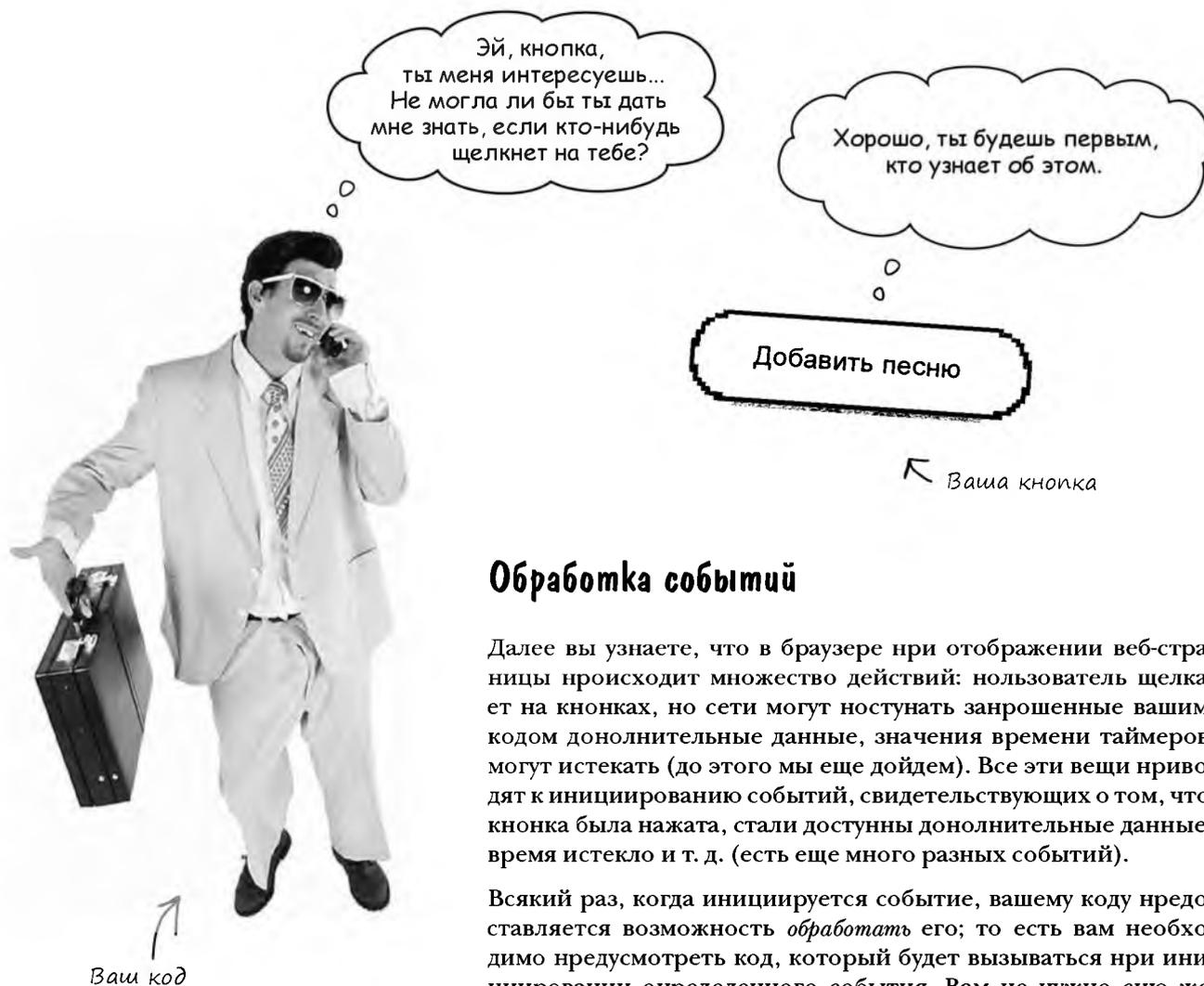
Здесь нам потребуются две вещи:

- 1 JavaScript-код, который будет оцениваться, когда пользователь щелкнет на кнопке Add Song (Добавить песню). Данный код (как только мы его напишем) обеспечит добавление новой песни в плейлист.
- 2 Способ «прицепить» данный код таким образом, чтобы при нажатии кнопки JavaScript знал, что необходимо добавить песню в плейлист.

Когда пользователь щелкает на кнопке (или, к примеру, нажимает ее пальцем на сенсорном экране какого-либо устройства), мы хотим узнать об этом.

Поэтому нас интересует событие, которое инициируется, когда «пользователь только что щелкнул на кнопке».





Обработка событий

Далее вы узнаете, что в браузере при отображении веб-страницы происходит множество действий: пользователь щелкает на кнопках, но сети могут поступать запрошенные вашим кодом дополнительные данные, значения времени таймеров могут истекать (до этого мы еще дойдем). Все эти вещи приводят к инициированию событий, свидетельствующих о том, что кнопка была нажата, стали доступны дополнительные данные, время истекло и т. д. (есть еще много разных событий).

Всякий раз, когда инициируется событие, вашему коду предоставляется возможность *обработать* его; то есть вам необходимо предусмотреть код, который будет вызываться при инициировании определенного события. Вам не нужно сразу же минуту обеспечивать обработку каких-либо событий, однако это потребует сделать, если вы хотите, чтобы при их инициировании что-то происходило: например, когда инициируется событие нажатия кнопки, вам может потребоваться, чтобы за этим последовало добавление новой песни в плейлист; когда поступают дополнительные данные, вы можете захотеть обработать их и отобразить на своей странице; когда запускается таймер, вам может потребоваться сообщить пользователю, что его бронь на билеты в первом ряду скоро истечет, и т. д.

Таким образом, мы знаем, что нам необходимо обеспечить обработку события, инициируемого при нажатии кнопки, поэтому рассмотрим, как это можно сделать.

Составляем план...

Давайте немного притормозим, нрежде чем углубимся в мир обработчиков и событий. Наша цель сейчас — сделать так, чтобы носле щелчка на кнонке Add Song (Добавить песню) произошло добавление песни в плейлист на странице. Подойдем к решению этой задачи на основе следующих этапов:

1. Задание обработчика для обработки события click в отношении кнопки Add Song (Добавить песню).
2. Написание обработчика для извлечения названия песни, введенного пользователем, после чего последует...
3. Создание нового элемента для размещения новой песни и...
4. Добавление нового элемента в DOM страницы.

Не беспокойтесь, если эти этапы не совсем ясны для вас, поскольку по ходу дела мы все вам объясним. Сейчас от вас требуется просто прочувствовать их, следуя за нами, пока мы будем заниматься написанием необходимого обработчика. Итак, откройте новый файл playlist.js, где будет располагаться весь ваш JavaScript-код.

Получение доступа к кнопке Add Song (Добавить песню)

Чтобы «нпросить» кнопку дать нам знать, когда будет инициировано событие, свидетельствующее о том, что пользователь щелкнул на ней, сначала необходимо получить доступ к этой кнопке. К счастью, мы создали данную кнопку с использованием HTML-разметки, и это означает, что... как вы уже догадались, она нредставлена в объектной модели документа (DOM), а вы уже знаете, как извлекать элементы оттуда. Если вы еще раз взглянете на HTML-разметку, то заметите, что мы нрисвоили id кнопки значение addButton. Таким образом, мы воспользуемся getElementById для извлечения ссылки на кнопку:

```
: var button = document.getElementById("addButton");
```

Тенерь нам нужно просто задать для кнопки код, который будет вызываться при событии click. Для этого мы создадим функцию с именем handleButtonClick, которая займется обработкой данного события. Подробнее о функциях мы поговорим немного позже, а пока необходимая нам функция будет выглядеть вот так:

Функция носит имя handleButtonClick; об особенностях синтаксиса мы поговорим чуть позже.

```
function handleButtonClick() {  
    alert("Button was clicked!");  
}
```

При вызове данной функции будет выводиться диалоговое окно alert.

Функция позволяет упаковать код в рамки отдельного блока. Вы можете присвоить ему имя и повторно использовать этот блок кода везде, где вам потребуется.

Весь код, который должен выполняться при вызове функции, мы заключаем в скобки.

1. Задание обработчика для обработки событий `click`
2. Написание обработчика для извлечения названия песни
3. Создание нового элемента для размещения новой песни
4. Добавление нового элемента в DOM страницы

Задание обработчика событий `click` для кнопки

Итак, теперь у нас есть кнопка, а также функция `handleButtonClick`, которая будет выступать обработчиком, так что давайте соединим их вместе. Для этого мы воспользуемся свойством `button` под именем `onclick`. Свойство `onclick` мы зададим следующим образом:

```
var button = document.getElementById("addButton");  
button.onclick = handleButtonClick;
```

Имея под рукой `button`, после вызова `getElementById` мы задаем для свойства `onclick` функцию, которая будет вызываться при наступлении события `click`.

Как вы помните, мы делали нечто подобное, когда использовали свойство `window.onload` для вызова функции после окончания загрузки окна. Однако в данном случае мы вызываем функцию после нажатия кнопки. Теперь соединим все:

```
window.onload = init;  
  
function init() {  
    var button = document.getElementById("addButton");  
    button.onclick = handleButtonClick;  
}  
  
function handleButtonClick() {  
    alert("Button was clicked!");  
}
```

Как и в предыдущей главе, здесь мы используем функцию `init`, которая не будет вызываться и выполняться до тех пор, пока страница полностью не загрузится.

После окончания загрузки страницы мы извлекаем `button` и задаем его обработчика `onclick`.

Обработчик событий `click` будет выводить диалоговое окно `alert` в случае щелчка на кнопке.

Проведение теста...



Нанечатайте приведенный выше код (в своем файле `playlist.js`), загрузите страницу, а затем нажмите на кнопку столько раз, сколько захотите. После каждого щелчка вы будете наблюдать появление диалогового окна `alert`.

Закончив тестирование своего нового обработчика событий `click` в отношении кнопки, откиньтесь на спинку стула и изучите код, продумав то, как он работает.

Когда вы поймете, что все это уже у вас в голове, переверните страницу, и мы с вами пройдемся по деталям, чтобы закрепить материал.



Более пристальный взгляд на происшедшее...

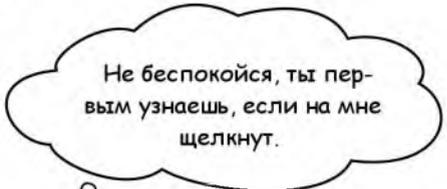
На нескольких последних страницах мы познакомил вас с массой новых аспектов, но поэтому давайте еще раз пройдемся по коду, чтобы убедиться в том, что вы все четко усвоили. Итак, приступим:

- 1 Первое, что мы сделали, — это добавили кнопку в свою HTML-форму. Затем нам потребовалось средство перехвата событий `click` в отношении этой кнопки, чтобы в результате был выполнен кое-какой код. Для этого мы создали обработчик и присвоили его свойству `onclick` нашего `button`.

```
function init() {  
    var button = document.getElementById("addButton");  
    button.onclick = handleButtonClick;  
}
```

Объект `button` обладает свойством `onclick`, для которого мы задаем функцию `handleButtonClick`.

Задаем обработчик событий `click` в функции `init` (то есть выполнение будет запускаться после завершения загрузки страницы).



Когда пользователь щелкает на кнопке, иницируется событие `click` и происходит вызов функции `handleButtonClick`.

```
function handleButtonClick() {  
    alert("Button was clicked!");  
}
```

- 2 Мы также написали простой обработчик, который выводит диалоговое окно `alert` с сообщением о том, что кнопка была нажата. Чуть позже мы напишем настоящий код для обработчика, а этот отлично подходит для тестирования.



Обработчик в вашем коде

3 Итак, код написан, страница загружается и отображается в окне браузера, обработчик задан. Теперь все зависит от пользователя...

4 Наконец, пользователь щелкает на нашей кнопке, в результате чего она активизируется, замечает, что у нее имеется обработчик, и вызывает его...

Ну, давай же...
Щелкай на кнопке...
Просто сделай это...



Очнись, пользователь щелкнул на тебе.

Вижу, у меня есть обработчик на этот случай, нужно дать ему знать.

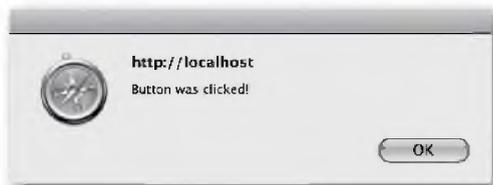


Да! Кто-то нажал кнопку. Я запускаю выполнение функции `handleButtonClick`.

```
function handleButtonClick() {
    alert("Button was clicked!");
}
```



Меня попросили сообщить вам, что кнопка была нажата... Я знаю, что для диалогового окна `alert` это не слишком впечатляюще, но, как бы там ни было, я просто делаю свою работу.



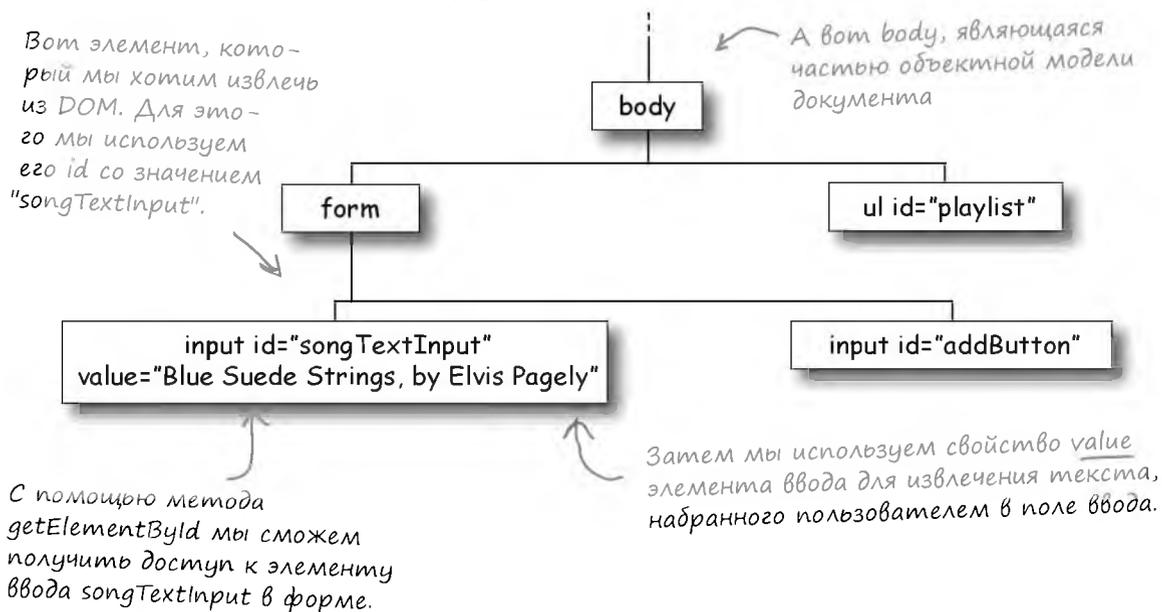
1. Задание обработчика для обработки событий click
2. Написание обработчика для извлечения названия песни
3. Создание нового элемента для размещения новой песни
4. Добавление нового элемента в DOM страницы

Извлечение названия песни

Мы готовы перейти ко второму этапу решения нашей задачи — извлечению названия песни, введенного пользователем. Сделав это, мы сможем задуматься над тем, как наш плейлист будет отображаться в браузере.

Однако как мы извлечем название песни? Это нечто такое, что ввел пользователь, ведь так? Ах да, все, что происходит на веб-странице, отражается и в объектной модели документа (DOM), поэтому текст, введенный пользователем, тоже должен быть там.

Для извлечения текста из текстового элемента ввода формы нам сначала потребуется извлечь этот элемент из DOM, и вы уже знаете, какой инструмент для этого нужен — `getElementById`. Сделав это, мы сможем воспользоваться свойством `value` текстового элемента ввода для получения доступа к тексту, введенному в поле формы пользователем, и вот как это все будет выглядеть:



Возьми в руку карандаш



Доработайте функцию `handleButtonClick`, приведенную ниже, чтобы извлечь название песни, набранное пользователем в элементе ввода формы. Проверить правильность своих ответов вы сможете, посмотрев решение данного задания на с. 130.

```
function handleButtonClick() {
    var textInput = document.getElementById(".....");
    var songName = .....value;
    alert("Adding " + .....);
}
```

Возьми в руку карандаш



БОНУС

А вдруг вам потребуется провести проверку, чтобы убедиться в том, что пользователь действительно ввел текст до того, как щелкнул на кнопке? Как это можно будет сделать? (Решение данного задания вы также сможете отыскать на с. 130.)

.....

.....

Часто
Задаваемые
Вопросы

В: Каким будет значение свойства `value` текстового элемента ввода, если пользователь ничего не ввел? Будет ли оно `null`? Или кнопка Add Song (Добавить песню) не станет вызывать обработчик, если пользователь ничего не ввел?

О: Кнопка Add Song (Добавить песню) не настолько умна. Если вы хотите определять, ввел ли что-нибудь пользователь, для этого вам потребуется соответствующий код. Чтобы узнать, является ли текстовое поле ввода пустым (то есть пользователь ничего в нем не напечатал), можете проверить, равно ли его значение строке, в которой ничего нет, также называемой *пустой строкой* и помечаемой как "" (пара двойных кавычек, между которыми ничего не стоит). Мы понимаем, почему вы решили, что значение может быть равно `null`, поскольку ранее отмечали, что это значение переменной, под которым подразумевается «значения нет», однако с точки зрения текстового поля ввода оно будет содержать не ничто, а строку, в которой ничего нет. Представьте себе! ;-).

В: Я думал, что `value` текстового элемента ввода — это атрибут. Но вы называете его свойством. Почему?

О: Вы правы, `value` — это *атрибут* текстового элемента ввода. Вы можете инициализировать значение текстового элемента ввода с использованием атрибута `value`. Однако в случае с JavaScript для доступа к значению, введенному пользователем, вам потребуется использовать *свойство* `value` элемента ввода, которое извлекается из объектной модели документа (DOM).

В: Какие еще типы событий можно обрабатывать на JavaScript помимо `click`?

О: Существует масса прочих событий, связанных с манипуляциями мышью, которые можно обрабатывать. Например, вы можете отслеживать и обрабатывать события, инициируемые при нажатии кнопки мыши, при наведении указателя мыши на элемент и убираании его прочь с элемента, при перетаскивании элемента с помощью мыши, при нажатии и удержании кнопки мыши (они отличаются от событий `click`). Кроме того, существует множество иных типов событий, о которых мы уже упоминали (например, события, инициируемые, когда становятся доступны дополнительные данные, события, связанные с таймерами и окном браузера, и т. д.). В процессе чтения книги вы увидите еще довольно много других типов событий, которые можно обрабатывать; если вы знаете, как обрабатывать события одного типа, то вы с большой долей вероятности сможете обработать события любых других типов!

В: Что делает JavaScript, пока ждет инициирования событий?

О: Если вы не запрограммировали свой JavaScript на выполнение каких-либо действий в это время, он будет бездельничать, пока что-нибудь не произойдет (пользователь начнет взаимодействовать с интерфейсом, поступят данные из Интернета, истечет значение времени таймера и т. д.). И это хорошо, поскольку вычислительная мощь вашего компьютера может быть направлена на другие вещи, например на то, чтобы сделать ваш браузер более отзывчивым. Позже мы расскажем, как создавать задачи, выполняемые в фоновом режиме, чтобы ваш браузер одновременно выполнял код задачи и реагировал на события.

Возьми в руку карандаш
Решение



Доработайте функцию `handleButtonClick`, приведенную ниже, чтобы извлечь название песни, набранное пользователем в элементе ввода формы. Вот наше решение этого задания.



Сначала нам необходимо извлечь ссылку на текстовый элемент ввода в форме. Мы присвоили `id` этого элемента значение `"songTextInput"`, поэтому можем использовать его в сочетании с `getElementById` для извлечения ссылки.

```
function handleButtonClick() {
    var textInput = document.getElementById("songTextInput");
    var songName = textInput.value;
    alert("Adding " + songName);
}
```

Будет выводиться диалоговое окно `alert`, в котором отобразится `"Adding"` и название песни.

Свойство `value` текстового элемента ввода содержит все, что набирается в текстовом поле и представляет собой строку. Здесь мы присваиваем введенный текст переменной `songName`.

Возьми в руку карандаш
Решение



БОНУС

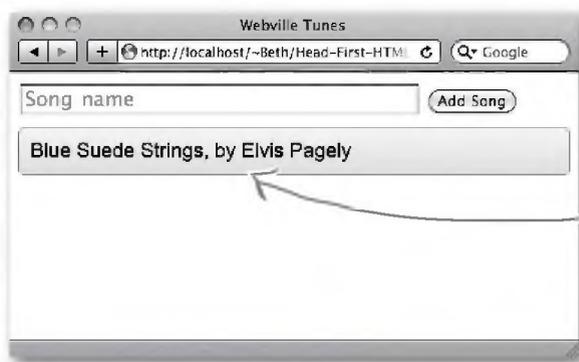
А вдруг вам потребуется провести проверку, чтобы убедиться в том, что пользователь действительно ввел текст до того, как щелкнул на кнопке? Как это можно будет сделать? Решение таково:

```
function handleButtonClick() {
    var textInput = document.getElementById("songTextInput");
    var songName = textInput.value;
    if (songName == "") {
        alert("Please enter a song");
    } else {
        alert("Adding " + songName);
    }
}
```

Мы можем использовать оператор `if` и сравнить строку `songName` с пустой строкой, чтобы убедиться, что пользователь действительно что-то напечатал. Если пользователь ничего не напечатал, то мы выведем диалоговое окно `alert` и попросим его ввести название песни.

Как добавить песню на страницу?

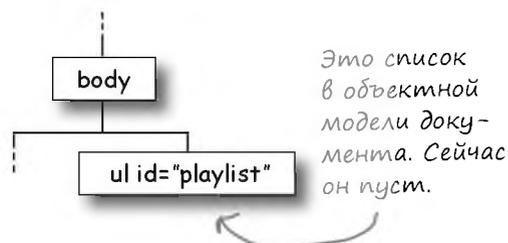
Мы с вами уже много чего сделали, и все работает! Вы можете ввести название песни в форму, щелкнуть на кнопке Add Song (Добавить песню) и извлечь то, что вводили в форму, — *все в рамках вашего кода*. Теперь мы отобразим плейлист на самой странице. Вот как это будет выглядеть.



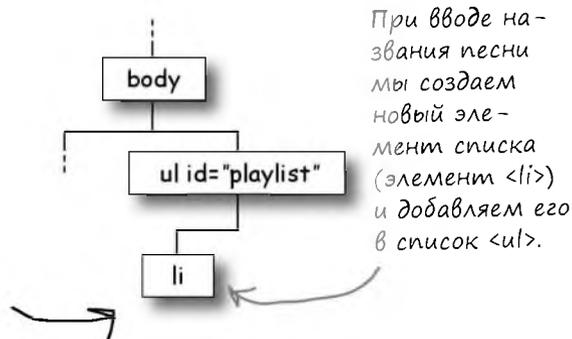
При нажатии кнопки Add Song (Добавить песню) ваш JavaScript добавляет соответствующую песню в список на странице.

Нам потребуется сделать следующее

- 1 Вы могли заметить, что мы уже добавили пустой список в HTML-разметку (пустой элемент ``), когда в первый раз вводили данные. В результате объектная модель документа (DOM) сейчас будет выглядеть так.



- 2 Каждый раз, когда вводится название новой песни, нам нужно добавлять новый элемент в неупорядоченный список. Для этого мы создадим новый элемент ``, в котором будет размещаться название новой песни. Затем мы возьмем новый элемент `` и добавим его в `` в объектной модели документа. В результате этого, когда браузер сделает свои дела, вы увидите, что страница подверглась обновлению, словно данный элемент `` был там всегда. И конечно же, все это мы сделаем в коде.

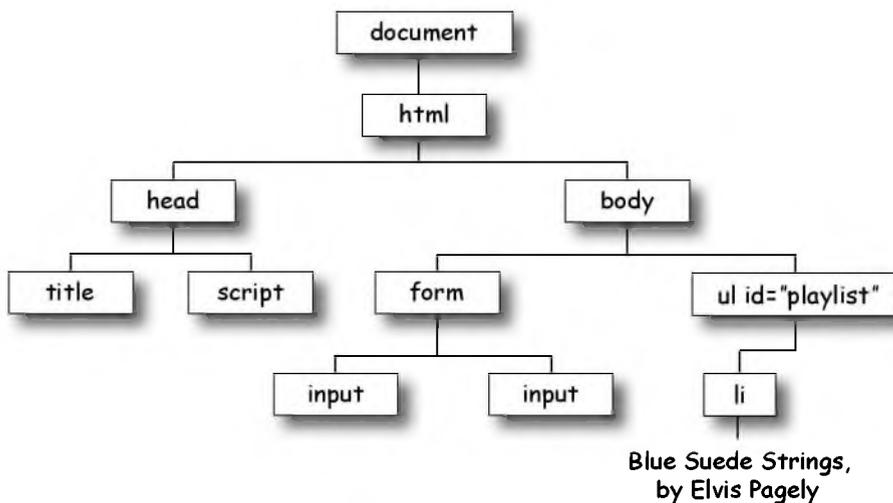


1. Задание обработчика для обработки событий `click`
2. Написание обработчика для извлечения названия песни
3. Создание нового элемента для размещения новой песни
4. Добавление нового элемента в DOM страницы



Упражнение

Здесь приведен плейлист, для которого вам необходимо нарисовать объектную модель документа в том виде, который он приобретет после добавления всех этих песен. Обратите внимание на порядок, в котором песни были добавлены на страницу, и разместите соответствующие элементы на своих местах в DOM. Один из них мы уже расположили в нужном месте, вам остается лишь сделать то же самое в отношении остальных элементов. Посмотрите решение данного задания в конце главы, прежде чем двинетесь дальше.



Blue Suede Strings,
by Elvis Pagely



Дорисуйте здесь остальную часть DOM для плейлиста, приведенного вверху.

Придется ли вам строить какие-либо предположения насчет порядка, в котором элементы `` добавляются в родительский элемент?

Как создать новый элемент

Вы уже видели, как можно получить доступ к *существующим* элементам посредством объектной модели документа. Однако DOM также можно использовать для создания новых элементов (с последующим *их* добавлением в DOM, о чем мы поговорим совсем скоро).

Давайте представим, что нам нужно создать элемент ``. Вот как мы это сделаем:

Для создания новых элементов используется `document.createElement`. Возвращается ссылка на новый элемент.

```
var li = document.createElement("li");
```

Здесь мы присваиваем новый элемент переменной `li`.

Передаем `createElement` в виде строки то, какой элемент хотим создать.



С помощью `createElement` создается совершенно новый элемент. Следует отметить, что он пока не вставляется в объектную модель документа. На данный момент он представляет собой лишь элемент, находящийся в свободном плавании, которому нужно пристанище в DOM.



Итак, теперь у нас имеется элемент ``, в котором ничего нет. Вы уже знаете способ добавить текстовое содержимое в элемент:

```
li.innerHTML = songName;
```

Наша переменная `li`.

Здесь мы задаем содержимое в виде названия песни для элемента ``.



Blue Suede Strings,
by Elvis Pagely

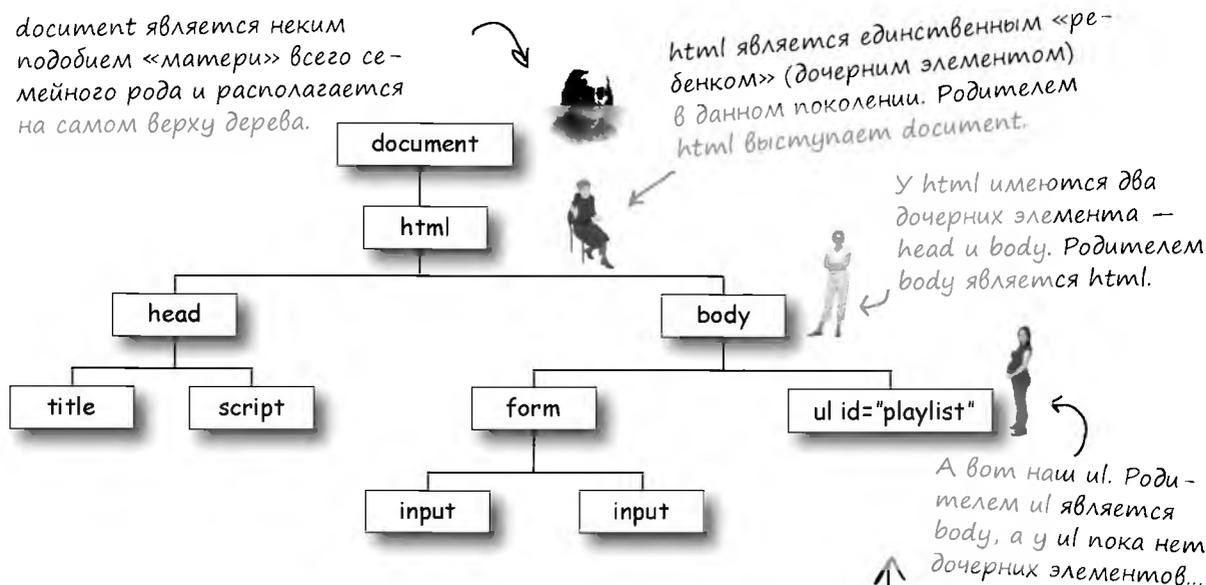
Наш новый объект элемента `li`, готовый ринуться в бой. Однако он еще не является частью DOM!

1. Задание обработчика для обработки событий click
2. Написание обработчика для извлечения названия песни
3. Создание нового элемента для размещения новой песни
4. Добавление нового элемента в DOM страницы

Добавление элемента в DOM

Чтобы добавить новый элемент в объектную модель документа, вам необходимо знать, куда его помещать. Что ж, нам это уже известно: мы собираемся поместить элемент `` в элемент ``. Но как это сделать? Давайте еще раз взглянем на DOM. Помните, как ранее мы отмечали, что она сродни дереву? Можете также считать ее чем-то вроде *родословного дерева*.

document является неким подобием «матери» всего семейного рода и располагается на самом верху дерева.



Таким образом, чтобы добавить наш элемент ``, необходимо сделать его дочерним по отношению к ``. Для этого нам сначала нужно отыскать элемент `` в дереве (мы присвоили его `id` значение "playlist", чтобы его проще было найти). Затем, чтобы произошло добавление элемента ``, мы предлагаем элементу `` добавить новый дочерний элемент. Все это будет выглядеть следующим образом:

Используем `getElementById` для извлечения ссылки на элемент `` с `id="playlist"`.

```
var ul = document.getElementById("playlist");
ul.appendChild(li);
```

Здесь мы предлагаем элементу `` добавить `` в качестве дочернего элемента. Как только это будет сделано, в DOM `` станет дочерним элементом ``, и браузер обновит страницу, чтобы в ней отразился новый ``.

...мы хотим сделать наш новый элемент `` дочерним по отношению к ``.

**Blue Suede Strings,
by Elvis Pagely**

При каждом вызове `appendChild` новый элемент `` будет добавляться в элемент `` после любых других элементов ``, которые уже там будут располагаться.

Соединяем все воедино...

Давайте соединим весь код и добавим его в функцию `handleButtonClick`. Наберите приведенный ниже код, если вы еще этого не сделали, чтобы затем можно было провести тестирование.

```
function handleButtonClick() {
    var textInput = document.getElementById("songTextInput");
    var songName = textInput.value;
    var li = document.createElement("li");
    li.innerHTML = songName;
    var ul = document.getElementById("playlist");
    ul.appendChild(li);
}
```

Сначала мы создаем новый элемент ``, в котором будет размещаться название новой песни.

Затем задаем содержимое в виде названия песни для данного элемента.

`` с `id "playlist"` является родительским элементом по отношению к нашему новому ``. Поэтому он следующий в очереди на извлечение.

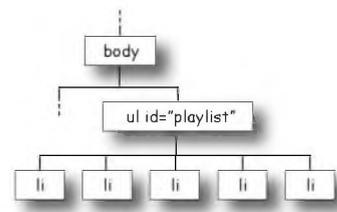
Добавляем объект `li` в `ul` с использованием `appendChild`.

Обратите внимание, что мы предлагаем родительскому элементу `ul` добавить в себя `li` в качестве дочернего элемента.

...и проводим тест-драйв



Протестируйте Webville Tunes, добавляя в плейлист песни. Вот наши результаты.



Так будет выглядеть объектная модель документа после добавления всех новых элементов ``.

Теперь, когда мы вводим название песни и нажимаем кнопку `Add Song` (Добавить песню), соответствующая песня добавляется в DOM, благодаря чему мы видим, что страница претерпевает изменения и в отображаемом на ней списке появляется новая песня.

Обзор того, что мы только что сделали

В этой главе мы с вами много чего сделали (нричем за короткий промежуток времени!). Используя JavaScript, мы создали менеджер нлейлистов, который нзволяет ввести название несни, щелкнуть на кнопке и добавить данную несню в список на странице

- 1 Первое, что мы сделали, — это задали обработчик для обработки событий click в отношении кнопки Add Song (Добавить песню). Мы создали функцию handleButtonClick и задали ее для свойства onclick кнопки Add Song (Добавить песню).

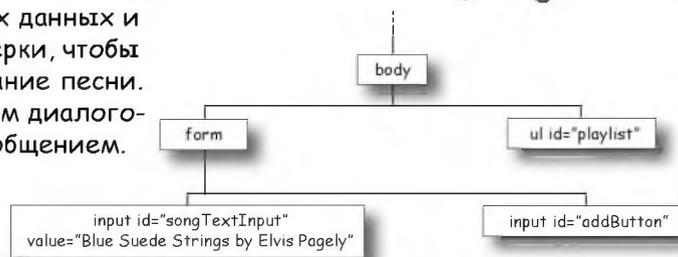
Когда пользователь щелкает на кнопке Add Song (Добавить песню), происходит вызов нашего обработчика handleButtonClick.

Add Song



- 2 Затем мы написали код для handleButtonClick, чтобы извлекать название песни из текстового поля ввода. Мы воспользовались свойством input.value для извлечения текстовых данных и даже позаботились о проведении проверки, чтобы убедиться, что пользователь ввел название песни. Если он этого не сделал, то мы выводим диалоговое окно alert с соответствующим сообщением.

В handleButtonClick мы извлекаем введенное пользователем название песни, используя свойство input.value для извлечения текстовых данных из объектной модели документа.



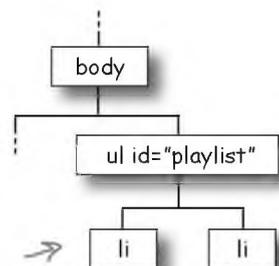
- 2 Чтобы добавить песню в плейлист, далее мы создали элемент , используя document.createElement, и задали содержимое в виде названия песни для этого элемента с помощью innerHTML.

Мы создали новый элемент и задали для него содержимое в виде названия песни.

li
Blue Suede Strings,
by Elvis Pagely

- 1 Наконец, мы внедрили новый элемент в объектную модель документа, добавив его в качестве дочернего элемента по отношению к родительскому . Мы сделали это с помощью appendChild, дав элементу команду «добавить в качестве дочернего элемента», в результате чего он был внедрен в объектную модель документа. При добавлении элемента в DOM браузер обновляет страницу, которую видит пользователь, и в плейлисте появляется название новой песни.

Добавление нового дочернего элемента в DOM приводит к обновлению страницы.



Постойте-ка, я понимаю, что мы взаимодействуем с DOM и все такое, но как данный код можно назвать реальным веб-приложением? Если я закрою окно браузера, то все мои песни пропадут. Разве не должны элементы моего плейлиста сохраняться, если это действительно приложение?

Мы с вами согласны, плейлист должен сохраняться; в конце концов, какой смысл добавлять все эти песни в плейлист, если они не будут сохранены? Кроме того, существует масса других функций, которые вы также можете захотеть внедрить. Например, вам может понадобиться добавить аудиоинтерфейс с использованием API-интерфейса `audio/video`, чтобы можно было прослушивать песни, делиться ими со своими друзьями посредством веб-служб (вроде Facebook и Twitter), искать в локальной сети людей, которым нравятся те же исполнители, что и вам (используя API-интерфейс Geolocation). К тому же мы уверены, что у вас могут возникнуть и другие пожелания в этом плане.

Возвращаясь к плейлисту... Мы хотели быстро продемонстрировать вам пример создания небольшого интерактивного приложения, и плейлист отлично подошел в данном случае. Кроме того, сохранение несен требует API-интерфейса Web Storage из версии HTML5, до рассмотрения которого — еще несколько глав.

С другой стороны, мы действительно не хотим, чтобы здесь оставалась какая-то недосказанность...

Переверните страницу





Готово
к употреблению



Мы приготовили для вас кое-какой код, так что вам не придется делать это самим.

Мы заранее приготовили для вас код, позволяющий сохранять плейлисты. Вам нужно просто напечатать его, а также внести нару небольших изменений в уже имеющийся код, чтобы в итоге получился сохраненный плейлист HTML5.

Обо всех особенностях сохранения данных локально в браузере мы поговорим в главе, посвященной API-интерфейсу Web Storage, а пока что вы научитесь сохранять плейлисты.

Естественно, никогда не лишним будет просмотреть приготовленный нами код. Вас может удивить, насколько многое вы уже знаете, не говоря уже о том, насколько многое вы сможете понять, даже если еще не знаете этого.



Будьте осторожны!

Приготовленный код не будет работать в Internet Explorer 6 и 7.

Данные версии браузера Internet Explorer не поддерживают localStorage. Поэтому если вы используете Internet Explorer, позаботьтесь о том, чтобы его версия была 8 или выше.



Будьте осторожны!

Приготовленный код не будет работать в некоторых браузерах, если вы загружаете свои страницы из file://, а не с сервера (например, localhost://) или онлайн-сервера.

В следующих главах мы поговорим об этой ситуации более подробно (она довольно часто возникает в случае с новыми функциями HTML5). А пока, если вы не хотите запускать выполнение сервера или копировать файлы на онлайн-сервер, используйте браузер Safari или Chrome.

Как добавить приготовленный код...

Чуть ниже приведен приготовленный код, который нужно добавить в приложение Webville Tunes, чтобы вы могли сохранить сформированный вами замечательный плейлист. Все, что вам нужно сделать, — это создать новый файл с именем `playlist_store.js`, перенести в него приведенный внизу код, а затем внести наряду изменений в уже имеющийся у вас код (на следующей странице).



Готово
К Употреблению

```
function save(item) {
    var playlistArray = getStoreArray("playlist");
    playlistArray.push(item);
    localStorage.setItem("playlist", JSON.stringify(playlistArray));
}

function loadPlaylist() {
    var playlistArray = getSavesSongs();
    var ul = document.getElementById("playlist");
    if (playlistArray != null) {
        for (var i = 0; i < playlistArray.length; i++) {
            var li = document.createElement("li");
            li.innerHTML = playlistArray[i];
            ul.appendChild(li);
        }
    }
}

function getSavesSongs() {
    return getStoreArray("playlist");
}

function getStoreArray(key) {
    var playlistArray = localStorage.getItem(key);
    if (playlistArray == null || playlistArray == "") {
        playlistArray = new Array();
    }
    else {
        playlistArray = JSON.parse(playlistArray);
    }
    return playlistArray;
}
```

Перенесите весь этот код
в файл `playlist_store.js`

Интегрирование подготовленного кода



Готово
к употреблению

Нам необходимо внести нару небольших изменений, чтобы интегрировать код для сохранения плейлиста. Сначала мы добавим ссылку на `playlist_store.js` в элемент `<head>` в `playlist.html`:

```
<script src="playlist_store.js"></script>
<script src="playlist.js"></script>
```

Добавьте эту строку, разместив ее прямо над ссылкой на `playlist.js`. Она загружает подготовленный код.

Теперь нужно добавить две строки в уже имеющийся у вас код в `playlist.js`, которые будут загружать и сохранять плейлист:

```
function init() {
    var button = document.getElementById("addButton");
    button.onclick = handleButtonClick;
    loadPlaylist();
}
```

← Данная строка загружает сохраненные песни из `localStorage`, когда вы загружаете свою страницу, благодаря чему они отображаются на экране в плейлисте.

```
function handleButtonClick() {
    var textInput = document.getElementById("songTextInput");
    var songName = textInput.value;
    var li = document.createElement("li");
    li.innerHTML = songName;
    var ul = document.getElementById("list");
    ul.appendChild(li);
    save(songName);
}
```

← А эта строка сохраняет каждую новую песню, которую вы добавляете в плейлист.

Мы добавили все эти песни, закрыли окно браузера, затем снова открыли его, загрузили страницу, и перед нами предстал сохраненный плейлист с песнями.

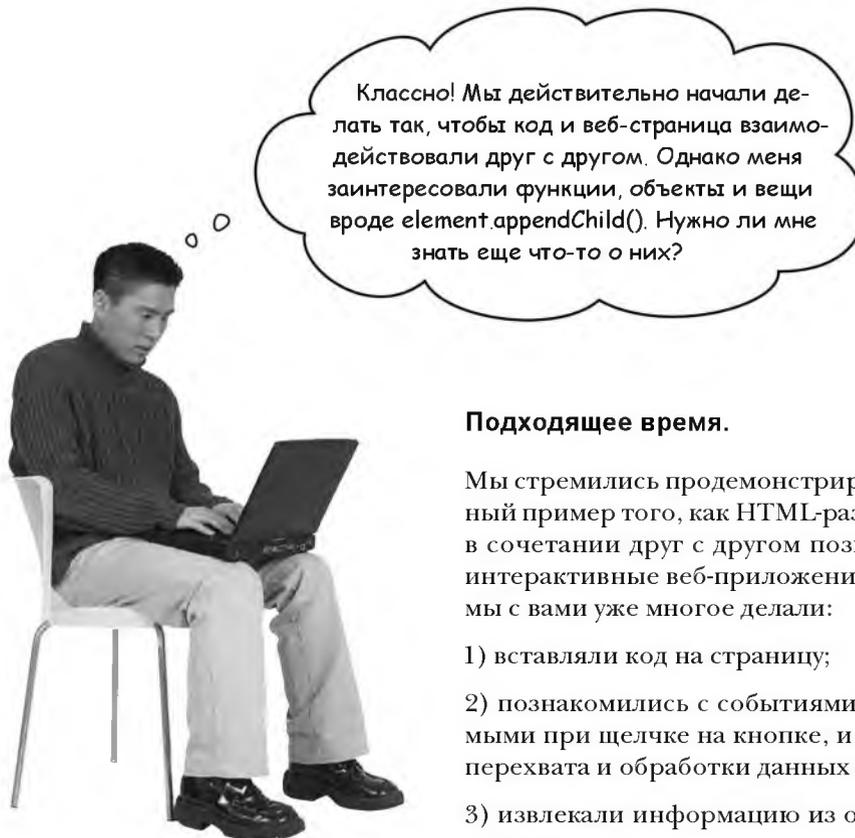
Тест-драйв сохраненного плейлиста



Итак, перезагрузите страницу и введите названия нескольких песен. Закройте окно браузера. Снова откройте его и опять загрузите страницу. После этого вы должны увидеть плейлист со всеми своими сохраненными песнями.

← Вам надоел ваш плейлист, и вы решили удалить его? Тогда вам придется заглянуть в главу, посвященную API-интерфейсу `Web Storage`!





Подходящее время.

Мы стремились продемонстрировать вам подробный пример того, как HTML-разметка и JavaScript в сочетании друг с другом позволяют создавать интерактивные веб-приложения. Если вдуматься, мы с вами уже многое делали:

- 1) вставляли код на страницу;
- 2) познакомились с событиями `click`, иницируемые при щелчке на кнопке, и написали код для перехвата и обработки данных событий;
- 3) извлекали информацию из объектной модели документа;
- 4) создавали и добавляли новые элементы в DOM.

Неплохо! А теперь, когда у вас сложилось некоторое интуитивное представление о том, как действует весь этот механизм, давайте сделаем небольшой крюк по авеню JavaScript и посмотрим, как именно работают функции и объекты.

Нет, это не будет заурядная прогулка, поскольку мы даже заглянем под крышки люков на дороге и разберемся в том, как функционирует Вебвилль.

Заинтересовались? Тогда присоединяйтесь к нам в главе 4...

КЛЮЧЕВЫЕ МОМЕНТЫ

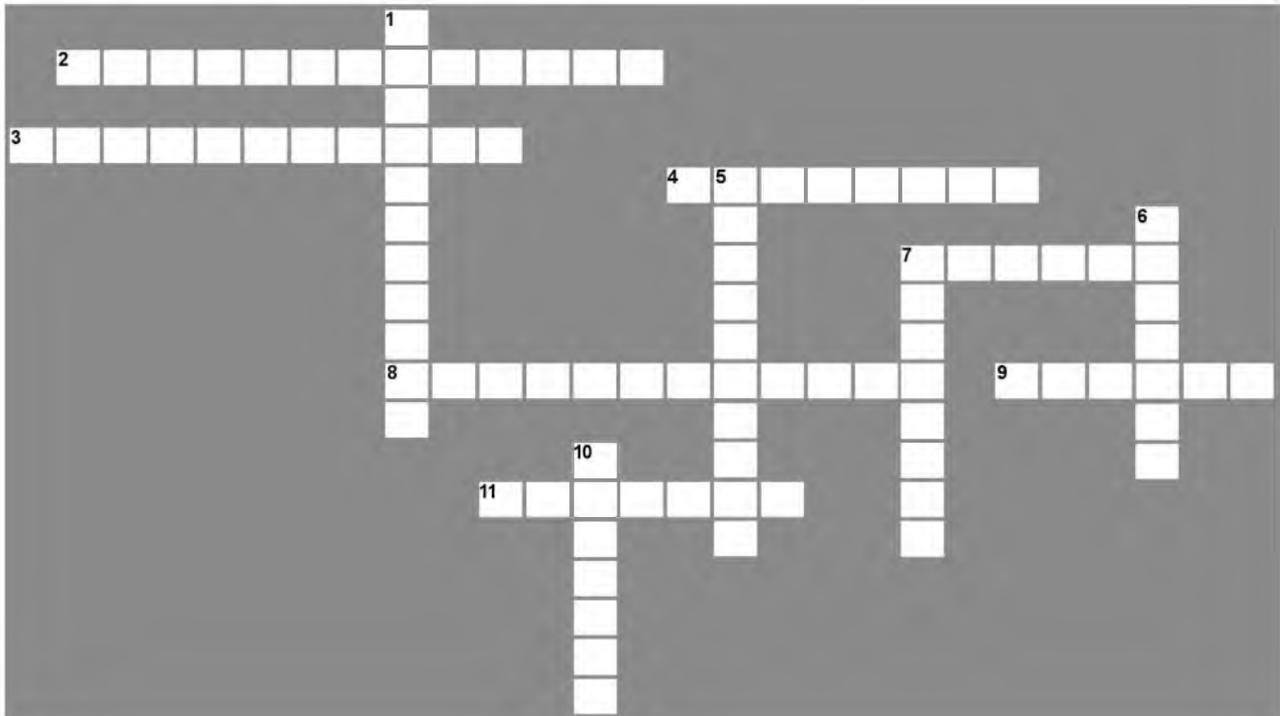


- В браузере постоянно происходит масса событий. Если вы хотите обеспечить реакцию на эти события, то вам потребуется обрабатывать их с помощью обработчиков событий.
- Событие `click` инициируется при нажатии кнопки на странице.
- Для обработки событий `click` необходимо зарегистрировать функцию, которая и займется их обработкой. Для этого нужно сначала написать функцию, а затем присвоить ее имя свойству кнопки `onclick`.
- Если обработчик событий `click` зарегистрирован, то данная функция будет вызываться, когда пользователь щелкнет на кнопке с помощью мыши.
- Для реагирования на события `click` необходимо написать соответствующий код в функции обработчика. Можно выводить сообщения для пользователя в диалоговых окнах `alert`, обновлять страницу, а также выполнять другие действия.
- Для извлечения данных, введенных пользователем в текстовое поле формы, необходимо использовать свойство `value` поля ввода.
- Если пользователь ничего не ввел в текстовое поле формы, значением данного поля будет пустая строка (`" "`).
- Вы можете проводить сравнение переменной с пустой строкой при помощи оператора `if` и `==`.
- Для добавления нового элемента в объектную модель документа его сначала нужно создать, а затем добавить в качестве дочернего элемента по отношению к другому элементу.
- Используйте `document.createElement` для создания новых элементов. Передайте имя тега (например, `"li"`) в вызов функции, чтобы указать, какой элемент вы хотите создать.
- Для добавления элемента в качестве дочернего по отношению к родительскому элементу в DOM необходимо извлечь ссылку на родительский элемент и вызвать в отношении него `appendChild`, передав при этом дочерний элемент, который требуется добавить.
- При добавлении множественных дочерних элементов в родительский элемент с использованием `appendChild` каждый новый дочерний элемент будет добавляться после всех прочих уже имеющихся там дочерних элементов, в результате чего они окажутся расположены позади или под дочерними элементами, которые уже присутствовали на странице (при условии, что вы не станете вносить изменения в макет посредством CSS).
- Вы можете использовать API-интерфейс Web Storage (`localStorage`) для сохранения данных в браузере пользователя.
- Мы применяли `localStorage` для сохранения плейлиста с песнями, используя при этом заранее подготовленный код. Подробнее о `localStorage` вы узнаете в главе 9.
- В следующей главе мы еще больше расскажем о DOM и JavaScript-возможностях, таких как функции и объекты.



HTML5-Кроссворд

Дайте своему мозгу некоторое время, чтобы он усвоил аспекты взаимодействия HTML с JavaScript. Поразмышлите над тем, как они работают сообща. А пока вы будете делать это, разгадайте приведенный ниже кроссворд для закрепления материала. Все использованные в нем слова взяты из данной главы.



По горизонтали

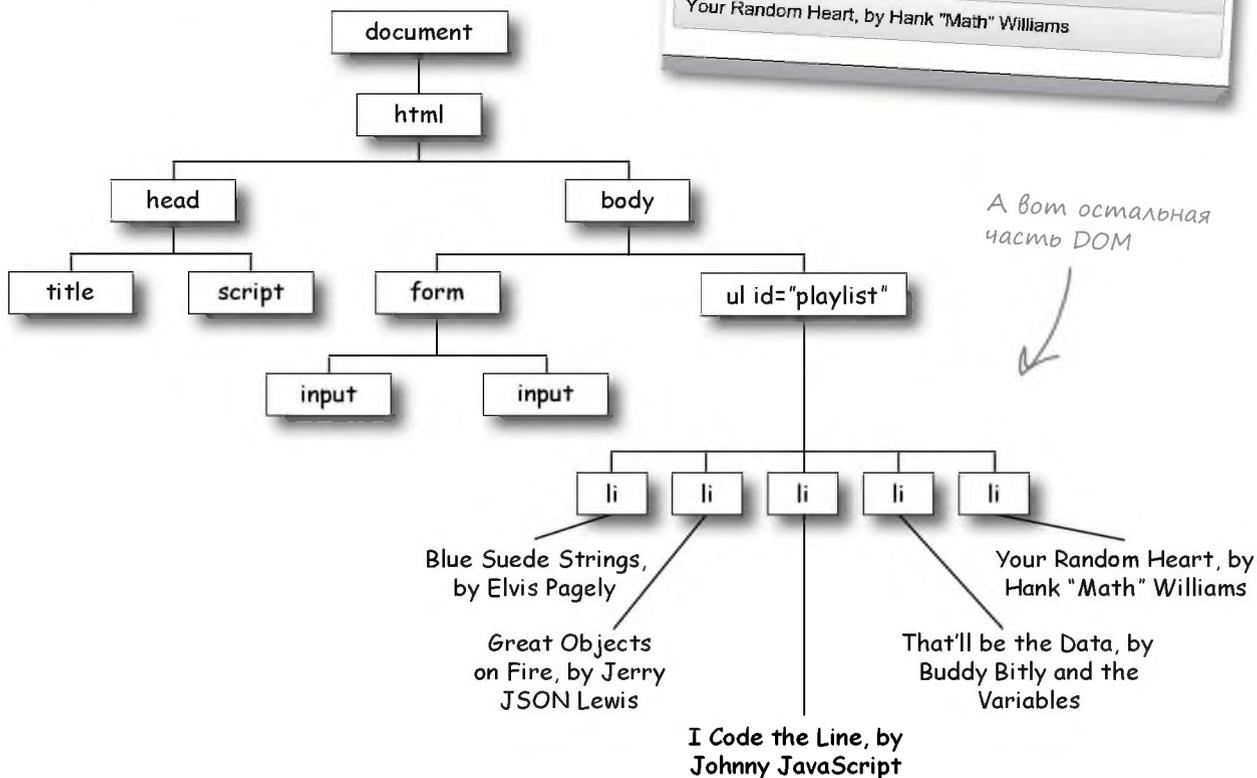
2. Метод для создания новых элементов, внедряемых в объектную модель документа (DOM).
3. Метод для добавления новых элементов в DOM.
4. «Мать» дерева объектной модели документа.
7. Объектная модель документа является чем-то вроде родословного _____.
8. Мы использовали его в заранее подготовленном коде, чтобы стало возможным сохранение плейлиста с песнями.
9. В случае, если пользователь ничего не введет, значением по умолчанию элемента ввода формы будет _____ строка.
11. Оно происходит, когда пользователь щелкает на кнопке.

По вертикали

1. Исполнитель, имя которого мы использовали в примере с песнями.
5. Код, который занимается обработкой событий _____.
6. В случае с кнопкой click — это _____.
7. Новый элемент добавляется в качестве _____ элемента.
10. Что мы будем рассматривать в следующей главе? Функции и _____.



Здесь приведен плейлист, для которого вам необходимо нарисовать объектную модель документа в том виде, который он приобретет после добавления всех этих песен. Обратите внимание на порядок, в котором песни были добавлены на страницу, и разместите соответствующие элементы на своих местах в DOM. Наше решение выглядит так.



Пришлось ли вам строить какие-либо предположения насчет порядка, в котором элементы `` добавляются в родительский элемент?

Да, поскольку он влияет на порядок отображения песен на странице; `appendChild` всегда добавляет новый элемент после дочернего элемента, который уже там был до него.



HTML5-кроссворд. Решение



4 функции и объекты javascript

Серьезный JavaScript



Можете ли вы уже назвать себя создателем сценариев? Вполне возможно, поскольку вы уже многое знаете о JavaScript, однако кто захочет быть простым писателем сценариев, когда можно быть программистом? Пора проявить серьезность и поднять планку, — настало время познакомиться с **функциями** и **объектами**. Они являются ключом к написанию более эффективного, лучше организованного и легкого в сопровождении кода. Они также активно используются наряду с API-интерфейсами HTML5 JavaScript, поэтому чем лучше вы будете в них разбираться, тем быстрее сможете освоиться с тем или иным новым API-интерфейсом и начать его использовать. Пристегнитесь, поскольку эта глава потребует вашего всецелого внимания...

Расширяем ваш словарный запас

Вы уже многое умеете делать с помощью JavaScript. Давайте взглянем на то, что вам уже по силам:



```

<script>
  var guessInput = document.getElementById("guess");
  var guess = guessInput.value;
  var answer = null;

  var answers = [ "red",
                  "green",
                  "blue"];

  var index = Math.floor(Math.random() * answers.length);

  if (guess == answers[index]) {
    answer = "You're right! I was thinking of " + answers[index];
  } else {
    answer = "Sorry, I was thinking of " + answers[index];
  }

  alert(answer);
</script>

```

Извлечение элемента из объектной модели документа (DOM).

Извлечение значения текстового поля ввода формы.

Создание нового массива, заполненного строками.

Использование библиотек функций.

Извлечение такого свойства массива, как length.

Принятие решений на основе результатов проверки условий.

Использование элементов массива.

Использование браузерных функций (например, alert).

Однако пока многие ваши знания являются неформальными. Вы, конечно, знаете, как извлекать элементы из объектной модели документа (DOM), присваивать новые значения, но если мы попросим вас объяснить, что такое `document.getElementById` в техническом плане, вероятно, у вас возникнут с этим некоторые трудности. Но не стоит беспокоиться, поскольку к концу главы вы сможете без труда сделать это.

А чтобы вам это действительно удалось, мы не станем начинать с глубокого анализа `getElementById`, а займемся кое-чем *более интересным*: расширим ваш словарный запас в области JavaScript и научимся делать новые вещи.

Как добавить свои собственные функции

Ранее мы с вами использовали встроенные функции, например `alert` или даже `Math.random`, но вдруг у вас возникнет необходимость добавить свою собственную функцию? Допустим, мы захотели написать код вроде следующего:

```
var guessInput = document.getElementById("guess");
var guess = guessInput.value;

var answer = checkGuess(guess);
alert(answer);
```

Мы извлекаем `guess` пользователя точно так же, как и на предыдущей странице...

...однако весь остальной код, который приведен на предыдущей странице и является частью основного кода, мы можем заменить на изящную функцию `checkGuess`, которую сможем вызывать и которая делает то же самое.

Создание функции `checkGuess`

- 1 Для создания функции необходимо воспользоваться ключевым словом `function`, после которого следует указать имя, например `checkGuess`.

```
function checkGuess(guess) {
    var answers = [ "red",
                    "green",
                    "blue"];

    var index = Math.floor(Math.random() * answers.length);

    if (guess == answers[index]) {
        answer = "You're right! I was thinking of " + answers[index];
    } else {
        answer = "Sorry, I was thinking of " + answers[index];
    }
    return answer;
}
```

- 2 Своей функции вы можете передавать ноль и более параметров. Используйте параметры для передачи значений функции. Здесь нам требуется один параметр: `guess` пользователя.

- 4 Опционально можно возвращать значение в качестве результата вызова функции. Здесь мы возвращаем строку с сообщением.

- 1 Задайте тело своей функции, которое следует заключить в фигурные скобки. Тело содержит весь код, который обеспечивает выполнение работы функции. Здесь в теле мы повторно используем код с предыдущей страницы.

Как работает функция

Так как же все это работает? Что происходит, когда мы *действительно* вызываем функцию? Давайте взглянем в общих чертах.

Итак, сначала нужно создать функцию.

Допустим, мы только что написали новую функцию `bark`, у которой имеется два параметра: `dogName` и `dogWeight`, а также весьма занимательный фрагмент кода, возвращающий кличку собаки и то, что она лает, в зависимости от того, какой она имеет вес.

Вот наша удобная функция `bark`.

```
function bark(dogName, dogWeight) {  
    if (dogWeight <= 10) {  
        return dogName + " says Yip";  
    } else {  
        return dogName + " says Woof";  
    }  
}
```

Теперь давайте вызовем функцию!

Вы уже знаете, как вызывать функцию: нужно просто ввести ее имя и передать ей все необходимые аргументы. В данном случае нам потребуется передать два аргумента: строку с кличкой собаки и вес собаки в виде целого числа.

Давайте выполним вызов и посмотрим, как все это работает:

При вызове `bark` аргументы присваиваются именам параметров в функции `bark`.

И каждый раз, когда параметры встречаются в функции, будут использоваться переданные нами значения.

Имя нашей функции

Здесь мы передаем два аргумента: кличку и вес.

```
bark("Fido", 50);
```

```
function bark(dogName, dogWeight) {  
    if (dogWeight <= 10) {  
        return dogName + " says Yip";  
    } else {  
        return dogName + " says Woof";  
    }  
}
```

А теперь давайте позволим телу функции сделать свою работу.

После того как мы присвоили значение каждому аргументу соответствующему параметру в функции (например, "Fido" для dogName и целое число 50 для dogWeight), мы готовы переходить к оценке всех операторов в теле функции.

Операторы оцениваются сверху вниз, как и любой другой код, который вы пишете. Разница заключается в том, что мы будем делать это в среде, где имена параметров dogName и dogWeight присваиваются аргументам, переданным в функцию.

```
function bark(dogName, dogWeight) {
  if (dogWeight <= 10) {
    return dogName + " says Yip";
  } else {
    return dogName + " says Woof";
  }
}
```

Здесь мы оцениваем весь код в теле функции.

Опционально у нас могут иметься операторы return в теле функции...

... где мы будем возвращать значение коду, совершившему вызов. Давайте посмотрим, как это работает:

Помните, что функции необязательно должны возвращать значения. Однако в данном случае функция bark возвращает значение.

Строка "Fido says Woof" возвращается вызывающему коду (то есть коду, вызвавшему функцию bark).

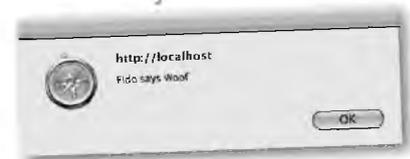
А когда строка возвращается, она присваивается переменной sound, которая затем передается alert, в результате чего на экран выводится диалоговое окно с соответствующим сообщением.

Как видно в функции, dogWeight не меньше или не равен 10, поэтому мы используем предложение else и возвращаем "Fido says Woof" в качестве строкового значения.

```
function bark(dogName, dogWeight) {
  if (dogWeight <= 10) {
    return dogName + " says Yip";
  } else {
    return dogName + " says Woof";
  }
}
```

"Fido says Woof"

```
var sound = bark("Fido", 50);
alert(sound);
```



Я неустанно твержу вам, что все API-интерфейсы HTML5 битком набиты функциями, объектами и прочими продвинутыми JavaScript-возможностями...



Если бы у нас была лишняя минутка поговорить...

Вы думали, что в главе 4 будете активно познать новое в HTML5? Конечно же, так и будет. Однако прежде вам обязательно нужно разобраться в том, что именно является *основой* API-интерфейсов JavaScript в HTML5, чем мы и займемся в данной главе.

Так что же является этой основой? Считайте, что API-интерфейсы JavaScript в HTML5 состоят из объектов, методов (также называемых *функциями*) и свойств. Чтобы постоянно овладеть API-интерфейсами JavaScript, вам необходимо хорошо разбираться в этих вещах. Естественно, вы можете попытаться обойтись и без знаний о них, однако в таком случае при использовании API-интерфейсов вам придется постоянно строить догадки, что не позволит вам полностью задействовать их потенциал (не говоря уже о том, что вы будете совершать массу ошибок и писать некачественный код).

Мы решили обратить ваше внимание на данные аспекты, чтобы вы знали, что вас ожидает впереди. Самое замечательное заключается в том, что к концу главы вы будете разбираться в объектах, функциях, методах и массе прочих связанных с ними вещей лучше, чем примерно 98 % людей, занимающихся написанием сценариев. И это не шутка.



ВСТРЕЧАЕМ ФУНКЦИЮ

Интервью недели:

Вещи, о которых вы не знали

Head First: Добро пожаловать, Функция! Надеюсь, сегодня Вы поведаете нам многое о себе.

Function: Рада быть здесь.

Head First: Мы заметили, что в настоящее время многие повички в JavaScript не склопы слишком часто использовать Вас. Они просто пишут свой код, строка за строкой, двигаясь сверху вниз. Почему им стоило бы уделять Вам больше внимания?

Function: Да, не склопы, и это прискорбно, поскольку я песу в себе большие возможности. Думайте обо мне так: я — то, что позволяет вам написать код один раз, а затем повторно использовать его.

Head First: Простите, что спрашиваю, но если Вы просто даете им возможность делать одни и те же вещи снова и снова... это ведь немного скучно, не так ли?

Function: Нет-нет, функции являются параметризованными. Другими словами, каждый раз, когда вы используете функцию, вы можете передавать ей аргументы, благодаря чему сможете получать разные результаты, которые будут отличаться друг от друга в зависимости от того, что вы ей передали.

Head First: А нельзя ли привести пример?

Function: Допустим, вам необходимо сообщать пользователям стоимость товаров, которые они поместили в свою электронную корзину в интернет-магазине, поэтому вы решили написать функцию `computeShoppingCartTotal`. Сделав это, вы сможете передавать функции различные электронные корзины, принадлежащие разным пользователям, и каждый раз получать соответствующее значение стоимости товаров в конкретной корзине.

...Кстати, возвращаясь к Вашему замечанию насчет того, что повички не очень часто используют функции; на самом деле это не так, поскольку

они используют их постоянно: `alert`, `document.getElementById`, `Math.random`. Они просто не определяют *свои собственные* функции.

Head First: Что ж, верно, в случае с `alert` все понятно, но вот другие две совсем не похожи на функции.

Function: Это функции, видите ли... постойте-ка, минуточку...

...Мне только что сказали, что читатели еще не познакомились с этими типами функций, однако они сделают это через несколько страниц. Так или иначе, следует отметить, что функции используются повсюду.

Head First: Итак, функция возвращает значение, правильно? В связи с этим хочу спросить: а что, если у меня нет значения, которое я хочу вернуть?

Function: Многие функции возвращают значения, однако функция вовсе не обязательно должна поступать так. Есть немало функций, которые просто делают что-то, например обновляют объектную модель документа (DOM), после чего не возвращают никакого значения, и все прекрасно.

Head First: То есть в таких функциях отсутствуют операторы `return`?

Function: Да, именно так.

Head First: Что ж, а как насчет присваивания имен своим функциям, а то мне доводилось слышать, что это делать не обязательно, если не хочется.

Function: Давайте не будем излишне пугать аудиторию. Давайте вернемся к этой теме, когда читатели узнают обо мне немного больше?

Head First: Обещаете мне дать эксклюзивное интервью?

Function: Мы это еще обсудим...

Не уверена, что понимаю разницу между параметром и аргументом: это случайно не одно и то же?

Нет, это разные понятия.

При определении функции вы можете *определить* ее с использованием одного или более *параметров*.

Здесь мы определяем три параметра: *degrees*, *mode* и *duration*.

```
function cook(degrees, mode, duration) {  
  // здесь будет ваш код  
}
```

При вызове функции вы *вызываете* ее с использованием *аргументов*:

```
cook(425.0, "bake", 45);
```

↑ ↑ ↑

Это аргументы. Здесь имеются три аргумента: число с плавающей точкой, строковое значение и целое число.

↙ ↓ ↓

```
cook(350.0, "broil", 10);
```

Таким образом, свои параметры вы будете определять только один раз, а вызывать свои функции будете с использованием массы разных аргументов.

Вас удивит, насколько много людей путается в том, где используются параметры, а где аргументы. В некоторых книгах даже встречается неверное толкование, поэтому если вы где-то увидите, что утверждается обратное, то будете знать, где правда...

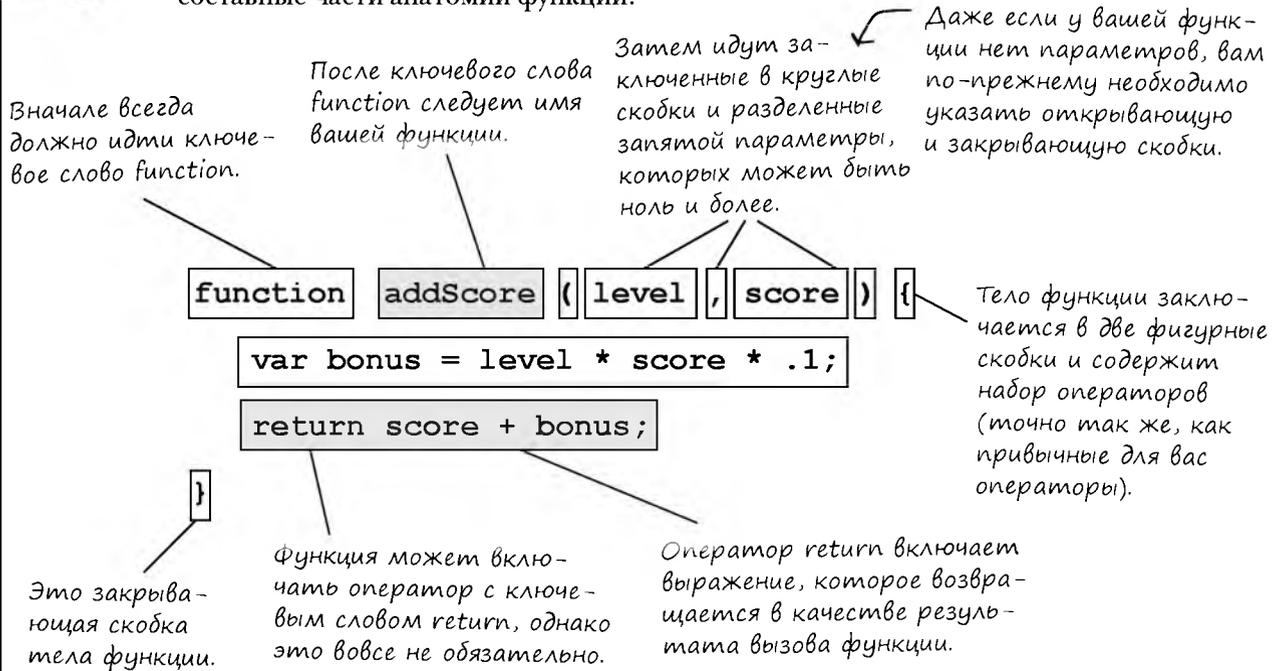
**Функцию определяют с использованием параметров,
а вызывают с использованием аргументов.**





Анатомия функции

Теперь, когда вы знаете, как определять и вызывать функции, давайте убедимся, что вы хорошо разбираетесь в синтаксисе. Вот как выглядят составные части анатомии функции:



Часто задаваемые Вопросы

В: Почему перед именами параметров не ставится `var`? Параметр — это же новая переменная, ведь так?

О: Да, так и есть. Функция сделает за вас всю работу по созданию экземпляра переменной, поэтому вам не нужно указывать ключевое слово `var` перед именами своих параметров.

В: Каковы правила присваивания имен функциям?

О: Они аналогичны правилам присваивания имен переменным.

В: Я передаю переменную своей функции; если я изменю значение соответствующего параметра в моей функции, то приведет ли это также к изменению моей исходной переменной?

О: Нет. Когда вы передаете примитивное значение, оно копируется в параметр. Мы называем это «передача по значению». Таким образом, если вы измените значение параметра в теле своей функции, то это никак не повлияет на оригинальное значение вашего аргумента. Исключением здесь является передача массива или объекта, но об этом мы поговорим чуть позже.

В: Смогу ли я изменять значения в функции?

О: Вы сможете изменять только значения глобальных переменных (они определяются вне функций) или переменных, которые вы явно определили в своей функции. Вскоре мы чуть подробнее затронем эту тему.

В: Что возвращает функция, если в ней отсутствует оператор `return`?

О: Функция без оператора `return` возвращает значение `undefined`.

Возьми в руку карандаш



Воспользуйтесь своими знаниями о функциях и передаче аргументов параметрам для оценки приведенного ниже кода. Сделав это, напишите внизу, какое значение будет иметь каждая из переменных. Проверьте свои ответы, посмотрев решение этого задания в конце главы, прежде чем двинетесь дальше.

```
function dogsAge(age) {
    return age * 7;
}

var myDogsAge = dogsAge(4);

function rectangleArea(width, height) {
    var area = width * height;
    return area;
}

var rectArea = rectangleArea(3, 4);

function addUp(numArray) {
    var total = 0;
    for (var i = 0; i < numArray.length; i++) {
        total += numArray[i];
    }
    return total;
}

var theTotal = addUp([1, 5, 3, 9]);

function getAvatar(points) {
    var avatar;
    if (points < 100) {
        avatar = "Mouse";
    } else if (points > 100 && points < 1000) {
        avatar = "Cat";
    } else {
        avatar = "Ape";
    }
    return avatar;
}

var myAvatar = getAvatar(335);
```

Напишите
здесь, какое
значение бу-
дет иметь
каждая из
переменных...

myDogsAge =

rectArea =

theTotal =

myAvatar =

Нам нужно поговорить о некоторых особенностях использования переменных...



Локальные и глобальные переменные

Нужно знать разницу между ними

Вы уже знаете, что можете объявлять переменные с использованием ключевого слова `var` и имени где угодно в своем JavaScript-коде:

```
var avatar;
var levelThreshold = 1000;
```

← Это глобальные переменные; они повсеместно доступны в вашем JavaScript-коде.

Вам также известно, что переменные можно объявлять и внутри функции:

```
function getScore(points) {
  var score;
  for (var i = 0; i < levelThreshold; i++) {
    //code here
  }
  return score;
}
```

Переменные `points`, `score` и `i` объявляются внутри функции.

Мы называем их локальными переменными, потому что они доступны только локально в самой функции.

Даже если мы используем `levelThreshold` внутри функции, она будет глобальной переменной, поскольку объявлена вне функции.

Однако какое все это имеет значение? Переменные есть переменные, не так ли? Что ж, от того, где вы объявляете свои переменные, зависит, насколько они будут видны для остальной части вашего кода. А понимание того, как работают две эти разновидности переменных, впоследствии поможет вам писать более легкий в сопровождении код (не говоря уже о том, что это поможет вам разобраться в коде, написанном другими людьми).

Если переменная объявлена в функции, то она является ГЛОБАЛЬНОЙ. Если же переменная объявлена внутри функции, то она — ЛОКАЛЬНАЯ

Понятие области видимости локальных и глобальных переменных

От того, где вы определяете свои переменные, будет зависеть их *область видимости*; то есть там, где переменные определены и не определены, они соответственно будут видимы и невидимы для вашего кода. Давайте взглянем на пример, где присутствуют переменные как с локальной, так и с глобальной областью видимости. Помните, что переменные, которые вы определяете вне функции, будут обладать глобальной областью видимости, а переменные, определяемые внутри функции, — локальной областью видимости.

```

var avatar = "generic";
var skill = 1.0;
var pointsPerLevel = 1000;
var userPoints = 2008;

function getAvatar(points) {
    var level = points / pointsPerLevel;

    if (level == 0) {
        return "Teddy bear";
    } else if (level == 1) {
        return "Cat";
    } else if (level >= 2) {
        return "Gorilla";
    }
}

function updatePoints(bonus, newPoints) {
    for (var i = 0; i < bonus; i++) {
        newPoints += skill * bonus;
    }
    return newPoints + userPoints;
}

userPoints = updatePoints(2, 100);
avatar = getAvatar(2112);
    
```

Эти четыре переменные обладают глобальной областью видимости, то есть они определены и видимы во всем приведенном ниже коде.

Следует отметить, что если вы укажете ссылки на дополнительные сценарии в своей странице, то они тоже смогут увидеть глобальные переменные!

Переменная `level` здесь является локальной и видимой только для кода внутри функции `getAvatar`. Это означает, что только данная функция сможет получить доступ к переменной `level`.

Не будем забывать о параметре `points`, который также обладает локальной областью видимости в функции `getAvatar`.

Обратите внимание, что в случае с `getAvatar` также используется глобальная переменная `pointsPerLevel`.

В `updatePoints` у нас имеется локальная переменная `i`. Данная переменная видима для всего кода в `updatePoints`.

`bonus` и `newPoints` также являются локальными по отношению к `updatePoints`, в то время как `userPoints` является глобальной переменной.

А здесь в коде мы можем использовать только глобальные переменные, при этом у нас нет доступа к каким-либо переменным внутри функций, поскольку они невидимы в глобальной области.

Недолгая жизнь переменных

Если вы — переменная, то вам приходится легко и ваша жизнь может оказаться короткой. Так и будет, если вы, копецко, не являетесь глобальной переменной, хотя даже в случае с ними продолжительность жизни ограничена. Но что определяет длительность существования переменной? Дело обстоит следующим образом:

Глобальные переменные существуют столько же, сколько и соответствующая веб-страница. Жизнь глобальной переменной начинается, когда ее JavaScript загружается в страницу. Однако жизнь вашей глобальной переменной закончится, как только пользователь покинет веб-страницу. Даже если он перезагрузит ту же самую страницу, все ваши глобальные переменные будут уничтожены, а затем воссозданы в запово загруженной странице.

Локальные переменные обычно исчезают по завершении работы функции. Локальные переменные создаются при первом вызове вашей функции и живут до тех пор, пока эта функция не закончит свою работу (возвратив при этом значение или же нет). Следует отметить, что вы можете взять значения своих локальных переменных и вернуть их из функции до того, как эти переменные представят перед цифровым творцом на том свете.

Так что же, получается, «сбежать» от страницы никак НЕЛЬЗЯ? Если вы — локальная переменная, то ваша жизнь пролетает быстро. Но если вам повезло родиться глобальной переменной, то вы будете здравствовать, пока пользователь не перезагрузит страницу в браузере.

Однако все-таки *должен* существовать способ «сбежать» от страницы! Мы можем его найти! Как вы считаете?

Готов поклясться, что переменная была прямо у меня за спиной, но когда я обернулся, она уже исчезла...



Мы сказали «обычно», поскольку существуют особые способы сохранить жизнь локальным переменным на более длительный срок, однако мы не станем беспокоиться о них сейчас.

Присоединяйтесь к нам в главе, посвященной API-интерфейсу Web Storage, где мы поможем вашим данным избежать «ужасающих» для них последствий перезагрузки веб-страницы!



Интересно, что будет, если я присвою локальной переменной то же имя, которое имеет уже существующая глобальная переменная?

Помещаем «в тень» глобальную переменную.

Вот что мы имеем в виду. Допустим, у вас есть глобальная переменная `beanCounter` и вы объявляете функцию, как показано далее:

```
var beanCounter = 10;

function getNumberOfItems(ordertype) {
  var beanCounter = 0;
  if (ordertype == "order") {
    // сделать что-то с помощью beanCounter...
  }
  return beanCounter;
}
```

← Здесь у нас имеются и глобальная и локальная переменные!

Когда вы сделаете это, любые ссылки на `beanCounter` внутри функции будут вести к локальной переменной, а не к глобальной. Таким образом, мы говорим, что глобальная переменная будет паходиться «в тени» локальной переменной (другими словами, мы не сможем ее увидеть, поскольку нам мешает это сделать локальная переменная).

↑ Следует отметить, что локальная и глобальная переменные не влияют друг на друга: если изменить одну из них, то это никак не скажется на другой. Они являются независимыми переменными.

Часть Задаваемые Вопросы

В: При отслеживании области видимости всех этих локальных и глобальных переменных можно запутаться, так почему же не использовать только глобальные переменные? Я именно так всегда и поступаю.

О: Если вы пишете код, который является сложным или будет нуждаться в длительном сопровождении, то нужно внимательно следить за тем, как вы распоряжаетесь своими переменными. Если вы слишком рьяно будете создавать глобальные переменные, вам станет сложно уследить за тем, где они используются (и где вы вносите изменения в значения своих переменных), что может привести к написанию некачественного кода. Все это окажется даже еще более важным, если вы будете писать код совместно с коллегами или станете использовать сторонние библиотеки (даже если они окажутся грамотно написанными, то все равно должны быть структурированы во избежание проблем).

Таким образом, глобальные переменные следует использовать там, где это имеет смысл, но при этом необходимо проявлять умеренность, и всякий раз, когда предоставляется возможность, делать свои переменные локальными. По мере увеличения опыта работы с JavaScript вы будете узнавать дополнительные методики структурирования кода, чтобы он оказался более легким в сопровождении.

В: У меня имеется глобальная переменная на странице, однако я загружаю еще и дополнительные JavaScript-файлы. Эти файлы будут видеть отдельные наборы глобальных переменных?

О: Поскольку существует только одна глобальная область видимости, каждый файл, который вы загружаете, будет видеть один и тот же набор переменных (и генерировать глобальные переменные в одном и том же пространстве). Вот почему так важно быть внимательным при использовании переменных — это позволит избежать конфликтов (кроме того, необходимо стараться снизить количество или вообще убрать глобальные переменные, если это представляется возможным).

В: Мне доводилось видеть код, в котором не используется ключевое слово `var` при присваивании значения имени новой переменной. Как такое возможно?

О: Да, так можно поступать; когда вы присваиваете значение имени переменной, которая ранее не была объявлена, она будет рассматриваться как новая глобальная переменная. Поэтому будьте осторожны, если делаете это внутри функции, в которой

создаете глобальную переменную. Следует отметить, что мы не рекомендуем прибегать к такой практике кодирования; не только потому, что она потенциально способна сделать ваш код неудобочитаемым, но и потому, что, как считают некоторые люди, данное поведение может однажды измениться в реализациях JavaScript (что, возможно, приведет к нарушению вашего кода).

В: Нужно ли мне определять функцию, прежде чем использовать ее, и может ли объявление функции появиться где угодно в моем сценарии?

О: Объявления функций могут появляться где угодно в вашем сценарии. Если захотите, вы можете объявить функцию ниже, там, где будете использовать ее. Это можно делать, поскольку когда вы загружаете свою страницу в первый раз, браузер производит разбор всего JavaScript, имеющегося на странице (или располагающегося во внешнем файле), и видит объявление функции до того, как начнет выполнение кода. Вы также можете размещать объявления глобальных переменных в любой части своего сценария, однако мы рекомендуем объявлять все свои глобальные переменные в верхней части файлов, чтобы их легче было отыскать.

При использовании более одного внешнего JavaScript-файла следует учитывать, что при наличии двух функций с одинаковым именем в разных файлах будет использоваться та, которую браузер увидит последней.

В: Кажется, что все жалуются на чрезмерное использование глобальных переменных в JavaScript. Почему так происходит? Язык JavaScript был не слишком удачно спроектирован, или же люди не знают, что делают, или причина в чем-то еще? И как нам быть в данной ситуации?

О: Глобальные переменные часто с излишком используются в JavaScript. Отчасти причина этого заключается в том, что язык JavaScript позволяет легко освоиться и начать писать код, и это хорошо, поскольку JavaScript не навязывает вам массу всяких структур и прочей нагрузки. Недостатки начнут проявляться при написании серьезного кода, который будет претерпевать изменения и нуждаться в сопровождении в течение длительного времени (что в большой степени характеризует все веб-страницы). Следует отметить, что JavaScript является мощным языком и включает различные возможности, например объекты, которые вы можете использовать для организации своего кода модульным образом. На эту тему написано много книг, а «попробовать на вкус» объекты вы сможете во второй части этой главы (до нее осталось всего несколько страниц).

Функции еще являются и значениями

Итак, мы с вами использовали переменные для размещения числовых и логических значений, строк, массивов и всего прочего, но упомянули ли мы о том, что вы также можете присваивать функцию переменной? Взгляните на следующий пример:

```
function addOne(num) {
    return num + 1;
}

var plusOne = addOne;

var result = plusOne(1);
```

Определим простую функцию, которая прибавляет 1 к своему аргументу.

Теперь сделаем кое-что новенькое. Мы воспользуемся именем функции `addOne` и присвоим `addOne` новой переменной `plusOne`.

Обратите внимание, что мы не вызываем функцию посредством `addOne()`, а просто указываем имя функции.

`plusOne` присваивается функции, поэтому мы можем вызывать ее с использованием целочисленного аргумента в виде `1`.

После выполнения данного вызова `result` будет равен 2.

Что же, ранее мы не только забыли упомянуть об этой небольшой детали, касающейся функций, но также не совсем были честны, когда рассказывали вам об атомности функции, — оказывается, вам даже не нужно присваивать имя своей функции. Все верно: ваша функция может быть *анонимной*. Что же все это значит и зачем вам может потребоваться поступить так? Давайте сначала посмотрим, как создать функцию без имени:

```
function(num) {
    return num + 1;
}

var f = function(num) {
    return num + 1;
}

var result = f(1);
alert(result);
```

Здесь мы создаем функцию, не указывая при этом имени... Хм... Но как же мы тогда сможем сделать что-нибудь при помощи нее?

Давайте сделаем это снова и на этот раз присвоим ее переменной.

А затем мы сможем использовать нашу переменную для вызова функции.

После выполнения данного вызова `result` будет равен 2.



МОЗГОВОЙ ШТУРМ

Взгляните на данный код: как вы думаете, что он делает?

```
var element = document.getElementById("button");
element.onclick = function () {
    alert("clicked!");
}
```

Все это должно быть для вас немного более понятным, учитывая то, что мы сейчас с вами рассмотрели...

Не беспокойтесь, если вы так и не сможете на 100 % во всем здесь разобраться, поскольку мы еще дойдем до этого...

Что можно сделать посредством функций как значений?

Так в чем же здесь важность? В чем польза? Важность заключается не столько в том, что мы можем присваивать функцию переменной, сколько в том, что это наш способ показать вам, что функция действительно является значением. А вы уже знаете, что можете сохранять значения в переменных или массивах, передавать их в качестве аргументов функциям или, как мы вскоре увидим, присваивать их свойствам объектов. Однако вместо того, чтобы рассказывать вам о полезности аномальных функций, просто взглянем на один из множества любопытных способов использования функций как значений:

```
function init() {
    alert("you rule!");
}
window.onload = init;
```

Вот простая функция `init`

Здесь мы присваиваем определенную нами функцию обработчику событий `onload`.

Посмотрите-ка, мы уже использовали функции как значения!

Либо мы можем поступить еще интереснее:

Здесь мы создаем функцию без явного имени, а затем напрямую передаем ее значение свойству `window.onload`.

```
window.onload = function() {
    alert("you rule!");
}
```

Красота! Разве это не проще и удобнее для чтения?

Не беспокойтесь, если `window.onload` кажется вам несколько непонятным, поскольку позже мы подробно о нем поговорим.

Как вы уже начали понимать, функции позволяют делать кое-что полезное помимо простой «упаковки» кода для повторного использования. Чтобы лучше разобраться в том, как полностью задействовать все преимущества функций, давайте взглянем на объекты и посмотрим, как они вписываются в JavaScript, после чего объединим их в единый механизм.

Эй, авторы! Еще раз привет!
Я девушка, которая купила вашу книгу о программировании на HTML5, помните меня? Какое все это имеет отношение к HTML5?

Мы полагали, что уже ответили на этот вопрос... но если вам кажется, будто мы подобрали вас и уже подороги провезли окольными путями по городу с включенным счетчиком (в то время как могли бы по прямой отвезти в центр города), что ж, тогда вспомните о том, что *в следующей главе мы собираемся начать изучение API-интерфейсов, работающих с HTML5. А чтобы сделать это, вам потребуется действительно хорошо разбираться* в функциях, объектах и прочих связанных с ними аспектах.

Так что потерпите — фактически, вы уже прошли половину пути! И не забывайте, что в данной главе вы превратитесь из писателя сценариев в программиста, из HTML/CSS-паездника в того, кто способен создавать серьезные приложения.

А мы уже упоминали о том, что все это также может позволить вам заработать намного больше денег?



Благодаря объектам
будущие перспективы выглядят
настолько яркими, что нам действи-
тельно НЕ ОБОЙТИСЬ без солнцеза-
щитных очков...



Кто-то сказал «объекты»?!

О, это наша любимая тема! Объекты перенесут ваши навыки в JavaScript-программировании на следующий уровень — они выступают ключом к управлению сложным кодом, к пониманию объектной модели документа (DOM), к организации ваших данных и даже являются основным способом «унаковки» API-интерфейсов JavaScript в HTML5 (данный список можно продолжить!). С учетом этого вам показалось, что объекты — сложная тема, не так ли? Ха! Мы с вами просто бросимся сломя голову вперед и незамедлительно перейдем к их использованию.

Раскроем вам секрет JavaScript-объектов: они представляют собой всего лишь коллекции свойств. Давайте в качестве примера возьмем, скажем, собаку. Собака обладает рядом свойств:

У всех собак есть набор любимых занятий, например гулять (walks) и приносить обратно брошенный мячик (fetching balls).

Собака (dog)



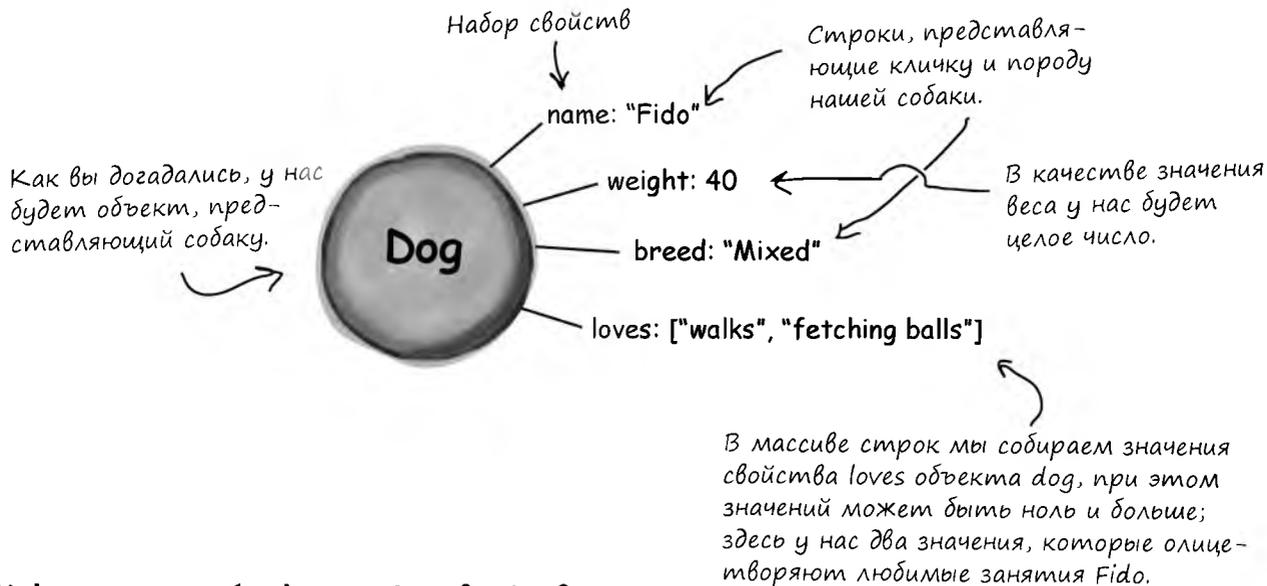
У большинства собак есть клички (name), как, например, Фидо (Fido) в данном случае.

← Каждая собака имеет определенный вес (weight).

И относится к определенной породе. В данном случае у Фидо (Fido) смешанная (mixed) порода (breed).

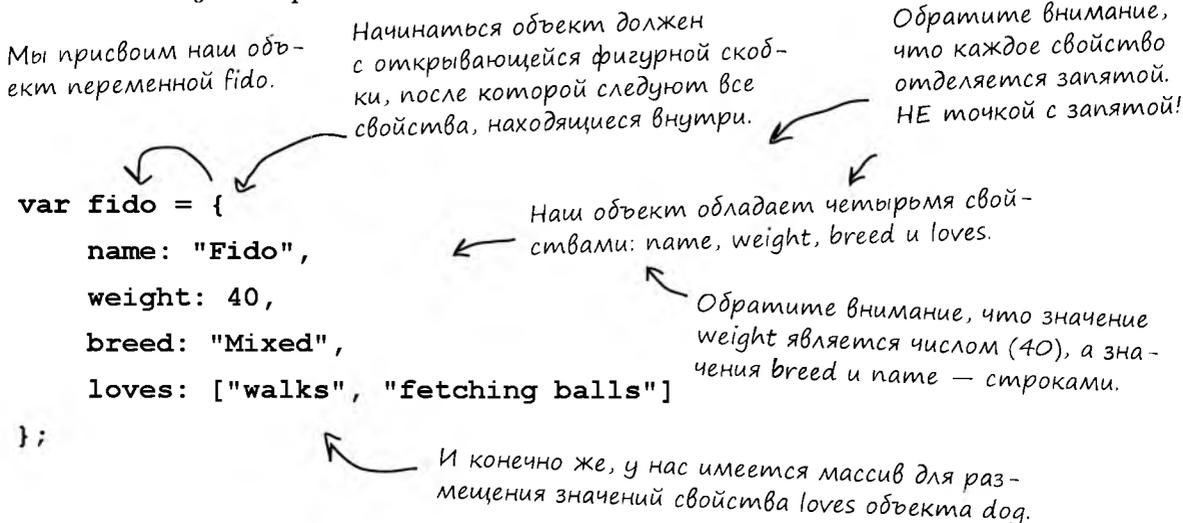
Размышления о свойствах...

Естественно, наш нес Fido, если бы мог говорить, сразу же отметил бы, что у него имеется намного больше свойств, чем мы перечислили выше, однако в данном примере именно их мы и отразим в программном коде. Давайте взглянем на эти свойства с точки зрения типов данных JavaScript:



Как создать объект на JavaScript?

Итак, у нас есть объект с рядом свойств. А как он создается с использованием JavaScript? Вот как:



Что можно сделать с объектами

1 Обращение к свойствам объекта с помощью «точечной» нотации:

```
if (fido.weight > 25) {
  alert("WOOF");
} else {
  alert("yip");
}
```

Используйте объект наряду с точкой и именем свойства для получения доступа к значению этого свойства.

Используйте точку (.)

fido.weight

Вот объект...

... а вот имя свойства.

1 Обращение к свойствам с использованием строки и скобочной нотации:

```
var breed = fido["breed"];
if (breed == "mixed") {
  alert("Best in show");
}
```

Используйте объект наряду с именем свойства, заключенным в кавычки, и квадратные скобки для получения доступа к значению этого свойства.

На этот раз заключите имя свойства в квадратные скобки.

fido["weight"]

Вот объект...

... а вот имя свойства в кавычках.

1 Изменение значения свойства:

```
fido.weight = 27;
```

Мы изменяем значение свойства weight объекта fido...

```
fido.breed = "Chawalla/Great Dane mix";
```

... значение его свойства breed...

```
fido.loves.push("chewing bones");
```

... и добавляем новый элемент в его массив loves.

push просто добавляет новый элемент в конец массива.

Мы считаем, что точечная нотация более удобочитаема, чем скобочная.

1 Перечисление всех свойств объекта:

«Перечислить» означает пройти по всем свойствам объекта.

```
var prop;
for (prop in fido) {
  alert("Fido has a " + prop + " property ");
  if (prop == "name") {
    alert("This is " + fido[prop]);
  }
}
```

Для перечисления свойств мы используем цикл for-in

При каждом выполнении цикла переменная prop получает строковое значение следующего по очереди имени свойства.

Мы используем скобочную нотацию для получения доступа к значению соответствующего свойства.

Обратите внимание: порядок свойств произвольный, так что не стоит рассчитывать на какую-то определенную последовательность.

5 Действия в отношении массива объекта:

```
var likes = fido.loves;
var likesString = "Fido likes";

for (var i = 0; i < likes.length; i++) {
    likesString += " " + likes[i];
}

alert(likesString);
```

Здесь мы присваиваем значение массива `likes` объекта `fido` переменной `likes`.

Мы можем выполнить цикл в отношении массива `likes` и создать `likesString` всех любимых занятий `fido`.

Мы также можем вывести диалоговое окно `alert` с соответствующей строкой.

6 Передача объекта функции:

```
function bark(dog) {
    if (dog.weight > 25) {
        alert("WOOF");
    } else {
        alert("yip");
    }
}

bark(fido);
```

Мы можем передать объект функции точно так же, как в случае с любой другой переменной.

В функции мы можем получить доступ к свойствам объекта как обычно, используя, конечно же, имя параметра для объекта.

Мы передаем `fido` в качестве аргумента функции `bark`, которая ожидает объект `dog`.



Оператор «точка» (.)

Оператор «точка» (.) обеспечивает доступ к свойствам объекта. В целом код получается более удобочитаемым, чем при использовании скобочной нотации (["строка"]):

- `fido.weight` это вес `fido`.
- `fido.breed` это порода `fido`.
- `fido.name` это кличка `fido`.
- `fido.loves` это массив, содержащий значения, олицетворяющие любимые занятия `fido`.



А можно ли добавлять свойства в объекты после того, как они уже были определены?

Да, вы сможете добавлять и удалять свойства в любой момент.

Чтобы добавить свойство в объект, нужно просто присвоить новому свойству значение, как показано далее:

```
fido.age = 5;
```

после чего у fido появится новое свойство: age.

А чтобы удалить свойство, нужно воспользоваться ключевым словом delete:

```
delete fido.age;
```

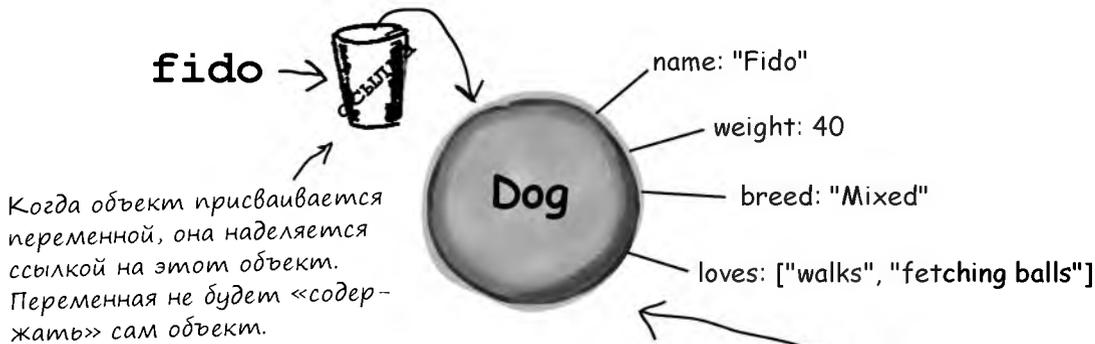
При удалении свойства вы удаляете не просто его значение, а само это свойство. Фактически, если вы захотите воспользоваться `fido.age` после того, как удалили его, то оно будет оценено как `undefined`.

Выражение `delete` возвращает `true`, если свойство было успешно удалено (либо если вы удаляете свойство, которое не существует, либо если то, что вы пытаетесь удалить, не является свойством объекта).

Поговорим о передаче объектов функциям

Мы с вами уже немного обсуждали то, как аргументы передаются функциям, — они передаются *по значению*, так что если мы передадим целое число, соответствующий параметр функции получит *копию* данного целочисленного значения для использования в функции. Аналогичные правила действуют и в случае с объектами, *однако* нам нужно более пристально взглянуть на то, что будет содержаться в переменной, когда она присваивается объекту, чтобы вы смогли понять, что все это означает.

Когда объект присваивается переменной, она будет содержать не сам этот объект, а *ссылку* на него. Считайте ссылку указателем на объект.



Таким образом, при вызове функции и передаче ей объекта вы будете передавать ссылку на объект — не сам объект, а только указатель на него. Копия ссылки передается в параметр, при этом указывать она будет на оригинальный объект.

Если вы вызовете функцию `bark` и передадим ей `fido` в качестве аргумента, то получим копию ссылки на объект `dog`.

```
function bark(dog) {
  ... code here ...
}
```

Так что же все это означает? При изменении свойства объекта вы изменяете данное свойство в оригинальном объекте, а не в копии, поэтому увидите все внесенные вами изменения в объект внутри и вне вашей функции. Давайте рассмотрим пример, где в случае с `dog` используется функция `loseWeight...`

Сажаем Fido на диету...



Давайте взглянем на то, что происходит, когда мы передаем fido функции loseWeight и изменяем свойство dog.weight.

- Мы определили объект fido и передаем его функции loseWeight.

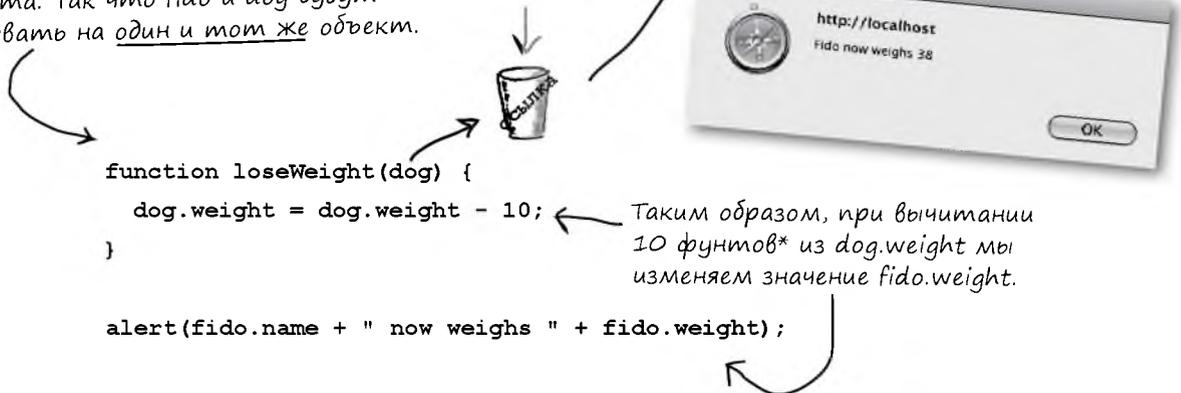
fido — это ссылка на объект, то есть соответствующий объект не располагается в переменной fido, однако эта переменная на него указывает.



- Параметр dog функции loseWeight получает копию ссылки на fido. Таким образом, любые изменения в свойствах параметра будут отражаться на переданном объекте.

Когда мы передаем fido функции loseWeight, параметру dog присваивается копия ссылки, а не копия объекта. Так что fido и dog будут указывать на один и тот же объект.

Ссылка dog — это копия ссылки fido.



```
function loseWeight(dog) {
  dog.weight = dog.weight - 10;
}

alert(fido.name + " now weighs " + fido.weight);
```

* 10 фунтов ≈ 4,54 кг.

ПРИЛОЖЕНИЕ WEBVILLE CINEMA ДЛЯ ПОКАЗА АФИШИ КИНОТЕАТРА

Здесь речь пойдет о приложении Webville Cinema и API-интерфейсах JavaScript; вам нужно будет создать объекты `movie`. Всего вам потребуются два таких объекта, каждый из которых будет включать название (`title`), жанр (`genre`) и рейтинг фильма (`rating`) (выставляемый по шкале от 1 до 5 звездочек), а также время киносеансов (`showtimes`). Вот образцы данных, которые вы сможете использовать для заполнения своих объектов.

Название фильма: Plan 9 from Outer Space.

Время киносеансов: 3:00pm, 7:00pm and 11:00pm.

Жанр: Cult Classic.

Рейтинг в звездочках: 2.

Название фильма: Forbidden Planet.

Время киносеансов: 5:00pm и 9:00pm.

Жанр: Classic Sci-fi.

Рейтинг в звездочках: 5.

↑ Решение к данному заданию находится на следующей странице, однако не заглядывайте туда, пока не сделаете все сами. Серьезно. Мы не шутим.

Код для создания ваших объектов напишите здесь. ↘



↖ Вы можете свободно заменить эти значения теми, которые вам больше нравятся.



ПРИЛОЖЕНИЕ WEBVILLE CINEMA ДЛЯ ПОКАЗА АФИШИ КИНОТЕАТРА

РЕШЕНИЕ

Ну как у вас прошло создание объектов movie?

А вот наше решение к данному заданию:

Мы создали два объекта с именами movie1 и movie2 для двух фильмов.

```
var movie1 = {
  title: "Plan 9 from Outer Space",
  genre: "Cult Classic",
  rating: 5,
  showtimes: ["3:00pm", "7:00pm", "11:00pm"]
};
```

movie1 обладает четырьмя свойствами: title, genre, rating и showtimes.

Значения title и genre являются строками.

Значение rating — это число.

А showtimes представляет собой массив, содержащий значения времени сеансов фильма в виде строк.

```
var movie2 = {
  title: "Forbidden Planet",
  genre: "Classic Sci-fi",
  rating: 5,
  showtimes: ["5:00pm", "9:00pm"]
};
```

У movie2 тоже имеются четыре свойства: title, genre, rating и showtimes.

Не следует забывать, что свойства необходимо разделять запятыми.

Мы использовали те же самые имена свойств, как и в случае с movie1, но значения свойств на этот раз будут уже другие.



Наш следующий сеанс состоится в...



Итак, мы с вами уже немного отведали на вкус, что значит смешивать объекты и функции. Давайте найдем еще дальше и напишем код, который станет выводить сообщение о времени следующего сеанса фильма. Наша функция будет принимать `movie` в качестве аргумента и возвращать строку, содержащую значение времени следующего сеанса фильма, отталкиваясь при этом от значения текущего времени.

Вот наша новая функция, принимающая объект `movie`.

Мы извлекаем значение текущего времени с помощью JavaScript-объекта `Date`. Не беспокойтесь о деталях — просто знайте, что он возвращает значение текущего времени в миллисекундах.

```
function getNextShowing(movie) {
  var now = new Date().getTime();

  for (var i = 0; i < movie.showtimes.length; i++) {
    var showtime = getTimeFromString(movie.showtimes[i]);
    if ((showtime - now) > 0) {
      return "Next showing of " + movie.title + " is " + movie.showtimes[i];
    }
  }
  return null;
}
```

Теперь мы задействуем массив `showtimes` объекта `movie` и совершаем итерации по нему.

В случае с каждым значением `showtimes` мы извлекаем значение текущего времени в миллисекундах, а затем проводим сравнение.

Если время еще не наступило, то это значение времени следующего сеанса, поэтому оно и возвращается.

Если сеансов больше не остается, мы возвращаем `null`.

```
function getTimeFromString(timeString) {
  var theTime = new Date();
  var time = timeString.match(/(\d+)(?:\s*(\d\d)?\s*(p?))/);
  theTime.setHours( parseInt(time[1]) + (time[3] ? 12 : 0) );
  theTime.setMinutes( parseInt(time[2]) || 0 );
  return theTime.getTime();
}
```



Готово к употреблению

Вот приготовленный нами код, который просто берет строку формата, например `1am` или `3pm`, и преобразует ее в значение времени в миллисекундах.

Не беспокойтесь насчет данного кода; в нем используются регулярные выражения, о которых вы узнаете по ходу изучения JavaScript. А пока просто взгляните на них!

```
var nextShowing = getNextShowing(movie1);
alert(nextShowing);
nextShowing = getNextShowing(movie2);
alert(nextShowing);
```

Здесь мы вызываем функцию `getNextShowing` и используем строку, которую она возвращает, для отображения в диалоговом окне `alert`.

И делаем то же самое в случае с `movie2`.



Как работает «объединение в Цепочку»

Вы обратили внимание на данную строку в предыдущем коде?

```
movie.showtimes.length
```

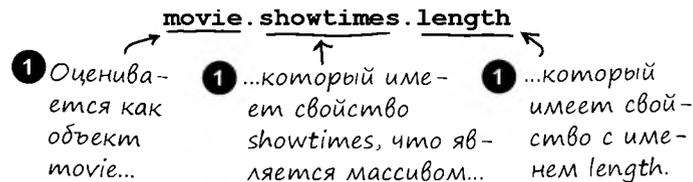
Эта строка не похожа ни на что из того, что нам доводилось видеть раньше. Она представляет собой сокращенный вариант серии шагов, которые нам потребовалось бы выолнить, чтобы извлечь длину массива `showtimes` из объекта `movie`. Вместо нее нам пришлось бы нанисать следующее:

```
var showtimesArray = movie.showtimes;
var len = showtimesArray.length;
```

← Сначала мы извлекаем массив `showtimes`.

← Затем мы используем его для доступа к свойству `length`.

Однако мы можем делать все это за один подход нутем объединения выражений в цепочку. Давайте носмотрим, как это работает:

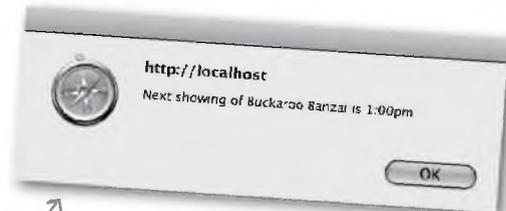


Тест-драйв

Нанечатайте код с предыдущей страницы и нроведите его тест-драйв. Вы увидите, что функция `getNextShowing` принимает объект `movie` и возвращает строку со значением следующего сеанса соответствующего фильма. Вы также можете свободно создавать собственные новые объекты и нроводить тест-драйв с их участием. Мы так и ностунили, при этом местное время было 12:30.

```
var banzaiMovie = {
  title: "Buckaroo Banzai",
  genre: "Cult classic",
  rating: 5,
  showtimes: ["1:00pm", "5:00pm", "7:00pm"]
}
```

```
var nextShowing = getNextShowing(banzaiMovie);
alert(nextShowing);
```



↑ Примечание: качество нашего кода является не совсем таким, как у «производственного кода»; если вы выполните его после наступления времени последнего киносеанса, то получите значение `null`. Повторите попытку на следующий день. 😊

Объекты также могут обладать поведением...

Вы же не думали, что объекты годятся только для сохранения чисел, строк и массивов? Объекты активны — они могут делать определенные вещи. Собаки, к примеру, не сидят на месте: они лают, бегают, играют в мяч, и наш объект `dog` тоже должен вести себя подобным образом! Принимая во внимание все изученное в этой главе, вы полностью готовы к тому, чтобы снабдить свои объекты поведением. Вот как это делается:

```
var fido = {  
  name: "Fido",  
  weight: 40,  
  breed: "Mixed",  
  loves: ["walks", "fetching balls"]  
  bark: function() {  
    alert("Woof woof!");  
  }  
};
```

Мы можем добавить функцию напрямую в наш объект, как показано здесь.

Вместо того чтобы говорить, что это «функция в объекте», мы просто говорим, что это метод. Между ними нет никакой разницы, однако функции в объектах все называют методами.

Обратите внимание, что мы используем анонимную функцию и присваиваем ее свойству `bark` объекта.

Для вызова метода в отношении объекта следует указать имя данного объекта наряду с именем метода, используя точечную нотацию, а также все необходимые аргументы.

```
fido.bark();
```

Мы говорим объекту сделать что-то, вызывая его методы. В данном случае мы вызываем метод `bark` объекта `fido`.

Когда объект
включает
в себя
функцию,
мы говорим,
что данный
объект
включает
в себя метод.



Возвращаемся к приложению Webville Cinema...



Теперь, когда ваши знания об объектах расширились, мы можем вернуться и усовершенствовать код нашего приложения Webville Cinema. Мы уже написали функцию `getNextShowing`, принимающую `movie` в качестве аргумента, однако взамен могли бы сделать эту функцию частью объекта `movie`, превратив ее в метод. Давайте так и поступим:

```
var movie1 = {
  title: "Plan 9 from Outer Space",
  genre: "Cult Classic",
  rating: 5,
  showtimes: ["3:00pm", "7:00pm", "11:00pm"],

  getNextShowing: function(movie) {
    var now = new Date().getTime();

    for (var i = 0; i < movie.showtimes.length; i++) {
      var showtime = getTimeFromString(movie.showtimes[i]);
      if ((showtime - now) > 0) {
        return "Next showing of " + movie.title + " is " + movie.showtimes[i];
      }
    }
    return null;
  }
};
```

Мы взяли наш код и поместили его в метод объекта `movie1` с использованием имени свойства `getNextShowing`.

Но мы знаем, что можем быть не совсем правы...

На самом деле мы не можем просто вбросить функцию в данный объект, поскольку `getNextShowing` принимает `movie` в качестве аргумента. При этом нам хочется вызывать `getNextShowing` следующим образом:

```
var nextShowing = movie1.getNextShowing();
```

Здесь не должны требоваться какие-либо аргументы, поскольку будет вполне ясно, время следующего сеанса какого фильма мы хотим извлечь, то есть того, который представлен объектом `movie1`.

Итак, какие же исправления нам необходимо внести? Нужно удалить параметр `movie` из определения метода `getNextShowing`, однако тогда нам придется что-то сделать со всеми ссылками на `movie.showtimes` в коде, поскольку как только мы удалим параметр, `movie` перестанет существовать как переменная. Что ж, давайте посмотрим...

Давайте уберем параметр movie...

Мы позволим себе удалить параметр `movie` и все ссылки на него. В результате у нас получится следующий код:

```
var movie1 = {
  title: "Plan 9 from Outer Space",
  genre: "Cult Classic",
  rating: 5,
  showtimes: ["3:00pm", "7:00pm", "11:00pm"],

  getNextShowing: function() {
    var now = new Date().getTime();

    for (var i = 0; i < showtimes.length; i++) {
      var showtime = getTimeFromString(showtimes[i]);
      if ((showtime - now) > 0) {
        return "Next showing of " + title + " is " + showtimes[i];
      }
    }
    return null;
  }
};
```

Внизу мы выделили изменения
серым цветом...

Все это выглядит вполне сносно, однако нам еще необходимо продумать, как метод `getNextShowing` будет использовать свойство `showtimes`...

...Нам привычны либо локальные переменные (но `showtimes` к ним не относится), либо глобальные переменные (к которым `showtimes` тоже не относится). Хм...

А вот и еще одно свойство — `title`.

И что теперь?

Итак, вот в чем заключается головоломка: у нас имеются ссылки на свойства `showtimes` и `title`. Обычно в функции мы ссылаемся на локальную переменную, на глобальную переменную или на параметр функции, однако `showtimes` и `title` являются свойствами объекта `movie1`. Впрочем, может все и работает... ведь JavaScript, кажется, достаточно умен для того, чтобы самостоятельно во всем разобраться?

Нет. Не работает. Можете провести тест-драйв, и JavaScript сообщит вам, что переменные `showtimes` и `title` имеют значение `undefined`. Как такое возможно?

Дело вот в чем: эти переменные являются свойствами объекта, однако мы не сказали JavaScript, какого именно объекта. Вы можете подумать: «Ведь очевидно же, что мы имеем в виду ДАННЫЙ объект, вот этот, который находится прямо здесь! Что тут может быть непонятного?». И, да, мы подразумеваем свойства именно этого объекта. В JavaScript присутствует ключевое слово `this`, которое позволяет точно дать понять, что вы имеете в виду *данный конкретный объект*.

На самом деле ситуация немного более сложная, чем она кажется здесь, и об этом мы поговорим совсем скоро, а пока займемся добавлением ключевого слова `this`, чтобы наш код работал как надо.

Добавление ключевого слова this

Давайте добавим `this` в каждое место, где мы указываем свойство, и тем самым дадим понять JavaScript, что имеем в виду свойство *данного* конкретного объекта:

```
var movie1 = {
  title: "Plan 9 from Outer Space",
  genre: "Cult Classic",
  rating: 5,
  showtimes: ["3:00pm", "7:00pm", "11:00pm"],

  getNextShowing: function() {
    var now = new Date().getTime();

    for (var i = 0; i < this.showtimes.length; i++) {
      var showtime = getTimeFromString(this.showtimes[i]);
      if ((showtime - now) > 0) {
        return "Next showing of " + this.title + " is " + this.showtimes[i];
      }
    }
    return null;
  }
};
```

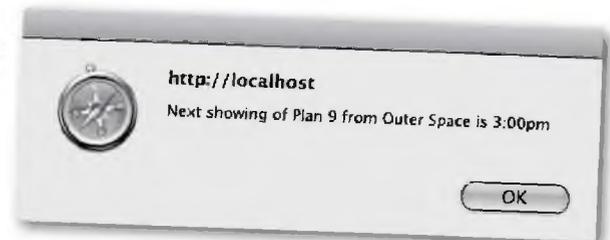
Здесь мы добавили ключевое слово `this` перед каждым свойством, обозначив тем самым, что имеем в виду ссылку на объект `movie1`.

Тест-драйв с участием this



Наберите показанный выше код, а также добавьте функцию `getNextShowing` в свой объект `movie2` (просто скопируйте и вставьте ее). Затем внесите приведенные внизу изменения в своей предыдущий тестовый код. И проведите тест-драйв! Вот что получилось у нас:

```
var nextShowing = movie1.getNextShowing();
alert(nextShowing);
nextShowing = movie2.getNextShowing();
alert(nextShowing);
```



Обратите внимание, что теперь мы вызываем `getNextShowing` в ОТНОШЕНИИ объекта. Так более понятно, не правда ли?



Верно подмечено.

У вас отличная интуиция, если вы догадались, что код дублируется, когда мы копируем `getNextShowing` в более чем один объект `movie`. Одна из целей объектно-ориентированного программирования заключается в максимизации повторного использования кода — здесь мы не используем повторно код, а фактически создаем каждый объект как уникальный, а наши объекты `movie` по соглашению (и из-за копирования и вставки!) в итоге должны получаться одинаковыми. Такой подход не только является излишней тратой ресурсов, но и может создавать благодатную почву для ошибок.

Существует лучший способ сделать это, используя *конструктор*. Что такое конструктор? Это специальная функция, которую мы с вами напишем и которая сможет создавать объекты и делать их все одинаковыми. Можете считать ее своего рода пебольшой фабрикой, которая припимает зачепия свойств, которые вы хотите задать в своем объекте, и возвращает новый объект со всеми пужпыми вам свойствами и методами.

Давайте создадим конструктор...

Как создать конструктор

Давайте создадим конструктор объектов dog. Мы уже знаем, как должны выглядеть наши объекты dog: они будут иметь свойства name, breed и weight, а также включать метод bark. Таким образом, нашему конструктору потребуется принять значения свойств в качестве параметров и вернуть объект dog, включающий метод bark. Вот необходимый нам код:

Функция-конструктор во многом схожа с обычной функцией. Однако по соглашению ее имя должно начинаться с прописной буквы.

Параметры конструктора принимают значения свойств, которыми должен обладать наш объект.

```
function Dog(name, breed, weight) {
```

```
    this.name = name;
```

```
    this.breed = breed;
```

```
    this.weight = weight;
```

```
    this.bark = function() {
```

```
        if (this.weight > 25) {
```

```
            alert(this.name + " says Woof!");
```

```
        } else {
```

```
            alert(this.name + " says Yip!");
```

```
        }
```

```
    };
```

```
}
```

Здесь мы инициализируем свойства объекта значениями, переданными конструктору.

Мы можем включить метод bark в конструируемый объект путем инициализации свойства bark значением функции точно так же, как делали раньше.

Нам необходимо использовать this.weight и this.name в методе для ссылки на свойства в объекте так же, как и раньше.

Имена свойств и имена параметров не обязательно должны быть одинаковыми, однако зачастую они оказываются таковыми — опять-таки, по соглашению.

Обратите внимание, насколько данный синтаксис отличается от синтаксиса объекта. Это операторы, поэтому каждый из них должен заканчиваться точкой с запятой (как это обычно бывает в функции).

Пройдемся по коду еще раз, чтобы убедиться в том, что вы во всем разобрались. Dog — это функция-конструктор, принимающая набор аргументов, которые являются начальными значениями необходимых нам свойств: name, breed и weight. Получив эти значения, конструктор присваивает свойства, используя ключевое слово this. Он также определяет наш метод bark. И каков же результат всего этого? Конструктор Dog возвращает новый объект. Давайте посмотрим, как фактически использовать конструктор.

Вам не нужно беспокоиться о создании всех этих объектов, поскольку мы сделаем это за вас.



Воспользуемся нашим конструктором

Теперь, когда мы построили нашу «фабрику», можно начинать использовать ее для создания объектов `dog`. Вам потребуется вызывать функцию-конструктор особым образом — путем размещения ключевого слова `new` перед вызовом. Вот ряд примеров:

Для создания объекта `dog` мы используем ключевое слово `new` в сочетании с конструктором.

А затем вызываем его подобно любой другой функции.

```
var fido = new Dog("Fido", "Mixed", 38);
var tiny = new Dog("Tiny", "Chawalla", 8);
var clifford = new Dog("Clifford", "Bloodhound", 65);
```

```
fido.bark();
tiny.bark();
clifford.bark();
```

Получив объекты, мы можем вызывать их методы `bark` для возврата соответствующих значений.

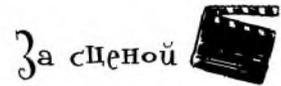
Мы создаем три разных объекта `dog` путем передачи разных аргументов для конфигурирования каждого из этих объектов.

Давайте еще раз разберемся, что здесь происходит: мы создаем три разных объекта `dog`, каждый из которых будет обладать своими свойствами, для чего используем ключевое слово `new` в сочетании с ранее созданным конструктором `Dog`. Конструктор возвращает объект `dog`, сконфигурированный в соответствии с переданными параметрами.

Далее мы вызываем метод `bark` в отношении каждого объекта `dog`. Обратите внимание, что мы используем один и тот же метод `bark` в случае со всеми объектами `dog`, а при каждом вызове `bark` ключевое слово `this` будет указывать на объект `dog`, в отношении которого был совершен вызов. Таким образом, если мы вызовем метод `bark` в отношении `fido`, то в методе `bark` ключевое слово `this` будет указывать на объект `fido`. Давайте более пристально взглянем на то, как все это работает.



Как на самом деле работаем this?



Всякий раз, когда мы помещаем ключевое слово `this` в код метода, оно будет интерпретироваться как ссылка на объект, в отношении которого был вызван данный метод. Таким образом, если мы вызовем `fido.bark`, то `this` будет указывать на `fido`. Либо, если мы вызовем его в отношении объекта `tiny`, то `this` будет указывать на `tiny` в вызове метода. Но откуда `this` знает, какой объект оно представляет? Давайте посмотрим.

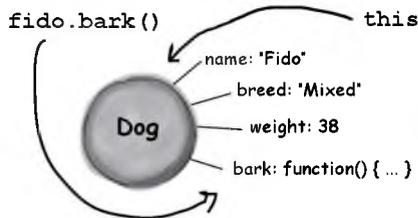
1 Допустим, у нас имеется объект `dog`, присвоенный `fido`:

```
fido = new Dog("Fido", "Mixed", 38);
```



Вот экземпляр нашего нового объекта `dog` с нужными значениями свойств.

2 Теперь мы вызываем `bark()` в отношении `fido`:



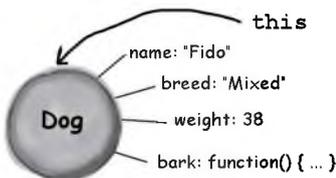
При каждом вызове метода в отношении объекта JavaScript делает так, чтобы ключевое слово `this` указывало на сам этот объект. Таким образом, здесь `this` указывает на `fido`.

Так что когда мы ссылаемся на `this.name`, мы знаем, что имя объекта будет "Fido".

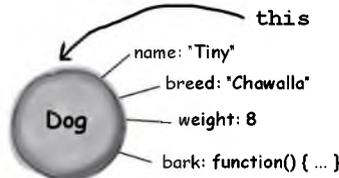
3 Таким образом, `this` всегда будет указывать на объект, в отношении которого был вызван метод, независимо от того, как много объектов `dog` мы создадим:

Вы можете вызывать `bark` в отношении любого объекта `dog`, а `this` при этом будет присваиваться конкретному `dog` до выполнения вашего основного кода.

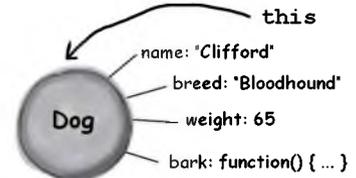
```
fido.bark()
```



```
tiny.bark()
```



```
clifford.bark()
```



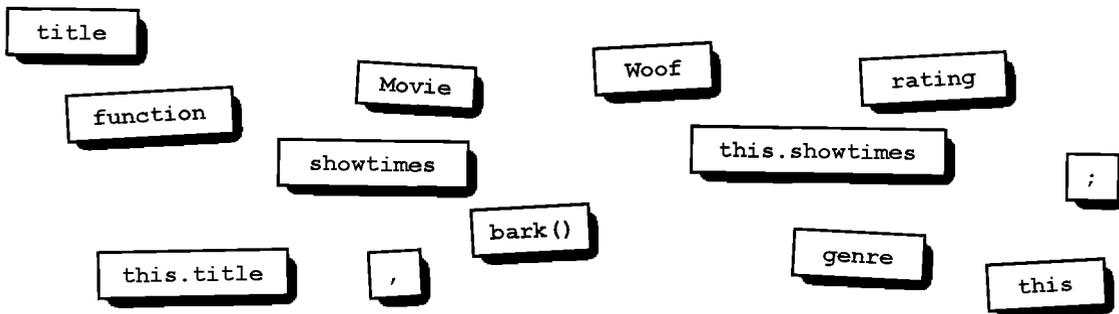


Развлечения с Магнитами

Таблички с рабочим кодом функции-конструктора Movie были прикреплены к холодильнику, однако некоторые из них упали на пол. Можете ли вы восстановить целостность данного кода? Будьте внимательны, поскольку на полу уже лежало несколько лишних табличек, которые могут сбить вас с толку.

```
function _____(_____, _____, rating, showtimes) {
    this.title = _____;
    this.genre = genre;
    this._____ = rating;
    this.showtimes = _____;
    this.getNextShowing = function() {
        var now = new Date().getTime();
        for (var i = 0; i < _____.length; i++) {
            var showtime = getTimeFromString(this._____ [i]);
            if ((showtime - now) > 0) {
                return "Next showing of " + _____ + " is " + this.showtimes[i];
            }
        }
    } _____
}
```

Используйте эти таблички, чтобы заполнить пробелы в коде.



Часть Задаваемые Вопросы

В: В чем заключается истинная разница между функцией и методом? В конце концов, если они являются одним и тем же, почему называются по-разному?

О: По соглашению, если объект включает функцию, мы называем ее *методом*. Функция и метод работают одинаково, за исключением того, что вызов метода объекта осуществляется с применением оператора «точка», при этом метод может использовать `this` для доступа к объекту, в отношении которого был вызван. Считайте функцию отдельным блоком кода, который можно вызывать, а метод — поведением, привязанным к определенному объекту.

В: Если я с помощью конструктора создам объекты, включающие метод, то в случае со всеми этими объектами будет совместно использоваться один и тот же код данного метода?

О: Да, все верно, причем в этом заключается одно из преимуществ объектно-ориентированного программирования: вы можете создать код для класса объектов (например, для всех своих объектов `dog`) в одном месте, и он будет совместно использоваться в случае со всеми объектами `dog`. Чтобы сделать его специфичным для каждого из объектов `dog`, необходимо обратиться к их свойствам, для доступа к которым вам потребуется использовать `this`.

В: Могу ли я задавать для `this` значения по своему выбору, и если да, не приведет ли это к тем или иным отрицательным последствиям?

О: Нет, вы не сможете задать для `this` какие-либо значения. Помните, что `this` — это ключевое слово, а не переменная! Оно выглядит и ведет себя отчасти как переменная, но не является ею.

В: Есть ли у `this` значение вне метода объекта?

О: Нет, если вы не вызываете метод объекта, то `this` будет `undefined`.

В: Насколько я понимаю, когда я вызываю метод в отношении объекта, этот объект задается как значение `this` на все время, пока будет идти оценка метода. Так ли это?

О: Да, в теле объекта `this` всегда будет указывать на сам объект. Но в некоторых особых ситуациях это может быть не так; например, все окажется несколько сложнее, когда объекты будут находиться внутри объектов, и если вы решите предпринять данное действие, то вам придется принимать во внимание семантику, поскольку таково общее правило.

В: Мне доводилось слышать о том, что при объектно-ориентированном программировании у меня могут иметься классы объектов, которые способны наследовать свойства и методы друг от друга по цепочке. Например, у меня мог бы быть класс `mammals`, от которого наследуют свойства и методы `dog` и `cat`. Возможно ли это на JavaScript?

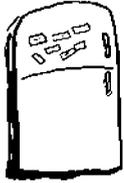
О: Да, возможно. В случае с JavaScript используется так называемое *прототипное наследование*, которое представляет собой даже более мощную модель, чем модели, основанные строго на классах. Рассмотрение прототипного наследования немного выходит за рамки данной книги, хотя, может, нам и стоило бы написать больше о нем в сфере JavaScript.

В: Когда мы указываем `new Date()`, то задействуем конструктор, не так ли?

О: Да, абсолютно верно! `Date` — это встроенный JavaScript-конструктор. Когда вы указываете `new Date()` в коде, то получаете в свое распоряжение объект `Date` с набором полезных методов, которые можно использовать для работы с датами.

В: В чем разница между объектами, которые мы пишем сами и которые создаем с помощью конструктора?

О: Основное различие заключается в том, как вы их создаете. Объекты, которые вы пишете сами, используя фигурные скобки и разделенные запятыми свойства, называются *литералами объектов*. Вы символ за символом вносите их в свой код! Если вам потребуется еще один такой объект, то придется самим написать его и позаботиться о том, чтобы он располагал аналогичными свойствами. Объекты, генерируемые конструктором, создаются с использованием `new` и функции-конструктора, возвращающей объект. Вы можете использовать функцию-конструктор для создания множества объектов с одинаковыми свойствами, но с разными значениями свойств, если пожелаете.



Развлечения с магнитами. Решение

Таблички с рабочим кодом функции-конструктора `Movie` были прикреплены к холодильнику, однако некоторые из них упали на пол. Можете ли вы восстановить целостность данного кода? Будьте внимательны, поскольку на полу уже лежало несколько лишних табличек, которые могут сбить вас с толку. Приведем решение данного задания.

```
function Movie ( title , genre , rating , showtimes ) {
  this.title = title ;
  this.genre = genre ;
  this.rating = rating ;
  this.showtimes = showtimes ;
  this.getNextShowing = function () {
    var now = new Date().getTime();
    for (var i = 0; i < this.showtimes.length; i++) {
      var showtime = getTimeFromString(this.showtimes[i]);
      if ((showtime - now) > 0) {
        return "Next showing of " + this.title + " is " + this.showtimes[i];
      }
    }
  }
}
```

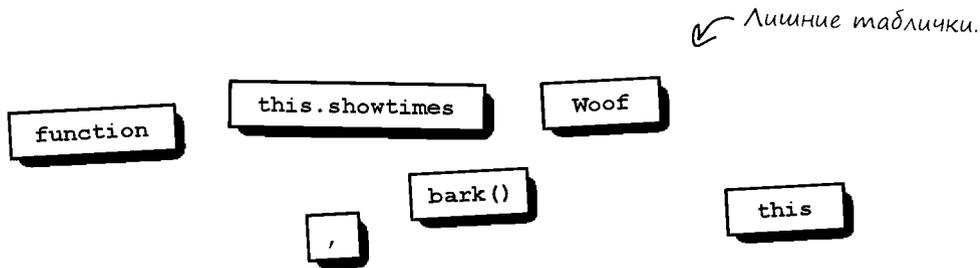
Этот конструктор, в качестве имени которого мы используем `Movie`.

Мы передаем значения свойств, которые хотим сконфигурировать: `title`, `genre`, `rating` и `showtimes`...

... и инициализируем эти свойства.

Для ссылки на свойства в объекте мы используем ключевое слово `this`.

Не забудьте поставить в конце данного оператора точку с запятой!



Сразу же проведем тест-драйв нашего конструктора



Теперь, когда у нас есть конструктор `Movie`, пора заняться созданием объектов `movie`! Напечатав код функции-конструктора `Movie`, добавьте приведенные ниже строки и проведите тест-драйв данного конструктора. Мы полагаем, вы согласитесь с тем, что это намного более легкий способ создания объектов.

Обратите внимание, значение массива для `showtimes` можно поместить прямо в вызов функции.

```
var banzaiMovie = new Movie("Buckaroo Banzai",
                            "Cult Classic",
                            5,
                            ["1:00pm", "5:00pm", "7:00pm", "11:00pm"]);
```

Сначала мы создаем объект `movie` для фильма `Buckaroo Banzai` (один из наших любимых представителей жанра культовой классики), а также передаем значения для параметров.

```
var plan9Movie = new Movie("Plan 9 from Outer Space",
                            "Cult Classic",
                            2,
                            ["3:00pm", "7:00pm", "11:00pm"]);
```

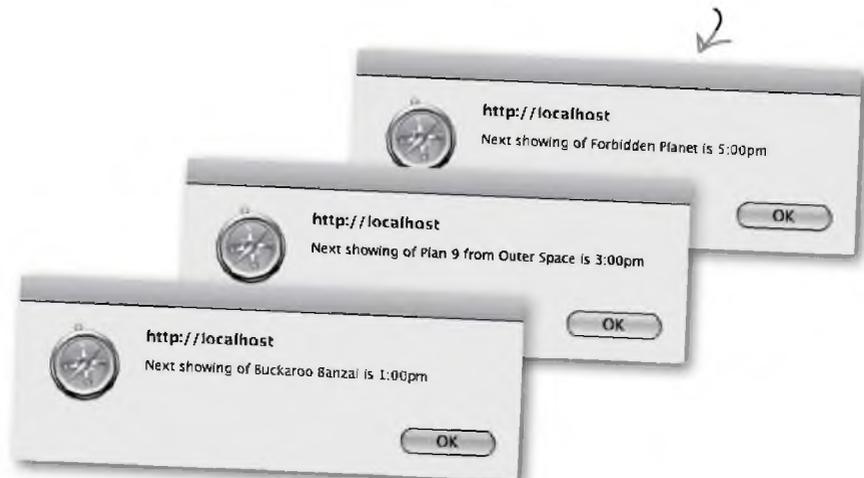
Далее следует `Plan 9 from Outer Space...`

```
var forbiddenPlanetMovie = new Movie("Forbidden Planet",
                                       "Classic Sci-fi",
                                       5,
                                       ["5:00pm", "9:00pm"]);
```

...и, конечно же, `Forbidden Planet.`

```
alert(banzaiMovie.getNextShowing());
alert(plan9Movie.getNextShowing());
alert(forbiddenPlanetMovie.getNextShowing());
```

Создав все необходимые объекты, мы можем вызывать метод `getNextShowing` и выводить для пользователя в диалоговых окнах `alert` сообщения о времени следующего сеанса соответствующего фильма.





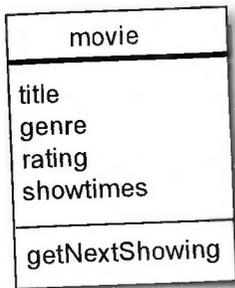
Поздравляем! Вы справились с изучением функций и объектов! Теперь, когда вы все знаете о них, и прежде, чем мы завершим эту главу, потратим немного времени и взглянем на JavaScript-объекты в «дикой природе», то есть в их родной среде — в браузере!

Вы, возможно, уже стали замечать...

...что объекты буквально окружают вас. К примеру, `document` и `window` — это объекты, равно как и элементы, возвращаемые посредством `document.getElementById`. Но это лишь часть того множества объектов, с которыми мы будем сталкиваться в дальнейшем, — когда дойдем до API-интерфейсов HTML5, вы увидите, что объекты встречаются на каждом шагу!

Давайте еще раз взглянем на некоторые объекты, которые уже использовали ранее в книге:

А вот наш объект `movie`.

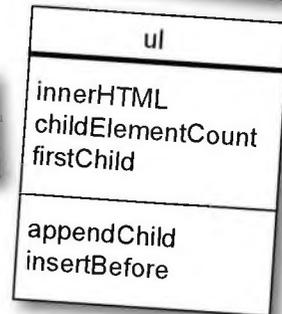
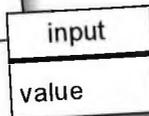
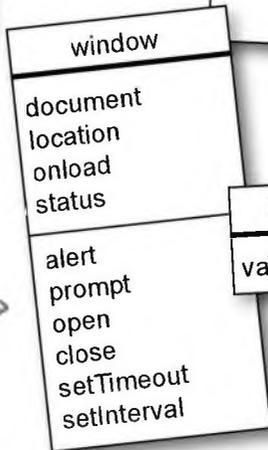
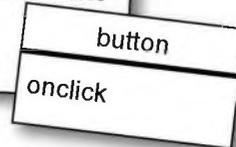
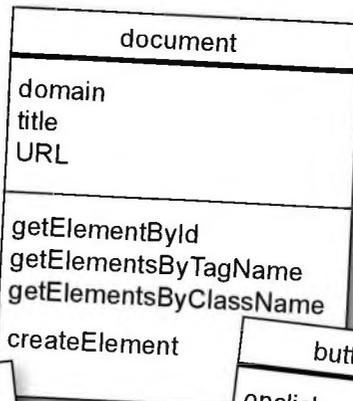


Объекты, с которыми мы уже сталкивались.



Мы изобразили объекты следующим образом: сверху приводятся свойства...

...а внизу — методы, благодаря чему вы сразу можете увидеть сводку по каждому объекту со всеми его свойствами и методами.





Что такое объект window?

Когда вы будете писать код, выполняемый в браузере, объект window станет частью вашей жизни. Данный объект представляет собой глобальную среду для JavaScript-программ, а также главное окно приложения и как таковой содержит множество важных свойств и методов. Давайте взглянем на него.

Вот наш объект window вместе с несколькими примечательными свойствами и методами, о которых вам нужно знать. Помимо них, есть еще множество других...



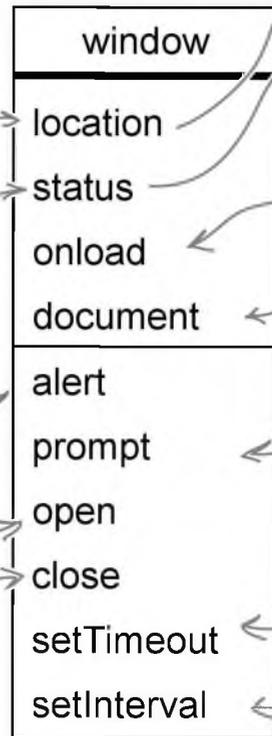
location содержит URL-адрес страницы. Если изменить его, то браузер извлечет новый URL-адрес!

status содержит строку, которая отображается в статусной строке браузера.

Вы уже знакомы с методом alert, который выводит соответствующее диалоговое окно.

Открывает новое окно браузера.

Закрывает окно.



Вы, несомненно, уже сталкивались с ним ранее: onload — это свойство, содержащее функцию, которая вызывается после полной загрузки страницы.

Свойство document содержит объектную модель документа (DOM)!

prompt подобен alert, за тем исключением, что получает информацию от пользователя.

Вызывает обработчик по истечении заданного интервала времени.

Многократно вызывает обработчик через заданный интервал времени.



Мы все время писали alert, а не window.alert... А откуда браузер будет знать, что мы имеем в виду метод alert объекта window?

Объект window является глобальным.

Это может показаться немного странным, но объект window действует как глобальная среда, поэтому любые имена свойств или методов из данного объекта разрешаются даже в том случае, если вы не указали перед ними слово window.

Кроме того, любые глобальные переменные, которые вы определяете, также помещаются в пространство имен window, так что вы сможете ссылаться на них следующим образом: window.имя_переменной.

Более пристальный взгляд на window.onload

По ходу книги мы с вами часто использовали обработчик событий window.onload. Путем присваивания функции свойству window.onload мы сможем гарантировать, что наш код не будет выполняться до тех пор, пока загрузка страницы не закончится и объектная модель документа (DOM) не будет полностью сгенерирована. В операторе window.onload много чего происходит, поэтому давайте еще раз взглянем на него, чтобы вы четко во всем разобрались.

Вот наш глобальный объект window.

onload является свойством объекта window.

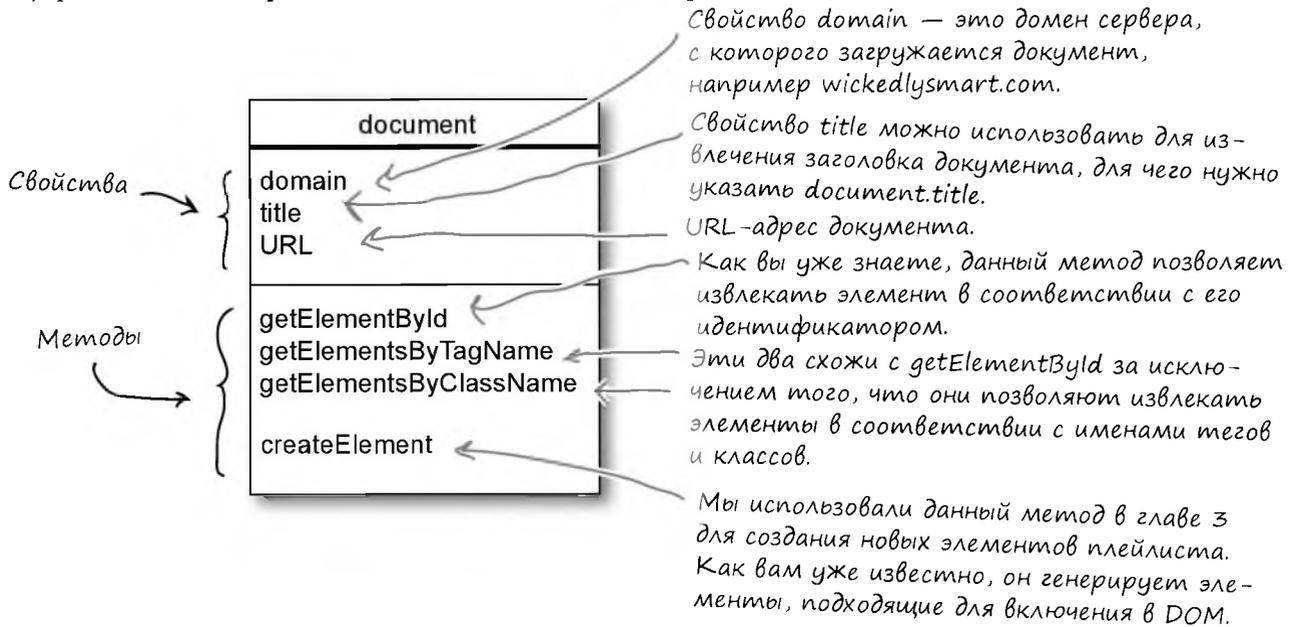
Это анонимная функция, которая присваивается свойству onload.

```
→ window.onload = function() {  
    // code here ←  
};
```

И, конечно же, тело функции будет выполняться, как только window полностью загрузит страницу и вызовет нашу анонимную функцию!

Еще один взгляд на объект document

Объект `document` также уже вам знаком — мы использовали его для доступа к объектной модели документа (DOM). Он является свойством объекта `window`. Мы, конечно, не указывали его как `window.document`, поскольку в этом не было необходимости. Давайте заглянем внутрь него и посмотрим, какие еще там имеются интересные свойства и методы:



Более пристальный взгляд на `document.getElementById`

В начале главы мы обещали, что к ее концу вы поймете, что такое `document.getElementById`. Что ж, вы справились с изучением функций, объектов и методов и теперь готовы к этому! Взгляните на следующее:

Это объект `document`, который является встроенным JavaScript-объектом, обеспечивающим доступ к объектной модели документа (DOM).

```
var div = document.getElementById("myDiv");
```

↑ Это метод, который...
 ↑ ...принимает один аргумент в виде значения `id` элемента `<div>` и возвращает объект элемента.



То, что ранее выглядело как приводящая в замешательство строка синтаксиса, теперь кажется намного более понятным, не так ли? Переменная `div` также является объектом — объектом элемента. Давайте более пристально взглянем и на него.

Еще один объект, о котором нужно знать: объект элемента

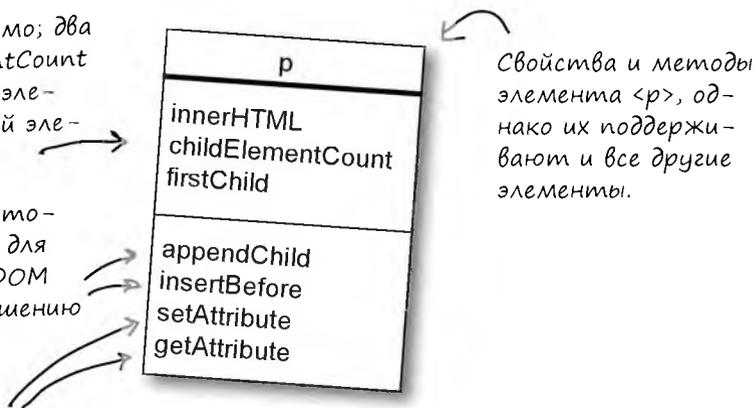
Не забывайте, что при работе с методами вроде `getElementById` элементы, которые они возвращают, также являются объектами! Ладно, вы могли и не осознавать этого, но теперь, с учетом того, что вам уже известно, вы, возможно, начали считать, что все в JavaScript является объектами (что, впрочем, в большой степени верно).

Вы уже знакомы с некоторыми свойствами элементов, например со свойством `innerHTML`; давайте взглянем на ряд более примечательных свойств и методов.

Свойство `innerHTML` уже вам знакомо; два других свойства — это `childElementCount` (количество дочерних элементов у элемента) и `firstChild` (первый дочерний элемент, если таковой имеется).

Вы можете использовать методы `appendChild` и `insertBefore` для вставки новых элементов в DOM в качестве дочерних по отношению к определенному элементу.

Мы будем использовать `setAttribute` и `getAttribute` для задания и извлечения атрибутов, таких как `src`, `class` и `id`, в случае с элементами.



Часть Задаваемые Вопросы

В: Поскольку `window` является глобальным объектом, означает ли это, что я могу использовать его свойства и методы, не указывая перед ними слово `window`?

О: Все верно. Вам решать, указывать ли слово `window` перед именем свойства или метода объекта `window`. В случае, например, с `alert` все знают, что это за метод, и никто не указывает перед ним `window`. С другой стороны, если вы используете менее известные свойства или методы, то можете захотеть сделать свой код более легким для понимания, и в таком случае вам потребуется указывать перед их именами слово `window`.

В: Технически, я могу написать `onload = init` вместо `window.onload = init`, не так ли?

О: Да, правильно. Однако мы не рекомендуем так поступать в данном конкретном случае, поскольку существует масса других объектов, имеющих свойство `onload`, в силу чего ваш код будет намного яснее, если вы укажете `window.` перед `onload`.

В: Причина, по которой мы не пишем `window.onload = init()`, в том, что в результате этого произошел бы вызов данной функции, а не присваивание ее значения свойству `onload`?

О: Все верно. Когда вы указываете круглые скобки после имени функции, например `init()`, то этим вы говорите, что хотите *вызвать* функцию `init`. Если же вы укажете ее имя без круглых скобок, то под этим будет подразумеваться, что вы присваиваете значение функции свойству `onload`. Это тонкий момент, который легко упустить из виду при создании кода, однако он влечет за собой весомые последствия, так что будьте очень внимательны.

В: Какой из двух способов создания обработчика `window.onload` лучше: с указанием имени функции или посредством использования анонимной функции?

О: Один не является лучше другого, поскольку при использовании любого из этих способов вы, по сути, будете делать одно и то же — задавать значение `window.onload` в виде функции, которая станет выполняться после полной загрузки страницы. Если вам по какой-то причине потребуется вызывать `init` из другой функции позже в своей программе, то придется определить функцию `init`. В противном случае не будет важно, какой способ вы предпочтете.

В: В чем разница между встроенными объектами вроде `window` и `document` и теми, которые создаем мы?

О: Первое различие заключается в том, что встроенные объекты отвечают принципам, определяемым спецификациями, и вы можете обратиться к спецификациям W3C, чтобы разобраться во всех их свойствах и методах. Во-вторых, многие встроенные объекты (например, `String`) обладают свойствами, которые не могут быть изменены. Кроме того, объекты есть объекты. Замечательная вещь, касающаяся встроенных объектов, состоит в том, что они уже созданы и готовы к использованию вами.

↑ Да, `String` — это объект! Загляните в хороший справочник по JavaScript, чтобы узнать все подробности о его свойствах и методах.

Поздравляем! Вы завершили свое путешествие по объектам и преодолели несколько глав учебного курса по JavaScript. Настало время применить эти знания и заняться программированием с использованием HTML5 и всех новых API-интерфейсов JavaScript начиная со следующей главы!

Вы завершаете эту главу, зная об объектах и функциях больше, чем многие другие люди. Естественно, вы всегда можете научиться чему-то еще, и мы призываем вас именно так и поступить (после того, как закончите читать данную книгу)!

Передохните немного, завершив эту главу, и прежде, чем двинуться дальше, ознакомьтесь с секцией «Ключевые моменты» и решите кроссворд для закрепления изученного материала.



КЛЮЧЕВЫЕ МОМЕНТЫ

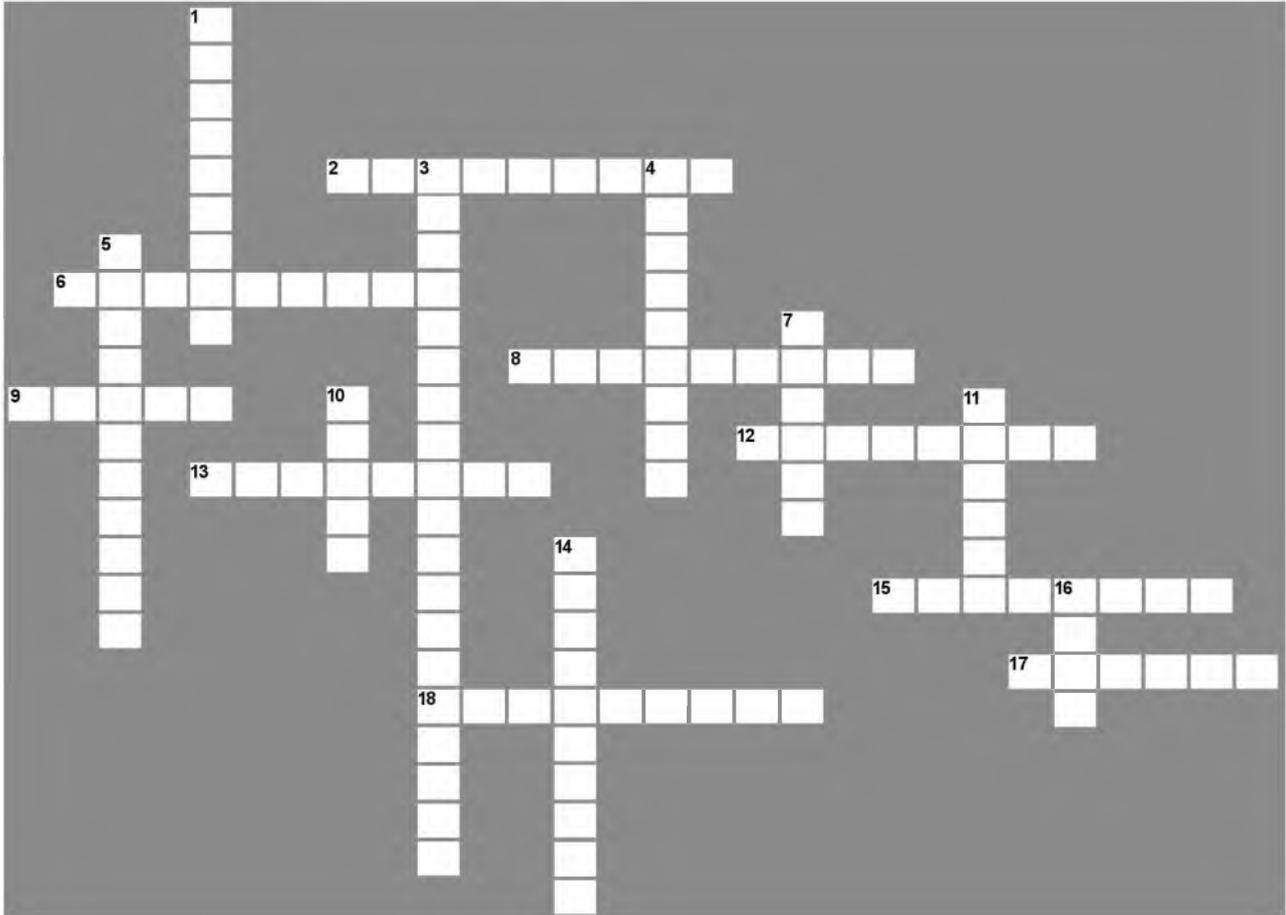


- Для создания функции необходимо использовать ключевое слово `function` в сочетании с круглыми скобками, в которые будут заключаться параметры при наличии таковых.
- Функции могут быть именованными либо анонимными.
- Правила присваивания имен функциям являются теми же самыми, как и в случае с переменными.
- Тело функции заключается в фигурные скобки и содержит операторы, выполняющие работу функции.
- Функция может возвращать значение посредством оператора `return`.
- Для вызова функции необходимо указать ее имя и передать все необходимые аргументы.
- В JavaScript используется передача параметров по значению.
- Когда вы передаете объект (например, `dog`) в качестве аргумента функции, соответствующий параметр получает копию **ссылки** на данный объект.
- Переменные, определяемые внутри функций, включая параметры, называются *локальными*.
- Переменные, определяемые вне функций, называются *глобальными*.
- Локальные переменные невидимы вне функции, в которой они определены. Это называется *областью видимости переменной*.
- Если объявить локальную переменную с тем же именем, что и у глобальной переменной, то глобальная переменная окажется «в тени» локальной переменной.
- При указании ссылок на множественные JavaScript-файлы на странице все глобальные переменные должны определяться в одинаковом глобальном пространстве.
- Если присвоить новую переменную, не используя ключевое слово `var`, эта переменная будет глобальной, даже если вы впервые присваиваете ее функции.
- Функции — это значения, которые могут присваиваться переменным, передаваться другим функциям, сохраняться в массивах и присваиваться свойствам объектов.
- Объекты — это коллекции свойств.
- Обращаться к свойствам объектов можно с использованием точечной или скобочной нотации.
- При использовании скобочной нотации имя свойства следует заключать в кавычки, как строку, например: `myObject["имя"]`.
- Вы можете изменять значения свойств, удалять свойства и добавлять новые свойства в объекты.
- Вы можете перечислять свойства объектов, используя цикл `for-in`.
- Функция, присвоенная свойству объекта, называется *методом*.
- Метод может использовать специальное ключевое слово `this` для ссылки на объект, в отношении которого он был вызван.
- Конструктор — это функция, создающая объекты.
- Работа конструктора заключается в создании нового объекта и инициализации его свойств.
- Для вызова конструктора с целью создания объекта необходимо использовать ключевое слово `new`. Например, `new Dog()`.
- По ходу книги мы с вами уже использовали несколько объектов, включая `document`, `window`, а также различные объекты элементов.
- Метод `document.getElementById` возвращает объект элемента.



HTML5-Кроссворд

Это была ураганная глава о функциях, объектах, свойствах и методах, так что вам много чего необходимо закрепить в памяти. Вот кроссворд к главе 4, который вам нужно решить.



По горизонтали

2. Функция без имени.
6. Эти переменные доступны только внутри функций.
8. Функции без операторов `return` возвращают это.
9. Функция в объекте.
12. Объект _____ представляет собой объектную модель документа (DOM).
13. Аргументы передаются по _____.
15. Данное ключевое слово необходимо использовать в начале определения функции.
17. Настоящий глобальный объект.
18. То, что указывается в объявлении функции.

По вертикали

1. По соглашению, имена конструкторов должны начинаться с _____ буквы.
3. Связывание свойств и вызовов функций посредством оператора «точка».
4. То, что указывается в вызове функции.
5. Функция данного рода создает объекты.
7. Свойство в `window`, которое мы присваиваем функции обработчика.
10. Оператор «_____» позволяет получить доступ к свойствам и методам объекта.
11. Функции могут включать, а могут и не включать данный оператор.
14. Область видимости переменной, которая доступна повсеместно.
16. Указывает на текущий объект в методе объекта.



Возьми в руку карандаш

Решение

Воспользуйтесь своими знаниями о функциях и передаче аргументов параметрам для оценки приведенного ниже кода. Сделав это, напишите внизу, какое значение будет иметь каждая из переменных. Вот наше решение этого задания.

```
function dogsAge(age) {
    return age * 7;
}

var myDogsAge = dogsAge(4);

function rectangleArea(width, height) {
    var area = width * height;
    return area;
}

var rectArea = rectangleArea(3, 4);

function addUp(numArray) {
    var total = 0;
    for (var i = 0; i < numArray.length; i++) {
        total += numArray[i];
    }
    return total;
}

var theTotal = addUp([1, 5, 3, 9]);

function getAvatar(points) {
    var avatar;
    if (points < 100) {
        avatar = "Mouse";
    } else if (points > 100 && points < 1000) {
        avatar = "Cat";
    } else {
        avatar = "Ape";
    }
    return avatar;
}

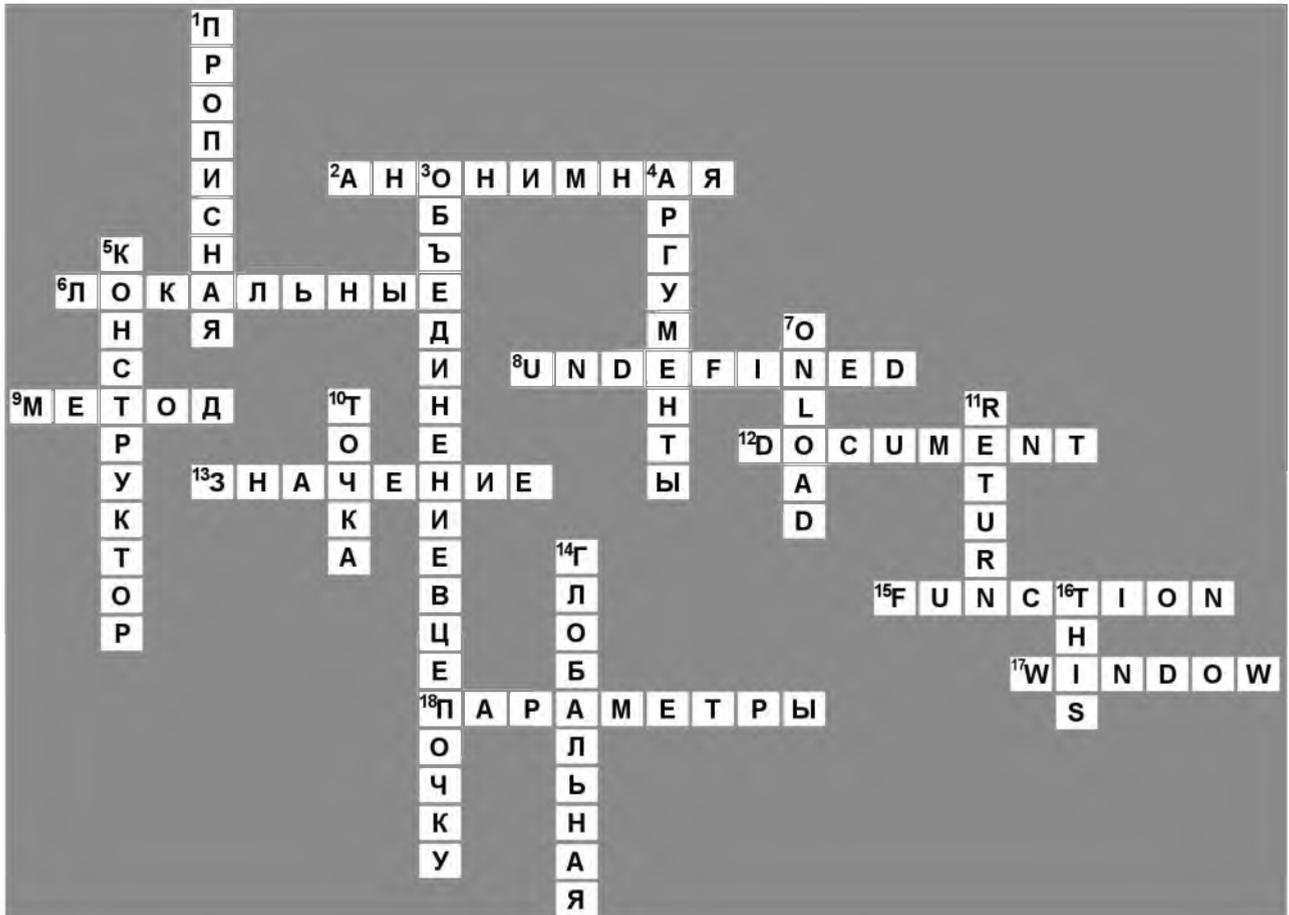
var myAvatar = getAvatar(335);
```

Напишите
здесь, какое
значение будет
иметь каждая
из переменных.

myDogsAge =28.....
 rectArea =12.....
 theTotal =18.....
 myAvatar =Cat.....



HTML5-Кроссворд. Решение



5 создание HTML-страниц с поддержкой определения Местоположения

API-интерфейс Geolocation

Разве не удивительно то, как эти новые технологии сближают всех нас?



Куда бы вы ни отправились, вас можно найти. Порой знание того, где вы находитесь, имеет существенное значение (особенно для веб-приложений). Из этой главы вы узнаете, как создавать веб-страницы с **поддержкой определения местоположения**, — иногда вы сможете определять местонахождение своих пользователей вплоть до угла, на котором они стоят, а иногда вам будет удаваться выяснить лишь район города, в котором они находятся. Время от времени вы вообще не сможете получить хоть какую-то информацию о местоположении пользователей в силу технических причин или просто потому, что им не нравится ваше чрезмерное любопытство. Да, представьте себе! Так или иначе, в этой главе мы рассмотрим API-интерфейс JavaScript под названием Geolocation. Возьмите свое лучшее устройство с поддержкой определения местоположения (даже если это будет настольный компьютер), и давайте приступим к работе.

Ваши пользователи путешествуют с мобильными устройствами, поддерживающими определение местоположения. Наилучшими будут приложения, которые повышают качество пользовательского взаимодействия путем использования данных об их местонахождении.



Местоположение, местоположение...

Знание того, где находятся ваши пользователи, дает возможность обеспечить для них более качественное взаимодействие: вы можете указывать им направление; советовать, куда бы они могли отправиться; дать им знать о том, кто еще, находящийся поблизости, интересуется, например, теми же мероприятиями, что и они. Существует бесчисленное множество путей использования информации о местоположении.

С помощью HTML5 (и API-интерфейса Geolocation на основе JavaScript) вы можете легко получать доступ к информации о местоположении на своих страницах. Однако есть вещи, о которых вам необходимо знать, прежде чем мы приступим к работе. Давайте рассмотрим их.

Часто задаваемые вопросы

В: Я слышал, что Geolocation не является настоящим API-интерфейсом. Это так?

О: Geolocation не считается полноценным членом семейства существующего стандарта HTML5, однако следует отметить, что он является стандартом W3C, который широко поддерживается, и многие относят Geolocation к числу важных API-интерфейсов в HTML5. И его наверняка можно назвать **настоящим** API-интерфейсом JavaScript!

В: API-интерфейс Geolocation и API-интерфейс Google Maps — это не одно и то же?

О: Нет. Это абсолютно разные API-интерфейсы. Geolocation сосредоточен исключительно на получении информации о местоположении человека на поверхности Земли. API-интерфейс Google Maps — это JavaScript-библиотека от компании Google, которая обеспечивает доступ ко всей функциональности Google Maps. Если вам необходимо отображать местоположение своих пользователей на карте, то API-интерфейс от Google станет удобным инструментом.

В: Если мое устройство раскрывает мое местоположение, разве это не вторжение в частную жизнь?

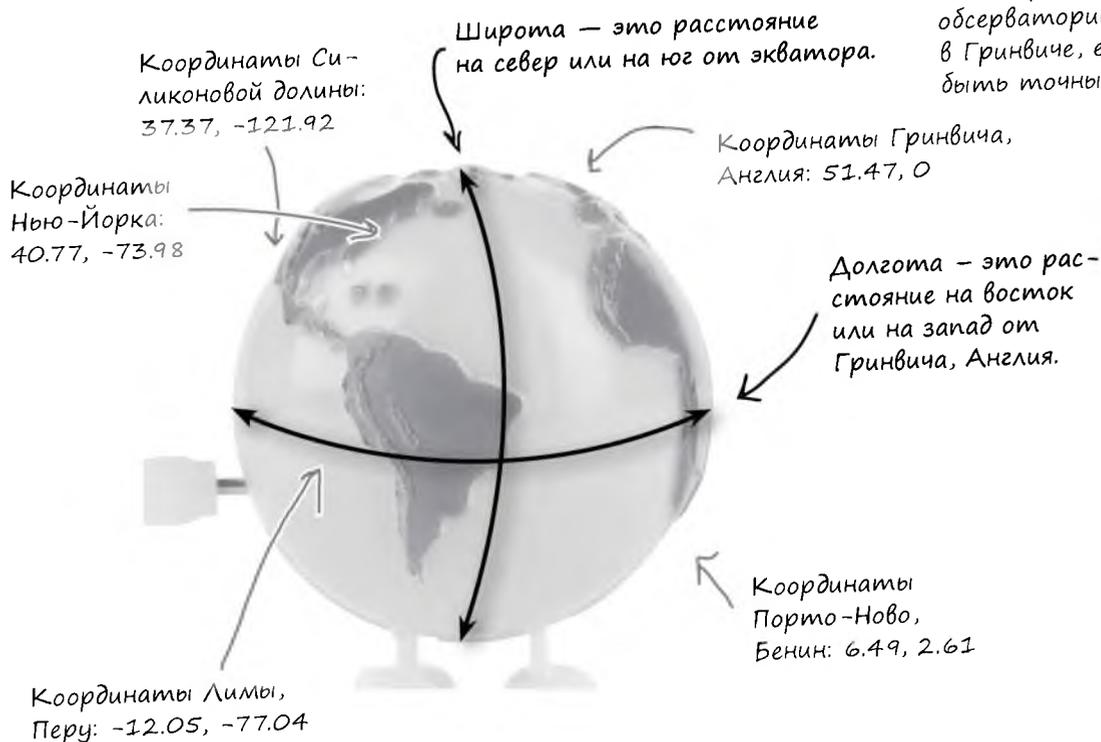
О: Спецификации API-интерфейса Geolocation определяют, что любой браузер должен получить от пользователя специальное разрешение на использование данных о его местоположении. Таким образом, если ваш код задействует API-интерфейс Geolocation, то первое, что сделает браузер, — это удостоверится в том, что пользователь согласен раскрыть информацию о своем местонахождении.

В: Насколько хорошо поддерживается API-интерфейс Geolocation?

О: Очень хорошо. Фактически, он доступен почти в любом современном браузере, включая настольный и мобильный сегменты. Только необходимо убедиться, что вы используете последнюю версию браузера. Если это так, скорее всего, никаких проблем с поддержкой API-интерфейса Geolocation у вас не возникнет.

Широта и долгота...

Чтобы узнать, где вы находитесь, потребуется система координат на поверхности Земли. К счастью, у нас имеется такая штука, которая задействует широту и долготу в качестве системы координат. Широта определяет северную/южную точку на поверхности Земли, а долгота — восточную/западную точку. Широта измеряется от экватора, а измерение долготы ведется от Гринвича, Англия. Задача API-интерфейса Geolocation заключается в том, чтобы дать нам координаты нашего местоположения в любой момент времени, используя следующие координаты:



Подробнее о широте/долготе

Вам, вероятно, доводилось видеть, что широта и долгота указываются как в градусах/минутах/секундах, например (47°38'34", 122°32'32"), так и в десятичных значениях, например (47.64, -122.54). В случае с API-интерфейсом Geolocation мы всегда будем использовать десятичные значения. Если вам потребуется преобразовать градусы/минуты/секунды в десятичные значения, это можно будет сделать с помощью следующей функции:

```
function degreesToDecimal(degrees, minutes, seconds) {
    return degrees + (minutes / 60.0) + (seconds / 3600.0);
}
```

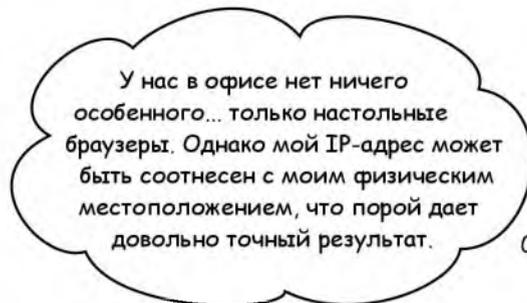
Следует отметить, что западная долгота и южная широта представляются отрицательными значениями.

Как API-интерфейс Geolocation определяет местоположение пользователя

Вам необязательно иметь новейший смартфон, чтобы определять свое местоположение. Это позволяют делать даже настольные браузеры. У вас может возникнуть вопрос о том, как настольный браузер может определить ваше местоположение, если у него нет GPS или другой специальной технологии, дающей возможность сделать это. Что ж, все браузеры (установленные в операционных системах мобильных устройств и настольных компьютеров) используют несколько способов определения местоположения, причем одни дают более точный результат, чем другие. Давайте взглянем на них.



Я выиграла новейший смартфон со встроенной поддержкой GPS и теперь могу с высокой точностью определять свои координаты!



У нас в офисе нет ничего особенного... только настольные браузеры. Однако мой IP-адрес может быть соотнесен с моим физическим местоположением, что порой дает довольно точный результат.

GPS

Глобальная система определения координат (Global Positioning System), поддерживаемая многими современными мобильными устройствами, позволяет очень точно определять местоположение благодаря спутникам. Информация о местоположении может включать данные о высоте, скорости и направлении. Однако для того, чтобы пользоваться этой системой, вашему устройству необходимо «видеть» небо, при этом получение данных о местоположении иногда занимает много времени. GPS также может ускорить разрядку аккумуляторной батареи вашего устройства.

IP-адрес

Для получения информации о местоположении пользователя по его IP-адресу используется внешняя база данных, посредством которой IP-адрес соотносится с физическим местоположением пользователя. Преимущество данного подхода заключается в том, что он может работать везде; однако зачастую оказывается, что IP-адреса принадлежат, например, местному офису интернет-провайдера, обслуживающего пользователей. Данный способ можно считать надежным, если речь идет о городе или его окрестностях.



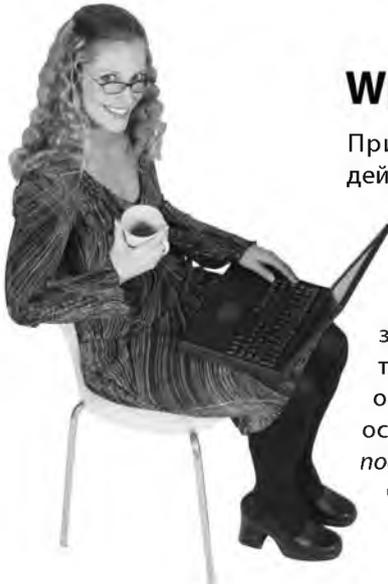
У меня мобильник устаревшей модели. В этом «малыше» нет GPS. Однако посредством триангуляции с использованием вышек сотовой связи мой телефон способен помочь определить мое местоположение с довольно большой точностью, чем может воспользоваться браузер.



Мобильный телефон

Триангуляция с использованием вышек сотовой связи и мобильного телефона позволяет определить местоположение, взяв за основу расстояние, на котором его обладатель находится от одной или более вышек (очевидно, что чем больше будет вышек, тем точнее окажется информация о местоположении). Данный способ может давать довольно точный результат и работает в помещениях (в отличие от GPS); кроме того, иногда он позволяет намного быстрее получить итоговые данные, чем GPS. Однако если пользователь окажется в какой-нибудь глухомани, где есть только одна вышка сотовой связи, точность определения его местоположения будет невысокой.

Я перемещаюсь из одного кафе в другое с ноутбуком и беспроводным подключением. Определить мое местоположение можно посредством триангуляции с использованием точек беспроводного доступа. Данный способ должен довольно хорошо работать.



WiFi

При WiFi-позиционировании задействуется одна или более точек доступа WiFi, что позволяет вычислить местоположение пользователя. Данный способ может давать весьма точный результат, является быстрым и работает в помещениях. Очевидно, что он требует, чтобы пользователь оставался в некоторой степени неподвижным (например, сидел и пил чай со льдом в кафе).



Здорово, что у нас
есть так много способов опре-
делить свое местоположение.
А как узнать, какой из них будет
использовать мое устройство.

Никак.

Краткий ответ на ваш вопрос — «никак», поскольку подход к определению местоположения зависит от реализации браузера. Однако есть и хорошая новость: браузер может использовать *любой* из упоминавшихся выше способов определения местоположения. Фактически, если браузер достаточно умен, он сначала может задействовать триангуляцию с использованием вышек сотовой связи, если таковая доступна, для грубой оценки местоположения, а затем выдать вам более точный результат посредством WiFi или GPS.

Позже вы увидите, что не стоит беспокоиться о том, как именно определяется местоположение. Вместо этого мы сосредоточимся на точности определения местоположения. Исходя из точности результатов вы сможете решить, насколько полезными окажутся для вас полученные данные. Мы вернемся к вопросу точности чуть позже.

Возьми в руку карандаш



Задумайтесь об HTML-страницах и приложениях, которые уже у вас имеются (или которые вы хотите создать); для чего может быть полезна включенная в них функция, позволяющая определять местоположение пользователей?

- Дать пользователям возможность отыскивать находящихся поблизости других людей с аналогичными интересами.
- Помогать пользователям находить местные развлекательные заведения или услуги.
- Отслеживать, где именно пользователи что-то делают.
- Указывать направление пользователям относительно той точки, где они находятся в текущий момент.
- Использовать данные о местоположении для выяснения иной демографической информации о пользователях.
-
-
-
-



Свои идеи напишите здесь!

Так где же вы находитесь?

Конечно же, вы знаете, где находитесь, однако давайте посмотрим, каково ваше местоположение с точки зрения браузера. Для этого мы создадим небольшую HTML-страницу.



```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Where am I?</title>
  <script src="myLoc.js"></script>
  <link rel="stylesheet" href="myLoc.css">
</head>
<body>
  <div id="location">
    Your location will go here.
  </div>
</body>
</html>
```

В верхней части располагаются привычные вещи, включая ссылку на файл `myLoc.js`, в котором будет размещаться наш JavaScript, и таблицу стилей `myLoc.css` для придания приложению красивого внешнего вида.

Наш геолокационный код будет размещаться в `myLoc.js`.

Данный элемент `<div>` будет использоваться для вывода информации о вашем местоположении.

Весь этот HTML необходимо поместить в файл `myLoc.html`.

Теперь создадим файл `myLoc.js` и напишем немного кода; сделаем это быстро, а чуть позже вернемся к данному коду и проанализируем его. Добавьте приведенный ниже код в файл `myLoc.js`.

```
window.onload = getMyLocation;

function getMyLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(displayLocation);
  } else {
    alert("Oops, no geolocation support");
  }
}
```

Мы вызываем функцию `getMyLocation`, как только браузер заканчивает загрузку страницы.

Проверяем, поддерживает ли браузер API-интерфейс `Geolocation`; если объект `navigator.geolocation` присутствует, то все в порядке!

Если поддержка имеется, вызываем метод `getCurrentPosition` и передаем функцию обработчика событий `displayLocation`. Мы реализуем ее через несколько мгновений.

Функция `displayLocation` — это обработчик, которому будет передаваться объект с данными о местоположении.

Если браузер НЕ поддерживает API-интерфейс `Geolocation`, то просто выводим диалоговое окно `alert` с соответствующим сообщением для пользователя.

Обработчик, который будет вызываться, когда браузер получит данные о местоположении.

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

    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
}
```

Обработчику `getCurrentPosition` передается объект `position`, содержащий широту и долготу вашего местоположения (наряду с данными, касающимися точности, о которых мы поговорим немного позже).

Извлекаем широту и долготу вашего местоположения из объекта `position.coords`.

Затем извлекаем наш `<div>` из HTML...

... и задаем ваше местоположение в качестве содержимого элемента `<div>` с использованием `innerHTML`.

Тест-драйв местоположения

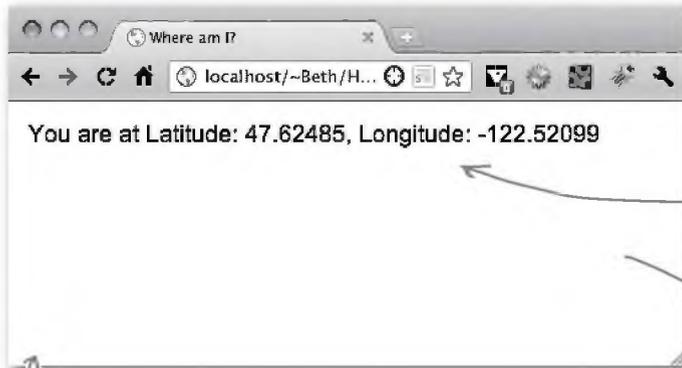


Наберите приведенный выше код и проведите тест-драйв своей новой страницы.

При первом выполнении геолокационного веб-приложения вы увидите в браузере окно с запросом на разрешение использовать данные о вашем местоположении. Это проверка системы защиты браузера, и вы можете отказать браузеру в данном запросе. Поскольку предполагается, что вы хотите протестировать свое веб-приложение, нужно будет нажать кнопку **Allow** (Разрешить) или **Yes** (Да). После этого приложение выдаст координаты вашего местоположения, как показано ниже.



Запрос на разрешение может выглядеть по-разному в зависимости от используемого вами браузера, но будет приблизительно таким.



А вот и ваши координаты! Ваше местоположение определено будет отличаться от нашего (а если нет, то мы будем волноваться за вас).

Если ваше приложение не выдает координаты местоположения и вы дважды проверили код на наличие опечаток, то немного подождите, поскольку через несколько страниц вы познакомитесь с диагностическим тестом, который поможет найти причину неполадок...

Имейте в виду, что получение данных о местоположении не всегда происходит мгновенно и может занять некоторое время...

Если ваш браузер поддерживает API-интерфейс Geolocation, то вы обнаружите свойство geolocation в объекте navigator.



Что мы только что сделали...

Теперь, когда мы с вами создали и протестировали геолокационный код (онять-таки, если данное приложение не выдало вам координат, потерпите немного, поскольку мы очень скоро поговорим об отладочных методиках), давайте пройдемся по коду более детально.

- 1 Первое, что вам необходимо знать, если вы собираетесь написать геолокационный код, — это «поддерживает ли данный браузер API-интерфейс Geolocation?». Опирайтесь на тот факт, что свойство geolocation присутствует в объекте navigator браузера только в том случае, если он поддерживает API-интерфейс Geolocation.

Таким образом, мы можем проверить, присутствует ли свойство geolocation, и если оно имеется, то воспользуемся им; в противном случае мы дадим пользователю знать об отсутствии поддержки API-интерфейса Geolocation посредством вывода соответствующего сообщения в диалоговом окне alert:

```
if (navigator.geolocation) {
    ...
} else {
    alert("Oops, no geolocation support");
}
```

Мы можем выяснить, имеется ли в объекте свойство geolocation (если оно отсутствует, то результатом оценки navigator.geolocation окажется null, и значение соответствующего условия будет false).

Если свойство присутствует, то мы сможем использовать его, а если его нет, то уведомляем пользователя посредством диалогового окна alert.

- 2 Если мы все же обнаружили свойство navigator.geolocation, можно воспользоваться им. Фактически, данное свойство представляет собой объект, содержащий весь API-интерфейс Geolocation. Основным методом, поддерживаемый этим API-интерфейсом, называется getCurrentPosition и занимается извлечением информации о местоположении браузера. Более пристально взглянем на данный метод, обладающий тремя параметрами, два последних из которых являются опциональными:

API-интерфейсы — это просто объекты со свойствами и методами! Теперь вы рады, что загодя прошли JavaScript-подготовку?

successHandler — функция, которая вызывается, если браузер способен успешно определить ваше местоположение.

errorHandler является другой функцией, которая вызывается, если что-то пойдет не так и браузер не сможет определить ваше местоположение.

```
getCurrentPosition(successHandler, errorHandler, options)
```

↑ ↑ ↑
Эти два параметра являются опциональными, что объясняет, почему они не требовались нам ранее.

↑ Параметр options позволяет сконфигурировать работу API-интерфейса Geolocation.

- 2 А сейчас взглянем на вызов метода `getCurrentPosition`. Мы передаем аргумент в виде обработчика успешного исполнения для обработки удачной попытки извлечения данных о местоположении браузера. Чуть позже мы рассмотрим ситуацию, когда браузеру не удастся отыскать местоположение.

```
if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(displayLocation);
}
```

Вызываем метод `getCurrentPosition` объекта `geolocation` с использованием одного аргумента — обработчика успешного исполнения.

Помните объединение в цепочку, о котором говорилось в главе 4? Мы используем объект `navigator` для получения доступа к объекту `geolocation`, который является свойством объекта `navigator`.

Когда API-интерфейс определит ваше местоположение, он вызовет `displayLocation`.

Вы заметили, что здесь мы передаем одну функцию другой функции? Как отмечалось в главе 4, функции являются значениями, поэтому мы можем это делать.

- 1 Теперь взглянем на обработчик успешного исполнения `displayLocation`. При вызове `displayLocation` API-интерфейс `Geolocation` передает ему объект `position`, включающий информацию о местоположении браузера, в том числе объект `coordinates`, в котором содержатся широта и долгота (а также прочие значения, о которых мы поговорим позже).

`position` — это объект, который API-интерфейс `Geolocation` передает вашему обработчику успешного исполнения.

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

    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
}
```

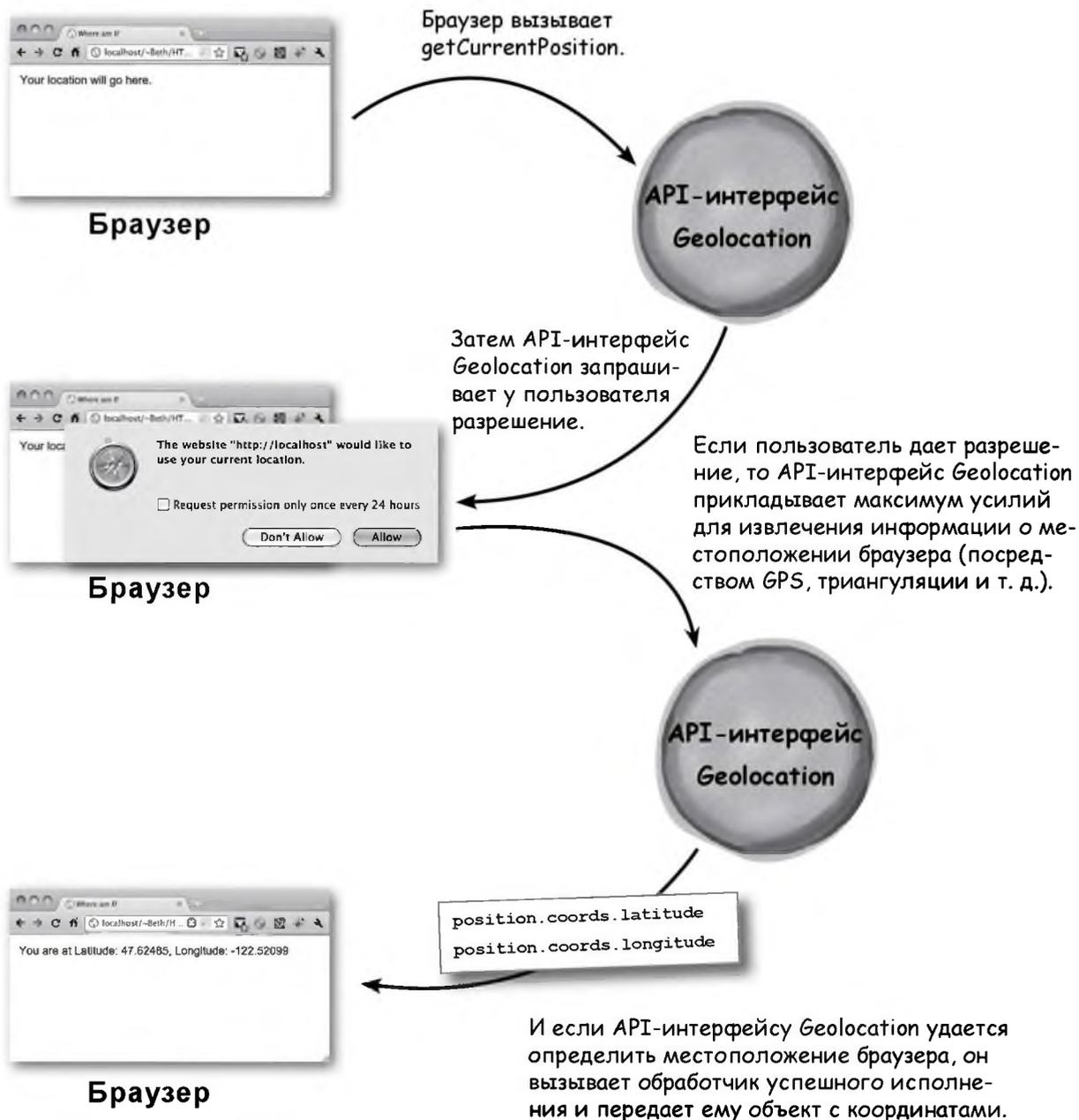
Объект `position` обладает свойством `coords`, которое содержит ссылку на объект `coordinates`...

...объект `coordinates`, в свою очередь, содержит ваши широту и долготу.

А эта часть к настоящему моменту должна быть вам абсолютно ясна: мы просто берем информацию о координатах и отображаем ее в элементе `<div>` страницы.

Как все это работает

Теперь, когда мы прошлись по коду, давайте посмотрим, как все это работает во время выполнения.



Диагностический тест-драйв



Когда дело касается API-интерфейса Geolocation, не каждый тест-драйв оказывается удачным; даже если в случае с первым тестом все пройдет успешно, в дальнейшем все равно что-то может пойти не так. Поэтому мы создали небольшой диагностический тест, который вы можете добавить в свой код. Таким образом, если у вас возникнут проблемы, вы сможете выяснить их причины; и даже если они обойдут вас стороной, кто-то из ваших пользователей все равно может столкнуться с той или иной проблемой, и вам будет необходимо знать, как решить ее в своем коде. Поэтому добавьте приведенный ниже код, и если вы обнаружите неполадки, заполните диагностическую форму в конце данной секции, указав в ней проблему, которую вам удалось диагностировать:

Для создания диагностического теста мы добавим обработчик ошибок в вызов метода `getCurrentPosition`. Данный обработчик будет вызываться всякий раз, когда API-интерфейс Geolocation сталкивается с рудностями определения вашего местоположения. Вот как будет осуществляться его добавление:

Добавьте второй аргумент в свой вызов `getCurrentPosition` с именем `displayError`. Это функция, которая будет вызываться, когда API-интерфейсу Geolocation не удастся определить местоположение.

```
navigator.geolocation.getCurrentPosition(displayLocation, displayError);
```

Теперь необходимо написать код обработчика ошибок. Для этого вам нужно знать, что API-интерфейс Geolocation передает вашему обработчику объект `error`, содержащий числовой код с описанием причины, по которой он не смог определить местоположение вашего браузера. В зависимости от кода также может выводиться сообщение с дополнительной информацией о возникшей ошибке. Вот как мы можем использовать объект `error` в обработчике:

Вот наш новый обработчик, которому API-интерфейс Geolocation передает объект `error`.

```
function displayError(error) {
    var errorTypes = {
        0: "Unknown error",
        1: "Permission denied by user",
        2: "Position is not available",
        3: "Request timed out"
    };
    var errorMessage = errorTypes[error.code];
    if (error.code == 0 || error.code == 2) {
        errorMessage = errorMessage + " " + error.message;
    }
    var div = document.getElementById("location");
    div.innerHTML = errorMessage;
}
```

Объект `error` содержит свойство `error.code`, которое имеет числовое значение от 0 до 3. Вот отличный способ ассоциировать сообщение об ошибке с соответствующим кодом JavaScript.

Создаем объект с несколькими свойствами, именуемыми имена 0, 1, 2 и 3 соответственно. Эти свойства представляют собой строки с сообщениями об ошибке, которые мы хотим ассоциировать с соответствующим кодом.

Используя свойство `error.code`, мы присваиваем одну из этих строк новой переменной `errorMessage`.

В случае с ошибками 0 и 2 иногда в свойстве `error.message` присутствует дополнительная информация, поэтому мы добавляем его в нашу строку `error.Message`.

А затем добавляем сообщение к странице для уведомления пользователя.



Прежде чем выполнить тест, более пристально взглянем на типы ошибок, которые могут выводиться на экран.

```
var errorTypes = {  
  0: "Unknown error",  
  1: "Permission denied by user",  
  2: "Position is not available",  
  3: "Request timed out"  
};
```

Это общая ошибка, которая выводится в ситуациях, когда ни одна из остальных ошибок не подходит. Обратитесь к свойству `error.message` для получения дополнительной информации.

Это означает, что пользователь отказал в запросе на разрешение использовать информацию о местоположении.

А это означает, что браузер попытался, но не смог определить ваше местоположение. Опять-таки, для получения дополнительной информации обратитесь к свойству `error.message`.

И наконец, API-интерфейс *Geolocation* предусматривает такую внутреннюю настройку, как время ожидания (*timeout*), и если оно истекает до того, как будет определено местоположение, выводится данная ошибка.

Позже в этой главе вы узнаете, как изменять значение `timeout`, задаваемое по умолчанию в случае API-интерфейса *Geolocation*.

Набрав диагностический тест, запустите его. Если вы получите координаты местоположения, значит, все работает нормально, и никаких сообщений об ошибках выводиться не будет. Вы можете форсировать вывод сообщения об ошибке, отказав браузеру в его запросе на разрешение использовать информацию о вашем местоположении. Либо вы можете проявить более творческий подход: например, войти в помещение со своим телефоном, поддерживающим GPS, что приведет к пропаданию GPS-сигнала от сети спутников.

В наихудшем случае, если вы долго ждете, но данные о местоположении так и не поступают и не выводится никаких сообщений об ошибке, то, скорее всего, для `timeout` задано большое значение, то есть время ожидания будет продолжительным. Чуть позже в этой главе мы расскажем, как сократить длительность ожидания.



Укажите здесь результаты проведенной вами диагностики

- Я не давал разрешения использовать информацию о моем местоположении.
- Мое местоположение оказалось недоступно.
- Я получил сообщение о том, что время ожидания ответа на запрос истекло.
- Вообще ничего не произошло — я не получил ни координат местоположения, ни сообщения об ошибке.
- Нечто другое _____



Будьте
осторожны!

Для тестирования своего геолокационного кода на мобильном устройстве вам потребуется сервер.

Если у вас нет средств загрузки своих HTML-, JavaScript- и CSS-файлов напрямую на мобильное устройство, то их

будет проще всего протестировать, разместив на сервере (загляните в следующую главу, чтобы узнать, как установить свой собственный сервер в случае необходимости), и обращаться к ним уже там. Если же у вас имеется сервер и вы решите именно так и поступить, мы полностью вас поддерживаем. С другой стороны, если данный вариант вам не подходит, знайте, что мы разместили соответствующий код на серверах Wickedly Smart, так что вы сможете провести тестирование с использованием своих мобильных устройств. Мы также рекомендуем вам сначала протестировать код на своем настольном компьютере; если все будет нормально работать, переходите к его тестированию на мобильном устройстве, используя сервер (свой собственный или Wickedly Smart).

При проведении первого тест-драйва (включая диагностирование ошибок) вводите на своем устройстве адрес <http://wickedlysmart.com/hfhtml5/chapter5/latlong/myL.cc.html>.

Часть
Задаваемые
Вопросы

В: Значения широты и долготы моего местоположения, возвращаемые приложением, не совсем точны. В чем же дело?

О: Существует масса методик, посредством которых ваше устройство и поставщик услуг по определению местоположения вычисляют место, где вы находитесь; при этом одни из них дают более точный результат, чем другие. GPS зачастую позволяет получать самые точные показатели. Мы с вами рассмотрим способ определить уровень точности данных, возвращаемых локационной службой как часть объекта position, благодаря чему вы сможете понять, насколько точные данные о местоположении можно получить.

Раскрываем наше тайное убежище...

Теперь, когда вы изучили основы, сделаем кое-что более интересное в плане определения местоположения. Как посчитать то, чтобы узнать, насколько далеко вы находитесь от нашего тайного убежища — штаб-квартиры Wickedly Smart? Для этого нам потребуются координаты данной штаб-квартиры, а также знание того, как вычисляется расстояние между двумя координатами. Сначала добавим еще один элемент `<div>` в HTML:

```

:
<body>
  <div id="location">
    Your location will go here.
  </div>
  <div id="distance">
    Distance from WickedlySmart HQ will go here.
  </div>
</body>
</html>

```

↖ Добавьте этот новый элемент `<div>` в свой HTML.



↖ Координаты штаб-квартиры Wickedly Smart: 47.62485, -122.52099.



Готово к употреблению: Вычисляем расстояние

Вам когда-нибудь хотелось узнать, как вычисляется расстояние между двумя точками на поверхности планеты? Подробности этого процесса покажутся вам весьма любопытными, однако они лежат немного вне рамок данной главы. Поэтому мы заранее подготовили для вас код, который просто выполняет эту задачу. Для вычисления расстояния между двумя координатами в большинстве случаев применяется формула гаверсинуса, реализация которой приведена ниже.

Данная функция принимает две точки координат — начальную (*startCoords*) и конечную (*destCoords*) — и возвращает расстояние между ними в километрах.

```
function computeDistance(startCoords, destCoords) {
    var startLatRads = degreesToRadians(startCoords.latitude);
    var startLongRads = degreesToRadians(startCoords.longitude);
    var destLatRads = degreesToRadians(destCoords.latitude);
    var destLongRads = degreesToRadians(destCoords.longitude);

    var Radius = 6371; // радиус Земли в километрах
    var distance = Math.acos(Math.sin(startLatRads) * Math.sin(destLatRads) +
        Math.cos(startLatRads) * Math.cos(destLatRads) *
        Math.cos(startLongRads - destLongRads)) * Radius;

    return distance;
}

function degreesToRadians(degrees) {
    var radians = (degrees * Math.PI) / 180;
    return radians;
}
```

← Эту функцию мы еще увидим в главе, посвященной элементу *canvas*.

Добавьте данный код в свой файл *myLoc.js*.

Написание кода для определения расстояния

Теперь, когда у нас имеется функция для вычисления расстояния между двумя точками координат, определим наше (то есть наше, авторов) местоположение — координаты штаб-квартиры Wickedly Smart (паберите и этот код)::

```
var ourCoords = {
  latitude: 47.624851,
  longitude: -122.52099
};
```

Определяем литеральный объект для координат местоположения нашей штаб-квартиры Wickedly Smart. Добавьте его как глобальную переменную в верхнюю часть своего файла myLoc.js.

Мы хотим вычислить расстояние от вас до нас по прямой



А теперь напишем код: все, что нам потребуется сделать, — это передать координаты вашего и нашего местоположения функции computeDistance:

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

  var div = document.getElementById("location");
  div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;

  var km = computeDistance(position.coords, ourCoords);
  var distance = document.getElementById("distance");
  distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
}
```

Передаем координаты вашего и нашего местоположения функции computeDistance.

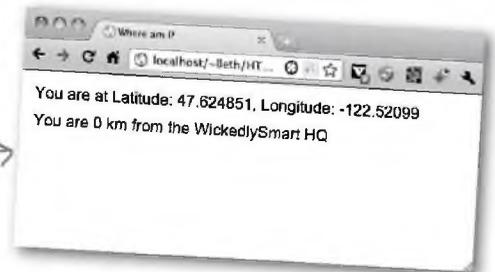
А затем берем результаты и обновляем содержимое элемента <div> с id в виде distance.

Локационный тест-драйв



Проведем тест-драйв нашего нового кода. Завершите добавление кода в myLoc.js, а затем перезагрузите myLoc.html в своем браузере. После этого вы должны увидеть координаты вашего местоположения, а также расстояние, на котором находитесь от нас.

Ваше местоположение и расстояние от штаб-квартиры Wickedly Smart явно будут отличаться в зависимости от того, в какой точке планеты вы находитесь.



Проведите тестирование онлайн: <http://wickedlysmart.com/hfhtml5/chapter5/distance/myLoc.html>

Да, знать координаты своего местоположения в виде 34.20472, -90.57528 — это, конечно, здорово, однако в данный момент как нельзя кстати пришлось бы карта!



Отображение вашего местоположения на карте

Как мы уже отмечали, API-интерфейс Geolocation довольно прост — он позволяет вам определять (и, как вы еще увидите, отслеживать) то, где вы находитесь, однако не предусматривает никаких инструментов для визуализации вашего местоположения. Для этого вам придется обратиться к стороннему инструменту, и, как вы уже могли догадаться, Google Maps — наиболее популярное средство для решения данной задачи. Google Maps не является частью спецификации HTML5, однако может отлично взаимодействовать с HTML5, поэтому мы не прочь немного отклониться кое-где от нашего маршрута и показывать вам, как интегрировать Google Maps с API-интерфейсом Geolocation. Например, вы можете начать с добавления приведенного ниже кода в `<head>` своего HTML-документа, а чуть позже мы поговорим о том, как можно добавить карту к странице:

```
<script src="http://maps.google.com/maps/api/js?sensor=true"></script>
```

↑
Здесь находится API-интерфейс JavaScript Google Maps.

↑ Убедитесь, что вы набрали данный код точно, как показано здесь, включая параметр запроса `sensor` (API-интерфейс не будет без него работать). Мы указали `sensor=true`, поскольку наш код задействует ваше местоположение. Если бы мы просто использовали карту без вашего местоположения, то напечатали бы `sensor=false`.

Как добавить карту к странице

Теперь, когда вы указали ссылку на API-интерфейс Google Maps, все его возможности будут доступны вам посредством JavaScript. Однако нам потребуется место для размещения карты Google, для чего необходимо определить элемент, который будет ее содержать.

```
⋮
<body>
  <div id="location">
    Your location will go here.
  </div>
  <div id="distance">
    Distance from WickedlySmart HQ will go here.
  </div>
  <div id="map">
  </div>
</body>
</html>
```

← Вот наш <div>. Следует отметить, что мы определили стиль в `myLoc.js`, который задает для элемента <div> с `id` в виде `map` размер 400 на 400 пикселей и черную рамку.

Подготовка к созданию карты...

Для создания карты нам потребуются широта и долгота (а мы уже знаем, как их извлечь), а также набор параметров, описывающих, какой именно должна быть наша карта. Начнем с широты и долготы. Мы знаем, как извлекать их посредством API-интерфейса Geolocation. При этом следует отметить, что API-интерфейс Google Maps предпочитает, чтобы они были «упакованы» в его собственный объект. Для создания одного из таких объектов мы можем воспользоваться конструктором, предусмотренным API-интерфейсом Google Maps:

```
var googleLatAndLong = new google.maps.LatLng(latitude, longitude);
```

google.maps ставится перед именами всех методов API-интерфейса Google Maps.

↑ Конструктор, который принимает широту и долготу и возвращает новый объект, где они будут содержаться.

← Не забывайте, что имена конструкторов должны начинаться с прописной буквы.

API-интерфейс Google Maps обеспечивает наличие параметров, которые мы можем задавать с целью контроля итогового вида нашей карты. Например, можно указать, насколько увеличенным или уменьшенным будет исходное представление карты, какое местоположение будет отображаться в ее центре, какого типа будет карта (например, ROADMAP), предусмотреть представление Satellite (Спутник) и т. д. Параметры задаются следующим образом:

```
var mapOptions = {
  zoom: 10,
  center: googleLatAndLong,
  mapTypeId: google.maps.MapTypeId.ROADMAP
};
```

← Для параметра `zoom` можно указывать значение от 0 до 21. Экспериментируйте с ним: более высокие значения соответствуют больше-му увеличению (то есть вы сможете рассмотреть больше деталей). Значение 10 подразумевает, так сказать, размер «с город».

← Новый объект, который мы только что создали. Мы хотим, чтобы данное местоположение отображалось в центре карты.

← Вы также можете попробовать поставить сюда `SATELLITE` или `HYBRID`.

Отображение карты

Соберем все воедино в новой функции с именем `showMap`, которая принимает набор координат и отображает карту на веб-странице:

```

var map;

function showMap(coords) {
  var googleLatAndLong =
    new google.maps.LatLng(coords.latitude,
      coords.longitude);

  var mapOptions = {
    zoom: 10,
    center: googleLatAndLong,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  };

  var mapDiv = document.getElementById("map");
  map = new google.maps.Map(mapDiv, mapOptions);
}

```

Объявляем глобальную переменную `map`, которая будет содержать нашу карту Google после того, как мы ее создадим. Чуть позже вы увидите, как она используется.

Задействуем широту и долготу из объекта `coords`...

...и применяем их для создания объекта `google.maps.LatLng`.

Создаем объект `mapOptions` с параметрами, которые хотим задать для нашей карты.

И наконец, извлекаем `map.Div` из объектной модели документа (DOM) и передаем его вместе с `mapOptions` конструктору `Map` с целью создания объекта `google.maps.Map`. Посредством него на нашей странице будет отображаться карта.

Еще один конструктор из API-интерфейса Google Maps, который принимает элемент и наши параметры, после чего создает и возвращает объект `map`.

Присваиваем новый объект `Map` нашей глобальной переменной `map`.

Добавьте данный код в нижнюю часть своего JavaScript-файла. Затем останется лишь присоединить его к уже имеющемуся у нас коду. Сделаем это путем редактирования функции `displayLocation`:

```

function displayLocation(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;

  var div = document.getElementById("location");
  div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;

  var km = computeDistance(position.coords, ourCoords);
  var div = document.getElementById("distance");
  distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";

  showMap(position.coords);
}

```

Будем вызывать `showMap` из `displayLocation` после того, как обновим остальные `<div>` на странице.

Тест-драйв нашей новой карты

Убедитесь, что вы добавили весь новый код с предыдущей страницы, а также элемент `<div>` с `id` в виде `map` в свой HTML. Затем перезагрузите веб-страницу, и если браузеру удастся определить ваше местоположение, вы увидите карту.

А вот и наша новая карта Google!

Здесь демонстрируется местоположение велосипедиста с координатами 34.20472, -90.57528; вы, естественно, скорее всего будете находиться в другом месте.

Здорово! А есть ли способ увидеть мое точное местоположение на карте? Отмеченное, например, булавкой?

Вы действительно хотите, чтобы эта штука валялась рядом с вашим велосипедом?

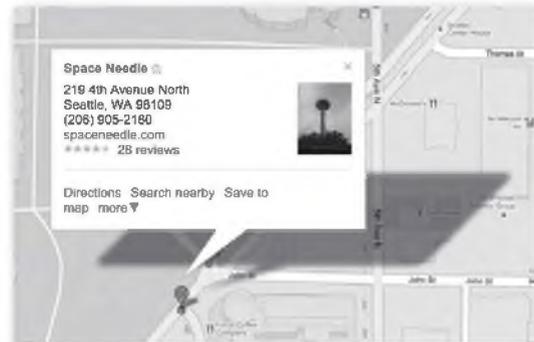


Проведите тестирование онлайн: <http://wickedlysmart.com/hfhtml5/chapter5/map/myLoc.html>

Прикалываем булавку на карту...

Будет намного лучше, если вы сможете увидеть на карте свое точное местоположение. Если вам доводилось пользоваться Google Maps, то вы, вероятно, видели, что там местоположение искомым физическим объектам отмечается с помощью своего рода булавок. Например, если вы ищете башню Space Needle («Космический шпиль») в Сиэтле, штат Вашингтон, то система поиска выдаст вам карту, где рядом с этой башней в городе будет «приколота» булавка, щелкнув на которой вы увидите информационное окно с подробностями о данном сооружении. Эти булавки называются *маркерами* и являются одной из многих возможностей, которые предлагает API-интерфейс Google Maps.

Добавление маркера с всплывающим информационным окном потребует написания некоторого кода, поскольку вам будет необходимо создать сам маркер, информационное окно и обработчик событий `click`, инициируемые при щелчке пользователя на данном маркере (что приводит к открытию окна). Поскольку мы отклоняемся от маршрута, на рассмотрение данного аспекта потратим не много времени, тем более, что у вас уже имеются все необходимые знания, чтобы без особого труда во всем разобраться!



Если вы ищете какой-либо объект в Google Maps, то его местоположение на карте будет отмечено красным маркером.

① Начнем с создания новой функции `addMarker`, после чего воспользуемся API-интерфейсом Google Maps для создания маркера:

Функция `addMarker` принимает `map`, `latlong`, `title` для маркера, а также `content` для информационного окна.

```
function addMarker(map, latlong, title, content) {
```

```
  var markerOptions = {
```

```
    position: latlong,
```

```
    map: map,
```

```
    title: title,
```

```
    clickable: true
```

```
  };
```

```
  var marker = new google.maps.Marker(markerOptions);
```

```
};
```

Создаем объект `markerOptions` с `latlong`, `map` и `title`, а также указываем, хотим ли мы, чтобы пользователь имел возможность щелкнуть на маркере...

...задаем для `clickable` значение `true`, поскольку хотим, чтобы пользователь мог щелчком на маркере вывести информационное окно.

Затем создаем объект `marker`, используя еще один конструктор из API-интерфейса Google Maps, которому передаем ранее созданный объект `markerOptions`.

- ② Создадим информационное окно путем определения специфичных для него параметров, после чего сгенерируем новый объект `InfoWindow` с помощью API-интерфейса `Google Maps`. Добавьте приведенный ниже код в свою функцию `addMarker`:

```
function addMarker(map, latlong, title, content) {
```

```
  : ← Наш остальной код по-прежнему здесь, мы просто экономим место...
```

```
  var infoWindowOptions = {
```

← А теперь определяем параметры информационного окна.

```
    content: content,
```

← Нам потребуется содержимое...

```
    position: latlong
```

← ...а также широта и долгота.

```
  };
```

```
  var infoWindow = new google.maps.InfoWindow(infoWindowOptions);
```

← С помощью этого мы создаем информационное окно.

```
  google.maps.event.addListener(marker, "click", function() {
```

```
    infoWindow.open(map);
```

```
  });
```

↑
} Когда пользователь щелкает на маркере, происходит вызов данной функции, и на карте открывается информационное окно.

↑ ↑ ↑
Передаем слушателя функции, которая вызывается, когда пользователь щелкает на маркере.

←
Затем мы используем метод `addListener` API-интерфейса `Google Maps` для добавления слушателя событий `click`. Слушатель подобен обработчику событий (например, `onload` или `onclick`), с которым вы сталкивались ранее.

- ③ Осталось только вызвать функцию `addMarker` из `showMap`, убедившись, что мы передали все надлежащие аргументы для четырех параметров. Добавьте данный код в нижнюю часть своей функции `showMap`:

```
var title = "Your Location";
```

```
var content = "You are here: " + coords.latitude + ", " + coords.longitude;
```

```
addMarker(map, googleLatAndLong, title, content);
```

↑
Передаем объекты `map` и `googleLatAndLong`, созданные с использованием API-интерфейса `Google Maps`...

↑ ↑ ←
... а также строки `title` и `content` для маркера.

Тест-драйв маркера



Добавьте весь код, касающийся `addMarker`, обновите `showMap` для вызова `addMarker` и перезагрузите страницу. После этого вы увидите карту с маркером, указывающим ваше местоположение.

Щелкните по маркеру кнопкой мыши. В результате этого появится всплывающее окно, в котором будут отображаться ширина и долгота вашего местоположения.

Все это здорово, поскольку теперь вы можете точно узнать, где находитесь (например, если заблудились).

Вот как будет выглядеть наша карта с маркером и всплывающим окном.



Проведите тестирование онлайн: <http://wickedlysmart.com/html5/chapter5/marker/myLoc.html>

Прочие интересные вещи, которые можно сделать с использованием API-интерфейса Google Maps



Мы лишь поверхностно затронули то, что можно сделать с помощью Google Maps, подробное изучение данного API-интерфейса выходит за рамки настоящей книги. Но у вас не возникнет сложностей, если вы захотите сделать это самостоятельно. Приведем ряд вещей, для реализации которых он может использоваться, а также некоторые подсказки для вас о том, с чего начать.

Элементы управления: ваша карта Google по умолчанию будет включать несколько элементов управления, позволяющих, например, изменять масштаб, осуществлять панорамирование, переключаться между представлениями Map (Карта) и Satellite (Спутник) и даже просматривать улицы (позволяющий делать это элемент управления выглядит как маленький оранжевый человечек над кнопками масштабирования). Вы можете обращаться к этим элементам управления программно из JavaScript, чтобы использовать их в своих приложениях.

Службы: вам когда-нибудь доводилось искать маршруты с помощью Google Maps? Если да, то вы пользовались при этом службой Directions (Направления). Для обращения к этой и другим службам, позволяющим, например, узнавать расстояние и просматривать улицы, вам потребуется прибегнуть к API-интерфейсам Google Maps для работы со службами.

Слои: они позволяют размещать дополнительное представление поверх карты Google (например, карту погоды). Если вы собираетесь в поездку, то сможете проверить загруженность дорог транспортом с помощью слоя трафика. Используя API-интерфейсы Google Maps, позволяющие работать со слоями, вы получаете возможность создавать пользовательские слои (например, пользовательские маркеры, применять в слоях свои фотографии) и делать массу всего другого, что только сможете себе представить.

Все это доступно посредством API-интерфейса JavaScript Google Maps. Если у вас возникнет желание пойти еще дальше в своих экспериментах, изучите документацию по адресу:

<http://code.google.com/apis/maps/documentation/javascript/>



ВСТРЕЧАЕМ API-ИНТЕРФЕЙС GEOLOCATION

Интервью недели:
Разговор с тем, кто хочет стать
API-интерфейсом HTML5...

Head First: Добро пожаловать, API-интерфейс Geolocation. Сразу хочу отметить, что немного удивлен видеть Вас здесь.

Geolocation: Почему же?

Head First: Вы даже не являетесь «официальной» частью спецификации HTML5, однако стали первым API-интерфейсом, которому была отведена целая глава в данной книге! Чем это объясняется?

Geolocation: Что ж, Вы правы в том, что я определен в спецификации, отдельной от HTML5, однако являюсь официальным стандартом W3C. Оглянитесь вокруг: я уже реализован в браузерах, устанавливаемых на любом стоящем мобильном устройстве. Под этим я подразумеваю следующее: будет ли много пользы от мобильного веб-приложения, если оно не поддерживает меня?

Head First: Какие приложения задействуют Вас?

Geolocation: К их числу относится большинство приложений, которыми люди пользуются в дороге: начиная с приложений, позволяющих обновлять свой статус и включать геолокационную информацию, и заканчивая приложениями для фотокамер, фиксирующими то, где были сделаны снимки, а также социальными приложениями, дающими возможность отыскать местных друзей или отметить в разных местах. Люди используют меня даже для фиксации того, где они ездили на велосипеде, бегали или останавливались перекусить, а также для того, чтобы добраться до нужного пункта назначения.

Head First: Вы как API-интерфейс выглядите просто, то есть я хочу сказать, что у Вас в сумме имеется лишь несколько методов и свойств, ведь так?

Geolocation: В компактности и простоте заключена мощь. Разве многие жалуются на меня? Нет. У меня есть то, что нужно всем разработчикам, которые каждый день создают массу приложений с поддержкой определения местоположения. Кроме того, моя компактность означает, что меня можно быстро и легко освоить, не так ли? Может, в этом и заключается при-

чина, по которой я стал первым API-интерфейсом, которому была отведена отдельная глава?

Head First: Давайте поговорим о поддержке.

Geolocation: Тут не о чем особо говорить, поскольку меня поддерживают почти все браузеры как в настольном, так и в мобильном сегменте.

Head First: Какая от Вас польза на устройствах, которые не поддерживают GPS?

Geolocation: Это заблуждение, что я как-то зависим от GPS. На сегодняшний день существует масса других способов определять местоположение, например посредством триангуляции с использованием вышек сотовой связи и мобильного телефона, посредством IP-адресов и т. д. Если в Вашем устройстве есть GPS, то это здорово, поскольку в этом случае я смогу быть для Вас еще более полезным; если же GPS отсутствует, то Вы всегда можете прибегнуть к массе других способов определения местоположения.

Head First: Быть еще более полезным?

Geolocation: Если у Вас достаточно хорошее мобильное устройство, то я смогу сообщить Вам еще и высоту, направление, скорость, то есть все возможные показатели.

Head First: Допустим, что ни один из этих способов не сработал — ни GPS, ни IP-адрес, ни триангуляция. Какой тогда от Вас толк?

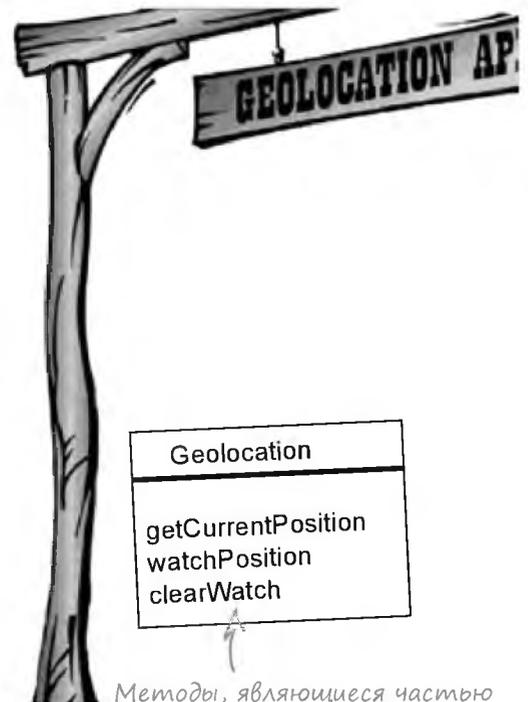
Geolocation: Что ж, я не всегда могу ручаться, что Вам обязательно удастся определить местоположение, однако есть и положительный момент — я даю возможность выяснить причину проблемы. Все, что Вам нужно сделать, — это дать мне обработчик ошибок, и я буду вызывать его в случае возникновения проблем.

Head First: И это хорошо. Что ж, наше время истекло. Благодарю Вас, API-интерфейс Geolocation, за то, что посетили нас. И поздравляю с обретением статуса настоящего стандарта W3C!

Возвращаясь к API-интерфейсу Geolocation...

Мы с вами уже прошли довольно длинный путь, используя API-интерфейс Geolocation: мы определяли наше местоположение, вычисляли расстояние до других объектов, обрабатывали состояния ошибки API-интерфейса и даже добавляли карту с помощью API-интерфейса Google Maps. Однако отдохнуть еще рано, поскольку мы только подошли к рассмотрению интересных особенностей API-интерфейса Geolocation. Мы также приблизились к точке, которая отделяет человека, знающего о данном API-интерфейсе, от того, кто мастерски им владеет, поэтому не будем останавливаться!

Прежде чем двинуться дальше, нам необходимо более пристально *взглянуть* на сам этот API-интерфейс. Как уже отмечалось ранее, данный API-интерфейс довольно прост и включает лишь три метода: `getCurrentPosition` (о нем вам уже кое-что известно), `watchPosition` (подробности о нем вы узнаете довольно скоро) и `clearWatch` (оп, как вы уже догадались, связан с `watchPosition`). Перед тем как мы перейдем к этим двум новым методам, еще раз взглянем на `getCurrentPosition` и связанные объекты, такие как `Position` и `Coordinates`. Вы откроете для себя то, о чем раньше не знали.



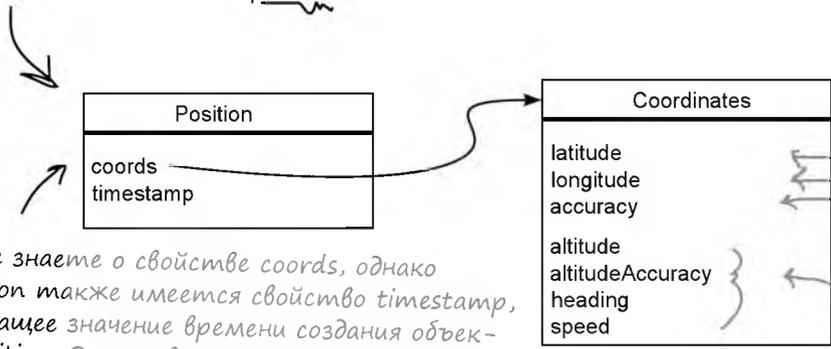
Методы, являющиеся частью API-интерфейса Geolocation.

Вызов `errorHandler` происходит, когда браузеру не удается определить местоположение пользователя. Как мы уже видели, причин тому может быть немало.

`getCurrentPosition(successHandler, errorHandler, positionOptions)`

Как вы можете помнить, `successHandler` (или функция обратного вызова) вызывается, когда местоположение определено, и ему передается объект `position`.

У нас также имеется новый параметр, который мы еще не использовали, позволяющий настроить поведение API-интерфейса Geolocation.



Вам уже знакомы `latitude` и `longitude`, однако в объекте `coordinates` имеются и другие свойства.

Эти три гарантированно будут там: `latitude`, `longitude` и `accuracy`.

Остальные могут как поддерживаться, так и не поддерживаться, — это зависит от используемого устройства.

Вы уже знаете о свойстве `coords`, однако в `position` также имеется свойство `timestamp`, содержащее значение времени создания объекта `position`. Оно позволяет узнать, насколько старым является местоположение.

Можем ли мы поговорить о точности?

Определение местоположения не является точной наукой. В зависимости от методики, применяемой вашим браузером, вы сможете узнать только штат, город и квартал города, в котором находитесь. С другой стороны, если устройство будет более продвинутым, то вы сможете определить собственное местоположение с точностью до 10 метров, включая показатели своей скорости, направления и высоты.

Так как же писать код, исходя из данной ситуации? Разработчики API-интерфейса Geolocation заключили с нами небольшой договор: каждый раз, когда они дают вам координаты местоположения, они также сообщают вам их точность в метрах в пределах 95%-ного уровня достоверности. Так, если бы мы вам сообщили ваше местоположение с точностью до 500 метров, то мы могли бы быть достаточно уверены в его достоверности, если не выходить за пределы этих 500 метров. В таком случае можно, например, давать пользователям верные рекомендации относительно города или его окрестностей, но не сообщать им подробный уличный маршрут проезда. В любом случае очевидно, что ваше приложение будет решать, как оно хочет использовать данные, касающиеся точности.

Но хватит разговоров, давайте взглянем на точность в случае с вашим текущим местоположением. Как вы уже знаете, информация, касающаяся точности, является частью объекта `coordinates`. Извлечем ее, а затем используем в функции `displayLocation`.

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var div = document.getElementById("location");

    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
    div.innerHTML += " (with " + position.coords.accuracy + " meters accuracy)";

    var km = computeDistance(position.coords, ourCoords);
    var div = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
    showMap(position.coords);
}
```

Используем свойство `accuracy`, которое добавляем в конец `innerHTML` элемента `<div>`.



Проверка точности



Убедитесь, что вы добавили приведенную выше выделенную строку кода к своей странице, после чего загрузите ее. В результате вы увидите, насколько точно было определено ваше местоположение. Не забудьте провести данный тест на имеющемся у вас мобильном устройстве.

Тест онлайн: <http://wickedlysmart.com/hfhtml5/chapters/accuracy/myLoc.html>

«Wherever you go, there you are»

Источник происхождения фразы из заголовка (одно из ее значений: «Куда бы ты ни шел — ты именно там», то есть в философском смысле здесь подразумевается, что от себя не убежишь) горячо обсуждается. Некоторые утверждают, что она впервые упоминалась в фильме «Бакару Банзай» (Buckaroo Banzai), в то время как другие полагают, что она происходит из дзэн-буддийских текстов, а третьи ссылаются на различные книги, прочие кинофильмы и популярные песни. Неважно, что именно послужило ее источником, главное — она была увековечена, и тем более сохранится после данной главы, потому что мы превратим ее в небольшое веб-приложение с соответствующим именем. Однако вам потребуется вашего участия с вашей читательской стороны.

Вам предстоит расширить свой текущий код таким образом, чтобы он смог отслеживать ваше перемещение в реальном времени. Для этого мы соберем все вместе, включая два последних метода, имеющих в API-интерфейсе Geolocation, и создадим приложение, которое будет отслеживать ваше перемещение в режиме, близком к режиму реального времени.

Как мы будем отслеживать ваше перемещение

Вы уже знаете, что API-интерфейс Geolocation включает метод `watchPosition`. Данный метод делает то, что подразумевает его имя: он следит за вашим перемещением и докладывает о вашем местоположении по мере того, как оно изменяется. Метод `watchPosition` на самом деле очень схож с методом `getCurrentPosition`, но ведет себя немного по-другому: он повторно вызывает обработчик успешного исполнения каждый раз, когда изменяется ваше местоположение. Давайте посмотрим, как он работает.



Браузер

- 1 Ваше приложение вызывает метод `watchPosition`, передавая ему функцию обработчика успешного выполнения.

- 2 `watchPosition` «сидит» в фоне и постоянно отслеживает ваше местоположение.



```
position.coords.latitude
position.coords.longitude
```

- 4 `watchPosition` будет продолжать отслеживать ваше местоположение (и докладывать о нем обработчику успешного исполнения) до тех пор, пока вы не остановите отслеживание путем вызова `clearWatch`.

- 3 При изменении вашего местоположения `watchPosition` вызывает функцию обработчика успешного исполнения для доклада ей о вашем новом местоположении.



А кто, по-вашему, прав в дебатах по поводу источника происхождения данной фразы? Исходит она от веб-ресурса Banzai Institute или же берет свое начало в дзен-буддийской литературе?

Приступаем к созданию приложения

В качестве отправной точки мы возьмем приводившийся ранее код; сначала добавим пару кнопок в HTML, чтобы можно было запускать и останавливать отслеживание вашего местоположения. Зачем вообще нужны кнопки? Прежде всего пользователи не хотят, чтобы их перемещения отслеживались постоянно, и, как правило, хотят иметь контроль над этой процедурой. Но есть и вторая причина: постоянная проверка местоположения на мобильном устройстве потребляет много электроэнергии, и если оставить ее активированной, это приведет к быстрой разрядке аккумулятора. Поэтому мы в первую очередь обновим HTML с целью добавить форму и две кнопки: одну для запуска отслеживания вашего местоположения, а другую — для его остановки.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Wherever you go, there you are</title>
  <script src="myLoc.js"></script>
  <link rel="stylesheet" href="myLoc.css">
</head>
<body>
  <form>
    <input type="button" id="watch" value="Watch me">
    <input type="button" id="clearWatch" value="Clear watch">
  </form>
  <div id="location">
    Your location will go here.
  </div>
  <div id="distance">
    Distance from WickedlySmart HQ will go here.
  </div>
  <div id="map">
  </div>
</body>
</html>
```



Отслеживание местоположения пользователя в режиме реального времени может пагубно сказаться на уровне заряда аккумулятора. Убедитесь, что вы предоставляете пользователям информацию об отслеживании их местоположения, а также средства контроля над этой процедурой.

Мы добавляем элемент `form` с двумя кнопками, одна из которых, именуемая `id` в виде `"watch"`, будет использоваться для запуска отслеживания, а вторая, с `id` в виде `"clearWatch"`, — для его остановки.

Снова воспользуемся нашими прежними элементами `<div>` для сообщения информации о местоположении в режиме реального времени

Чуть позже мы вернемся и позаботимся о карте Google...

Дорабатываем наш старый код...

Теперь нам необходимо добавить обработчики событий `click`, иницируемые при щелчке на кнопку, для двух наших кнопок. Мы будем добавлять их в функцию `getMyLocation` только при наличии поддержки API-интерфейса `Geolocation`. И поскольку мы собираемся полностью контролировать геолокационное отслеживание посредством двух кнопок, удалим существующий вызов `getCurrentPosition` из `getMyLocation`. Итак, удалим соответствующий код и добавим два обработчика: `watchLocation` — для кнопки, используемой для запуска отслеживания, и `clearWatch` — для кнопки, применяемой для остановки отслеживания:

```
function getMyLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(displayLocation, displayError);
    var watchButton = document.getElementById("watch");
    watchButton.onclick = watchLocation;
    var clearWatchButton = document.getElementById("clearWatch");
    clearWatchButton.onclick = clearWatch;
  }
  else {
    alert("Oops, no geolocation support");
  }
}
```

Если браузер поддерживает API-интерфейс `Geolocation`, то мы добавляем обработчики событий `click`. В их добавлении не будет смысла в случае отсутствия такой поддержки.

Будем вызывать `watchLocation` для запуска отслеживания, а `clearWatch` — для его остановки.

Написание обработчика `watchLocation`

Вот что мы попытаемся сделать: когда пользователь нажимает кнопку с `id` в виде "watch", он хочет печатать отслеживание своего местоположения. Поэтому мы воспользуемся методом `geolocation.watchPosition` для запуска отслеживания его местоположения. Метод `geolocation.watchPosition` обладает двумя параметрами — обработчиком успешного исполнения и обработчиком ошибок, так что мы сможем повторно использовать обработчики, которые у нас уже имеются. Он также возвращает `watchId`, что может использоваться в любой момент для отмены отслеживающего поведения. Мы припрячем `watchId` в глобальной переменной, которую используем, когда будем писать обработчик событий `click` для кнопки, применяемой для остановки отслеживания. Вот код для функции `watchLocation` и `watchId`, который вам нужно добавить в `myLoc.js`:

```
var watchId = null;
function watchLocation() {
  watchId = navigator.geolocation.watchPosition(displayLocation,
                                                displayError);
}
```

Добавьте `watchId` в верхнюю часть своего файла как глобальную переменную. Мы инициализируем ее значением `null`. Она потребуется нам позже для остановки отслеживания.

Вызываем метод `watchPosition`, передавая ему уже написанный обработчик успешного исполнения `displayLocation`, а также имеющийся у нас обработчик ошибок `displayError`.

Написание обработчика clearWatch

Теперь напишем обработчик для остановки отслеживания. Для этого нам потребуется взять `watchId` и передать его методу `geolocation.clearWatch`.

```
function clearWatch() {
  if (watchId) {
    navigator.geolocation.clearWatch(watchId);
    watchId = null;
  }
}
```

Убеждаемся, что у нас имеется `watchId`, а затем...

...вызываем метод `geolocation.clearWatch`, передав ему `watchId`. Это приведет к остановке отслеживания.

Еще необходимо внести небольшое обновление в `displayLocation`...

Есть еще одно небольшое обновление, которое нам нужно выполнить, и касается оно кода Google Maps, написанного ранее. В данном коде мы вызываем `showMap` для отображения карты Google (`showMap` генерирует новую карту на веб-странице). При этом нам необходимо, чтобы данная процедура осуществлялась только один раз. Однако, как вы можете помнить, при запуске отслеживания вашего местоположения посредством `watchPosition` обработчик `displayLocation` будет вызываться каждый раз, когда происходит обновление вашего местоположения.

Чтобы гарантировать, что вызов `showMap` состоится только один раз, сначала проверим, существует ли карта, и если она отсутствует, то вызовем `showMap`. В противном случае, если карта присутствует, это будет означать, что вызов функции `showMap` уже происходил, карта была сгенерирована, поэтому нам не нужно вызывать ее снова.

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

  var div = document.getElementById("location");
  div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
  div.innerHTML += " (with " + position.coords.accuracy + " meters accuracy)";

  var km = computeDistance(position.coords, ourCoords);
  var distance = document.getElementById("distance");
  distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";

  if (map == null) {
    showMap(position.coords);
  }
}
```

Если функция `showMap` еще не вызывалась, следует вызвать ее; в противном случае ее не нужно будет вызывать при каждом вызове `displayLocation`.

Пора отправляться в путь!



Убедитесь, что вы добавили весь новый код, и перезагрузите свою страницу `myLoc.html`. Теперь, для того чтобы по-настоящему протестировать данную веб-страницу, вам потребуется переместиться самим, чтобы обновить свое местоположение. Поэтому отправляйтесь на прогулку, прокатитесь на велосипеде или автомобиле либо воспользуйтесь любым другим видом транспорта.

Стоит ли говорить, что если вы будете тестировать данное приложение на своем настольном компьютере, то оно покажется довольно скучным (так как вы не сможете взять его с собой), поэтому вам следует проводить тестирование с использованием мобильного устройства. А если вам потребуется доступ со своего мобильного устройства к версии кода, размещенной онлайн, вы сможете найти ее по адресу <http://wickedlysmart.com/hfhtml5/chapter5/watchme/myLoc.html>.

Вот наш тестовый прогон...

Эти показатели будут обновляться по мере вашего перемещения.

Следует отметить, что на данный момент в центре карты будет отображаться ваше начальное местоположение...



Проведите тестирование онлайн: <http://wickedlysmart.com/hfhtml5/chapter5/watchme/myLoc.html>

Часть Задаваемые Вопросы

В: Как я могу регулировать частоту обновления браузером моего местоположения при использовании watchPosition?

О: Никак. Браузер сам определяет оптимальную частоту обновления и судит о том, когда именно произошло изменение вашего местоположения.

В: Почему показатели моего местоположения изменяются по несколько раз при первой загрузке страницы, даже несмотря на то что я сижу на месте не перемещаясь?

О: Помните, мы говорили, что браузер может применять разные методики для определения вашего местоположения? В зависимости от методики (или методик), используемой браузером для выяснения того, где вы находитесь, точность определения вашего местоположения со временем может изменяться. Как правило, точность улучшается, но иногда (например, если вы заехали в сельскую местность, где есть только одна вышка сотовой связи) может ухудшаться. Кроме того, вы всегда можете задействовать свойство accuracy в объекте position.coords, чтобы следить за точностью.

В: Могу ли я использовать свойства altitude и altitudeAccuracy объекта coordinates?

О: Нет гарантий, что эти свойства обязательно будут поддерживаться (и, очевидно, их поддержка будет обеспечиваться только на мобильных устройствах высшего класса), поэтому вам потребуется удостовериться в том, что ваш код сможет справиться с отсутствием их поддержки.

В: Что такое heading и speed?

О: heading — это направление, в котором вы двигаетесь, а speed — скорость, показывающая, насколько быстро вы это делаете. Представьте, что вы едете в автомобиле на север по автомагистрали, а спидометр показывает 90 км/ч. Ваше направление в данном случае — на север, а скорость равна 90 км/ч. Если же вы сидите в машине, стоящей на парковке, то ваша скорость равна 0, а направления у вас нет (поскольку вы не двигаетесь).

В: Когда я проверяю по карте расстояние от своего местоположения до вашего, у меня получается намного больший результат, чем рапортует приложение. В чем причина?

О: Как вы можете помнить, функция computeDistance вычисляет расстояние по прямой. Ваш картографический инструмент, скорее всего, выдает расстояние, которое придется проехать на автомобиле по извилистому пути.

Возьми в руку карандаш



Ниже приведена альтернативная реализация для displayLocation. Сможете ли вы догадаться, что она делает? Взгляните на нее и напишите свой ответ в самом низу. Если в вас проснулась предприимчивость, то затем протестируйте данный код!

```
distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
if (km < 0.1) {
  distance.innerHTML = "You're on fire!";
} else {
  if (prevKm < km) {
    distance.innerHTML = "You're getting hotter!";
  } else {
    distance.innerHTML = "You're getting colder...";
  }
}
prevKm = km;
```

Напишите здесь, что, по-вашему,
делает весь этот код. →

Параметр positionOptions...

До постоянного момента мы не затрагивали третий параметр `getCurrentPosition` (и `watchPosition`) — `positionOptions`. С помощью данного параметра мы можем контролировать, как API-интерфейс Geolocation вычисляет соответствующие показатели. Давайте взглянем на три параметра вместе с их значениями по умолчанию.

```
var positionOptions = {
  enableHighAccuracy: false,
  timeout: Infinity,
  maximumAge: 0
}
```

И наконец, с помощью параметра `maximumAge` устанавливаем максимальный «возраст» данных о местоположении, который они могут иметь до того, как браузеру потребуется заново определить местоположение. По умолчанию данный параметр имеет значение 0, которое подразумевает, что браузеру придется всегда заново определять местоположение пользователя (при каждом вызове `getCurrentPosition`).

Сначала идет свойство, которое активирует высокую точность; что именно под этим подразумевается, мы поговорим через несколько мгновений...

Параметр `timeout` позволяет контролировать, как долго браузер сможет определять местоположение пользователя. По умолчанию для данного параметра задается значение `Infinity`, то есть браузер получит столько времени, сколько ему потребуется.

Для данного параметра также можно задать значение в миллисекундах (например, `10 000`), то есть браузер получит 10 секунд на определение местоположения, в противном случае будет вызван обработчик ошибок.

Нельзя ли снова поговорить о точности?

Вы уже видели, что все данные о местоположении, возвращаемые нам API-интерфейсом Geolocation, включают свойство `accuracy`. Однако мы также можем сказать API-интерфейсу Geolocation, что хотим получать только наиболее точные результаты, которые он способен выдать. Но это будет лишь памяк браузеру, а различные реализации в действительности могут по-разному относиться к такому памяку. Несмотря на то что данный вариант может показаться мало важным, он несет в себе массу подтекста. Например, если вас не интересует суперточность результатов определения местоположения — вам может быть вполне достаточно знать, что ваш пользователь находится в Балтиморе, — то API-интерфейс Geolocation сможет очень быстро и дешево (в плане энергопотребления) сообщить вам эти сведения. С другой стороны, если вы захотите узнать улицу, на которой находится ваш пользователь, то нет проблем, однако в таком случае API-интерфейсу Geolocation придется задействовать GPS и использовать массу электроэнергии для извлечения данной информации. Посредством параметра `enableHighAccuracy` вы говорите API-интерфейсу Geolocation, что вам необходимы наиболее точные результаты определения местоположения, какие он только способен дать, нусть даже и высокой ценой. Имейте в виду: использование данного параметра не гарантирует того, что браузеру обязательно удастся выдать вам более точное местоположение.



Мир параметров `timeout` и `maximumAge`...

Еще раз взглянем па то, что представляют собой параметры `timeout` и `maximumAge`.

timeout: сообщает браузеру, как **долго** он сможет определять местоположение пользователя. Следует отметить, что если пользователю будет предложено одобрить запрос па определение местоположения, то отсчет времени ожидания не пачнется до тех пор, пока он не даст своего согласия. Если браузеру не удастся определить повое местоположение в течение времени в миллисекундах, указанного в значении `timeout`, произойдет вызов обработчика ошибок. *Параметр `timeout` по умолчанию имеет значение `Infinity`.*

maximumAge: сообщает браузеру, насколько **устаревшими** могут быть данные о местоположении. Таким образом, если у браузера имеются данные о местоположении, которое было определено 60 секунд назад, а для `maximumAge` при этом задано значение 90000 (90 секунд), то вызов `getCurrentPosition` приведет к возврату уже имеющихся кэшированных данных о местоположении (то есть браузер не станет пытаться определить повое местоположение пользователя). Однако если в данной ситуации для `maximumAge` будет задано значение, равное 30 секундам, браузеру придется определять повое местоположение пользователя.



Таким образом, посредством `maximumAge` я могу устанавливать, насколько часто браузер будет пересчитывать или определять мое местоположение. Я понимаю, что это позволит сделать мое приложение более быстрым и энергоэффективным. А как насчет `timeout`? Как я могу использовать данный параметр, чтобы сделать код лучше?



Вы правы в своих суждениях насчет `maximumAge`. Что касается `timeout`, следует рассуждать так: при использовании `maximumAge` вы получаете старые (кэшированные) данные, пока они остаются «моложе» значения, заданного для `maximumAge`, и это действительно помогает оптимизировать производительность вашего приложения. Но что будет, когда «возраст» данных о местоположении превысит значение, заданное для `maximumAge`? В таком случае браузер попытается извлечь данные о новом местоположении. Быть может, все это не будет слишком вас беспокоить, например, вас устроят данные о новом местоположении, если они есть у браузера, а если же их у него нет, то эти сведения не потребуются сию же минуту. Вы могли бы задать для `timeout` значение 0, и если у браузера будут данные о местоположении, которые проходят «тест» `maximumAge`, то все отлично, в противном случае соответствующий вызов незамедлительно потерпит неудачу и произойдет вызов обработчика ошибок (с кодом ошибки `TIMEOUT`). Это лишь пример творческого подхода к использованию `maximumAge` и `timeout` для пастройки поведения вашего приложения.

* КТО И ЧТО ДЕЛАЕТ? *

Ниже приведены параметры, касающиеся API-интерфейса Geolocation. Ваша задача заключается в том, чтобы подобрать каждому параметру пару в виде соответствующего ему поведения.

```
{maximumAge: 600000}
```

Мне требуются только кэшированные данные о местоположении, которым меньше 10 минут. При отсутствии таких данных я хочу получить данные о новом местоположении, но только в том случае, если на это уйдет 1 секунда или меньше.

```
{timeout: 1000, maximumAge: 600000}
```

Я буду использовать кэшированные данные о местоположении, если таковые будут иметься у браузера, и при этом им будет меньше 10 минут; в противном случае я хочу получить данные о новом местоположении.

```
{timeout: 0, maximumAge: Infinity}
```

Мне требуются только данные о новом местоположении. Браузер может использовать столько времени на определение местоположения, сколько ему потребуется.

```
{timeout: Infinity, maximumAge: 0}
```

Мне требуются только кэшированные данные о местоположении. Меня устраивают данные любого «возраста». Если кэшированные данные о местоположении будут полностью отсутствовать, произойдет вызов обработчика ошибок. Сведения о новом местоположении извлекаться не будут! Данные пастройки предпазачепы для использования в ситуациях, когда речь идет об автопомпой работе.

Как задавать параметры

Одна из замечательных особенностей JavaScript заключается в том, что если вам потребуется задать целый набор параметров в объекте, мы сможем сделать это, просто вставив литеральный объект прямо посередине вызова метода. Допустим, вам необходимо активировать высокую точность, а также задать максимальный «возраст» для данных о местоположении, равный 60 секундам (60 000 миллисекунд). Переменная `options` создается следующим образом:

```
var options = {enableHighAccuracy: true, maximumAge: 60000};
```

Вы уже начали замечать, что JavaScript — это по-настоящему круто? По крайней мере, мы считаем именно так. 😊

Затем мы можем передать `options` либо `getCurrentPosition`, либо `watchPosition`, как показано далее:

```
navigator.geolocation.getCurrentPosition(
    displayLocation,
    displayError,
    options);
```

Передаем наши параметры, используя переменную `options`.

Наши параметры, добавленные в виде литерального объекта прямо в вызов функции! Некоторые считают, что так будет проще, поскольку код получается более удобным.

Или мы могли бы просто добавить соответствующий объект как встроенный:

```
navigator.geolocation.getCurrentPosition(
    displayLocation,
    displayError,
    {enableHighAccuracy: true, maximumAge: 60000});
```

Вы будете часто сталкиваться с использованием такой методики в JavaScript-коде.

Теперь, когда вы знаете, что представляют собой данные параметры, что они делают и как их задавать, можно переходить к их использованию. Этим мы и займемся далее, однако не забывайте, что они предназначены для настройки *вашего* приложения, которое может предъявлять свои специфические требования. На эти параметры также влияет используемое вами устройство, реализация браузера и сеть, поэтому полным их исследованием вам придется заняться самостоятельно.

Диагностический тест-драйв



Ранее, при выполнении диагностических тест-драйвов, сталкивались ли вы с ситуациями, когда вы все ждали и ждали, но ничего не происходило? Причина, скорее всего, заключалась в том, что для `timeout` было задано значение `Infinity`. Другими словами, браузер будет бесконечно долго пытаться определить местоположение, пока не столкнется с условием возникновения ошибок. Что ж, теперь вы знаете, как исправить это, поскольку можете заставить API-интерфейс `Geolocation` вести себя более рационально в подобных ситуациях, задав соответствующее значение для `timeout`. Вот как это делается:

```
function watchLocation() {
    watchId = navigator.geolocation.watchPosition(
        displayLocation,
        displayError,
        {timeout: 5000});
}
```

Задав для `timeout` значение 5000 миллисекунд (5 секунд), вы сделаете так, что браузер не будет бесконечно долго пытаться определить местоположение.

Проведите тестирование, подставляя сюда разные значения.

~~НЕ~~ ПЫТАЙТЕСЬ СДЕЛАТЬ ЭТО ДОМА (КАК ЗАСТАВИТЬ GEOLOCATION РАБОТАТЬ НА ПРЕДЕЛЕ ВОЗМОЖНОСТЕЙ)

Разве не интересно узнать, как быстро браузер сможет определить ваше местоположение? Мы усложним для него задачу, насколько это представляется возможным:

- активируем высокую точность;
- не позволим ему использовать кэшированные данные о местоположении (задав для `maximumAge` значение 0);
- ограничим его по времени, задав для `timeout` значение 100, которое будем увеличивать после каждой неудачной попытки браузера определить местоположение в течение заданного времени.

Предупреждение: мы не знаем, как долго продержится аккумулятор того или иного устройства при определении местоположения на основе таких настроек, поэтому используйте их на свой страх и риск!

Вот как будут выглядеть наши исходные параметры:

```
{enableHighAccuracy: true, timeout: 100, maximumAge: 0}
```

```
{enableHighAccuracy: true, timeout: 200, maximumAge: 0}
```

```
{enableHighAccuracy: true, timeout: 300, maximumAge: 0}
```

← Мы начнем с этого...

← и если браузеру не удастся определить местоположение, дадим ему больше времени...

← и так далее...

Теперь ознакомьтесь с кодом, приведенным на следующей странице, — он покажется вам довольно интересным. Наберите его (можете добавить его в свой JavaScript-файл `myLoc.js`). Протестируйте данный код на различных устройствах, которые у вас имеются, а результаты запишите здесь:

здесь пометьте устройство

А здесь укажите время



На _____ местоположение было определено за _____ миллисекунд

Тестирование онлайн: <http://wickedlysmart.com/hfhtml5/chapter5/speedtest/speedtest.html>

```

var options = { enableHighAccuracy: true, timeout:100, maximumAge: 0 };
window.onload = getMyLocation;
function getMyLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(
      displayLocation,
      displayError,
      options);
  } else {
    alert("Oops, no geolocation support");
  }
}

```

Сначала мы инициализируем `options`, при этом `timeout` будет иметь значение 100, а `maximumAge` — 0.

Здесь поступаем как обычно: задействуем `displayLocation` и `displayError` в качестве соответственно обработчика успешного исполнения и обработчика ошибок — и передаем `options` в роли третьего параметра.

```

function displayError(error) {
  var errorTypes = {
    0: "Unknown error",
    1: "Permission denied",
    2: "Position is not available",
    3: "Request timeout"
  };
  var errorMessage = errorTypes[error.code];
  if (error.code == 0 || error.code == 2) {
    errorMessage = errorMessage + " " + error.message;
  }
  var div = document.getElementById("location");
  div.innerHTML = errorMessage;
  options.timeout += 100;
  navigator.geolocation.getCurrentPosition(
    displayLocation,
    displayError,
    options);
  div.innerHTML += " ... checking again with timeout=" + options.timeout;
}
function displayLocation(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  var div = document.getElementById("location");
  div.innerHTML = "You are at Latitude: " + latitude +
    ", Longitude: " + longitude;
  div.innerHTML += " (found in " + options.timeout + " milliseconds)";
}

```

Первым делом разбираемся с обработчиком ошибок.

Этот код будет таким же, как и прежде...

В случае, если браузеру не удастся определить местоположение, будем увеличивать значение `timeout` на 100 миллисекунд и давать ему повторную попытку. Мы также уведомим пользователя о повторении попытки.

```

function displayLocation(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  var div = document.getElementById("location");
  div.innerHTML = "You are at Latitude: " + latitude +
    ", Longitude: " + longitude;
  div.innerHTML += " (found in " + options.timeout + " milliseconds)";
}

```

Когда браузеру удастся определить местоположение пользователя, мы сообщим ему, сколько времени это заняло.

Шлифуем наше приложение!

Если откинуться па спинку стула и призадуматься, то мы с вами, используя пемного HTML и JavaScript, создали веб-приложение, которое способно не только определять ваше местоположение, но также отслеживать и отображать его почти в режиме реального времени. Да, HTML и вправду серьезно попрос (как и ваши павыки!).

Однако если вести речь о дадном приложении, не кажется ли вам, что оно нуждается в пембольшой шлифовке с целью придания ему завершепного вида? Например, мы могли бы сделать так, чтобы ваше местоположение отображалось на карте по ходу перемещения; кроме того, мы могли бы пойти еще дальше и показывать места, где вы были, то есть на карте будет отображаться ваш путь.

Напишем функцию для того, чтобы ваше местоположение продолжало отображаться в центре карты по мере вашего передвижения. Будем добавлять повый маркер, отмечая каждое повое местоположение.

Итак, будем вызывать данную функцию и передавать ей соответствующие координаты.

```
function scrollMapToPosition(coords) {
    var latitude = coords.latitude;
    var longitude = coords.longitude;
    var latlong = new google.maps.LatLng(latitude, longitude);

    map.panTo(latlong);

    addMarker(map, latlong, "Your new location", "You moved to: " +
        latitude + ", " + longitude);
}
```

И наконец, добавляем маркер для отметки вашего нового местоположения, используя функцию `addMarker`, написанную нами ранее, которой передаем `map`, объект `LatLng`, заголовок и содержимое для нового маркера.



Интеграция нашей новой функции

Нам останется лишь обновлять функцию `displayLocation` для вызова `scrollMapToPosition` каждый раз, когда изменяется ваше местоположение. Не забывайте, что при первом вызове `displayLocation` мы вызываем `showMap` для генерирования карты и отображения маркера, отмечающего ваше начальное местоположение. Каждый раз после этого нам потребуется вызывать `scrollMapToPosition` для добавления нового маркера и центрирования карты запово. Вот как будет выглядеть обновленный код:

```
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude
        + ", Longitude: " + longitude;
    div.innerHTML += " (with " + position.coords.accuracy + " meters accuracy)";
    var km = computeDistance(position.coords, ourCoords);
    var distance = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";

    if (map == null) {
        showMap(position.coords);
    } else {
        scrollMapToPosition(position.coords);
    }
}
```

← При первом вызове `displayLocation` нам необходимо сгенерировать и отобразить карту, а также добавить первый маркер.

← После этого останется лишь добавлять новые маркеры на имеющуюся карту.

И еще раз...



Перезагрузите свою страницу и начните движение... Ну как, вычерчивается на карте ваш нуть? Вы должны увидеть дорожку из маркеров, добавляемых на карту по мере вашего передвижения (если, конечно, вы не сидите за пастольным компьютером!).

Таким образом, данное приложение мы представляем как твердое доказательство того, что «куда бы ты ни шел — ты имеппо там» (в нашем случае под этой философской фразой понимается, что вас можно будет пайти).

→ Дорожка из маркеров, показывающая наш недавний путь от штаб-квартиры Wickedly Smart до тайного подземного логова... Ой, зря мы проболтались об этом...



Тестирование онлайн: <http://wickedlysmart.com/hfhtml5/chapter5/watchmepan/myLoc.html>



Развлечения с Магнитами

Завершая эту главу, мы предполагаем, что вы можете захотеть еще больше отшлифовать рассмотренное выше приложение. Вы обратили внимание (в силу тех или иных обстоятельств) на то, что на карту добавляется слишком много маркеров, когда ведется отслеживание вашего местоположения?

Это происходит потому, что `watchPosition` слишком часто детектирует перемещение, что приводит к вызову обработчика успешного исполнения `displayLocation` каждый раз, когда вы успеаете сделать лишь несколько шагов. Один из способов исправить это заключается в добавлении кода, благодаря которому новый маркер будет создаваться только после того, как вы пройдете более значительное расстояние (например, 20 метров в целях тестирования).

У нас уже имеется функция, которая вычисляет расстояние между двумя точками (`computeDistance`), поэтому нам останется лишь сделать так, чтобы ваше местоположение сохранялось при каждом вызове `displayLocation` и проводилась проверка того, является ли расстояние между вашим предыдущим местоположением и новым больше 20 метров, прежде чем произойдет вызов `scrollMapToPosition`. Необходимый для этого код приведен ниже; ваша задача заключается в том, чтобы заполнить имеющиеся в нем пробелы. Будьте внимательны, поскольку некоторые магнитные таблички нужно использовать более одного раза!



```

var _____;
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
    div.innerHTML += " (with " + position.coords.accuracy + " meters accuracy)";
    var km = computeDistance(position.coords, ourCoords);
    var distance = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
    if (map == null) {
        showMap(position.coords);
        prevCoords = _____;
    } else {
        var meters = _____(position.coords, prevCoords) * 1000;
        if (_____ > _____) {
            scrollMapToPosition(position.coords);
            _____ = _____;
        }
    }
}
}

```

Magical floating boxes containing code snippets:

- `computeDistance`
- `meters`
- `prevCoords = null;`
- `position.coords`
- `20`
- `prevCoords`

КЛЮЧЕВЫЕ МОМЕНТЫ

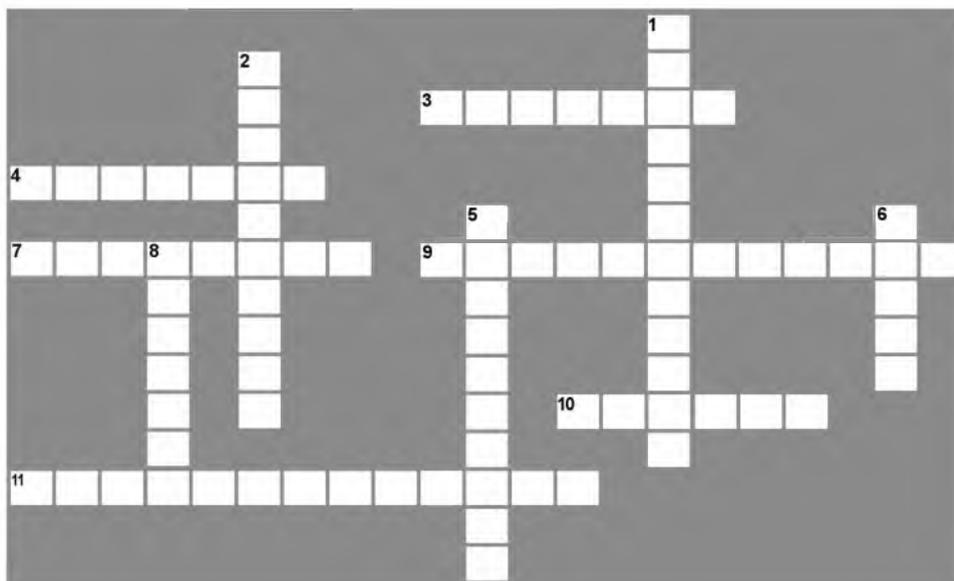


- API-интерфейс Geolocation не является «официальной» частью спецификации HTML5, но считается частью семейства технологий HTML5.
- Можно использовать разные методики определения своего местоположения в зависимости от устройства, которое у вас имеется.
- GPS позволяет получать более точные данные о местоположении, чем триангуляция с использованием вышек сотовой связи и мобильного телефона или методика на основе применения IP-адресов.
- В случае с мобильными устройствами, не поддерживающими GPS, для определения местоположения можно прибегнуть к триангуляции с использованием вышек сотовой связи и мобильного телефона.
- API-интерфейс Geolocation включает три метода и несколько свойств.
- Основным методом в API-интерфейсе Geolocation является `getCurrentPosition` — метод объекта `navigator.geolocation`.
- У `getCurrentPosition` имеется один обязательный параметр — обработчик успешного исполнения, и два опциональных — обработчик ошибок и `options`.
- Объект `position` передается обработчику успешного исполнения наряду с информацией о местоположении пользователя, включая широту и долготу.
- Объект `position` содержит свойство `coords`, которое является объектом `coordinates`.
- Объект `coordinates` обладает свойствами, включая `latitude`, `longitude` и `accuracy`.
- Некоторые устройства могут поддерживать и другие свойства объекта `coordinates`: `altitude`, `altitudeAccuracy`, `heading` и `speed`.
- Свойство `accuracy` используется для определения точности местоположения в метрах.
- При вызове `getCurrentPosition` ваш браузер должен удостовериться, что вы дали разрешение на использование информации о вашем местоположении.
- `watchPosition` — это метод объекта `geolocation`, который отслеживает ваше местоположение и вызывает обработчик успешного исполнения при изменении вашего местоположения.
- Как и у `getCurrentPosition`, у `watchPosition` имеется один обязательный параметр — обработчик успешного исполнения и два опциональных — обработчик ошибок и `options`.
- `clearWatch` используется для остановки отслеживания местоположения пользователя.
- При использовании `watchPosition` энергопотребление устройства возрастает, поэтому его аккумулятор будет разряжаться быстрее.
- Третий параметр методов `getCurrentPosition` и `watchPosition` с именем `options` представляет собой объект со свойствами, которые вы задаете с целью контроля над поведением API-интерфейса Geolocation.
- Свойство `maximumAge` устанавливает, будет ли `getCurrentPosition` использовать кэшированные данные о местоположении, и если да, то насколько устаревшими смогут быть эти данные, прежде чем потребуется определять новое местоположение.
- Свойство `timeout` устанавливает, сколько времени будет у `getCurrentPosition` на определение нового местоположения, прежде чем произойдет вызов обработчика ошибок.
- Свойство `enableHighAccuracy` намекает устройствам на то, что им следует затрачивать больше усилий на определение точного местоположения, если это представляется возможным.
- Вы можете использовать API-интерфейс Geolocation в сочетании с API-интерфейсом Google Maps для отображения своего местоположения на карте.



HTML5-кроссворд

Вы проделали довольно длинный путь в этой главе, используя свой первый API-интерфейс JavaScript. Закрепите изученный материал, решив данный кроссворд.



По горизонтали

3. Измерение долготы ведется от _____, Англия.
4. Точность определения местоположения имеет определенный подтекст в случае с веб-приложениями, который заключается в том, что она может сказываться на уровне заряда аккумуляторной _____ устройства.
7. Не давайте кому-либо рекомендаций насчет направления движения, если ваши координаты не будут обладать достаточной _____.
9. Фраза «Куда бы ты ни шел — ты уже там» упоминалась в фильме _____.
10. Если вы откажете браузеру в его запросе на разрешение использовать данные о вашем местоположении, это приведет к вызову обработчика ошибок с кодом _____ в виде 1.
11. Тайное местоположение штаб-квартиры _____ имеет координаты 47.62485, -122.52099.

По вертикали

1. В случае применения устройств без поддержки GPS определить ваше местоположение можно будет посредством _____ с использованием вышек сотовой связи.
2. Для вычисления расстояния между двумя точками координат можно использовать формулу _____.
5. Вы никогда не получите кэшированных данных о местоположении, если для параметра _____ задано значение 0.
6. Центрирование карты заново осуществляется с использованием метода _____.
8. Место, где располагается город _____, имеет широту и долготу соответственно 40.77, -73.98.



Развлечения с магнитами. Решение

Ваша задача заключается в том, чтобы заполнить пробелы в приведенном ниже коде, благодаря которому новый маркер будет отображаться только в том случае, если вы пройдете более 20 метров от места, отмеченного предыдущим маркером. Для заполнения пробелов в коде используйте соответствующие магнитные таблички. Будьте внимательны, поскольку некоторые из них придется использовать более одного раза! Приведем решение этого задания.

```

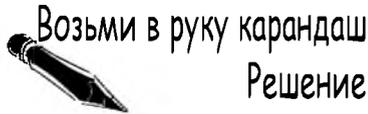
var prevCoords = null;
function displayLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var div = document.getElementById("location");
    div.innerHTML = "You are at Latitude: " + latitude + ", Longitude: " + longitude;
    div.innerHTML += " (with " + position.coords.accuracy + " meters accuracy)";
    var km = computeDistance(position.coords, ourCoords);
    var distance = document.getElementById("distance");
    distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
    if (map == null) {
        showMap(position.coords);
        prevCoords = position.coords;
    }
    else {
        var meters = computeDistance (position.coords, prevCoords) * 1000;
        if (meters > 20) {
            scrollMapToPosition(position.coords);
            prevCoords = position.coords;
        }
    }
}

```

Так намного лучше!



Проведите тестирование онлайн:
<http://wickedlysmart.com/hfhtml5/chapter6/final/myLoc.html>



Решение

Ниже приведена альтернативная реализация для `displayLocation`. Сможете ли вы догадаться, что она делает? Взгляните на нее и напишите свой ответ в самом низу. Если в вас проснулась предприимчивость, протестируйте данный код!

```
distance.innerHTML = "You are " + km + " km from the WickedlySmart HQ";
if (km < 0.1) {
    distance.innerHTML = "You're on fire!";
} else {
    if (prevKm < km) {
        distance.innerHTML = "You're getting hotter!";
    } else {
        distance.innerHTML = "You're getting colder...";
    }
}
prevKm = km;
```

Данный код превращает наше приложение в игру «Горячо/холодно». Он выводит сообщение "You're getting hotter!" ("Теплее!"), если вы подходите ближе к штаб-квартире Wickedly Smart, либо "You're getting colder!" ("Холоднее!"), если вы уходите все дальше от нее. Когда вы окажетесь в 0,1 километра от штаб-квартиры Wickedly Smart, появится сообщение "You're on fire!" ("Очень горячо!")

Напишитесь здесь, что, по-вашему, делает весь этот код.



Мы протестировали данный код, и вот что у нас получилось!!



* КТО И ЧТО ДЕЛАЕТ? *

РЕШЕНИЕ

Ниже приведены параметры, касающиеся API-интерфейса Geolocation. Ваша задача заключается в том, чтобы подобрать каждому параметру пару в виде соответствующего ему поведения.

`{maximumAge:600000}`

Мне требуются только кэшированные данные о местоположении, которым меньше 10 минут. При отсутствии таких данных я хочу получить данные о новом местоположении, но только в том случае, если на это уйдет 1 секунда или меньше.

`{timeout:1000, maximumAge:600000}`

Я буду использовать кэшированные данные о местоположении, если таковые будут иметься у браузера, и при этом им будет меньше 10 минут; в противном случае я хочу получить данные о новом местоположении.

`{timeout:0, maximumAge:Infinity}`

Мне требуются только данные о новом местоположении. Браузер может использовать столько времени на определение местоположения, сколько ему потребуется.

`{timeout:Infinity, maximumAge:0}`

Мне требуются только кэшированные данные о местоположении. Меня устраивают данные любого «возраста». Если кэшированные данные о местоположении будут полностью отсутствовать, произойдет вызов обработчика ошибок. Сведения о новом местоположении извлекаться не будут! Данные пастройки предпазачепы для использования в ситуациях, когда речь идет об автопомпой работе.



HTML5-кроссворд. Решение



Приложения-экстраверты

Если бы я только знала, что обращение к веб-службам и взаимодействие с ними может быть настолько увлекательным...



Что-то вы слишком засиделись на своей странице. Настало время пообщаться с веб-службами с целью сбора данных и последующего возврата этой информации, что позволяет создавать более эффективные веб-ресурсы, которые объединяют собираемые данные. Это важный момент в написании современных HTML5-приложений, а чтобы успешно этим заниматься, вам необходимо *знать*, как происходит общение с веб-службами. Кроме того, вы научитесь внедрять данные от веб-служб в свои страницы. Усвоив изложенный здесь материал, вы сможете взаимодействовать с любой веб-службой. Мы даже расскажем вам о новомодном жаргоне, которым следует пользоваться при общении с веб-службами. Вы познакомитесь с некоторыми новыми API-интерфейсами — так называемыми *коммуникационными API-интерфейсами*.

Компанию Mighty Gumball требуется веб-приложение

Свежая новость: ипповационная компания Mighty Gumball, Inc., которая производит и устанавливает настоящие автоматы для продажи жевательной резинки в виде шариков, связалась с нами и попросила помочь. Если вы еще не знаете, то эта компания недавно стала оснащать свои торговые автоматы возможностью подключения к Интернету, чтобы отслеживать уровни продаж почти в режиме реального времени.

Стоит ли говорить, что в Mighty Gumball работают те, кто знает толк в автоматах для продажи жевательной резинки, а не разработчики программного обеспечения, поэтому они и обратились к нам за помощью и попросили написать приложение, которое позволит отслеживать уровни продаж данной жевательной резинки.

Вот что они нам прислали:

Ознакомьтесь с нашей новой моделью автомата для продажи жевательной резинки MG2200, поддерживающей подключение к Интернету. Он произведет настоящую революцию в бизнесе.

Вы можете помнить эту компанию по нашей книге «Изучаем шаблоны проектирования» (Head First Design Patterns), где мы помогли им разрабатывать код на стороне сервера.

Исполнительный директор
Mighty Gumball



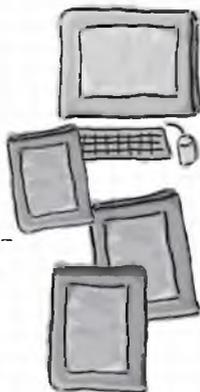
Там, где автомат для продажи жевательной резинки никогда не пуст наполовину

Благодарим, что согласились помочь! Вот как, на наш взгляд, должен работать инструмент для отслеживания продаж жевательной резинки из автоматов, и мы надеемся, что вы сможете его для нас реализовать! Обращайтесь, если у вас возникнут какие-либо вопросы!

Кроме того, мы вскоре вышлем вам спецификации, касающиеся веб-службы.

Инженеры компании Mighty Gumball

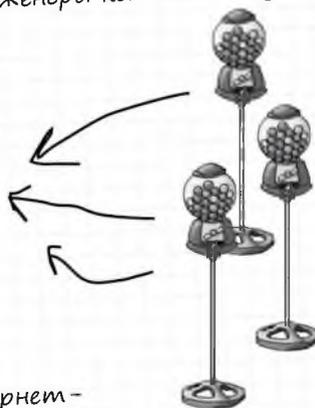
Мобильные и настольные устройства будут получать сведения о продажах в реальное время с помощью веб-службы.



Нам необходимо, чтобы вы написали эту часть, естественно, используя HTML5!!



Наш интернет-сервер



Все наши автоматы для продажи жевательной резинки будут отправлять отчеты на центральный сервер.

Прежде чем мы приступим, потратьте немного времени и подумайте, как бы вы подошли к разработке приложения, которое извлекает данные, предоставляемые веб-службой, и использует их для обновления веб-страницы. Не беспокойтесь, если вы еще не знаете, как извлекать данные, просто обдумайте высокоуровневый дизайн приложения. Сделайте наброски, пометки, напишите псевдокод для любого кода, который вам может потребоваться. Считайте это разминкой для мозга...



Mighty Gumball, Inc.

Там, где автомат для
продажи жевательной
резинки никогда не пуст
наполовину

Заметки по разработке

Как мы будем извлекать
данные, предоставляемые
веб-службой, и направлять их
к нашей веб-странице?

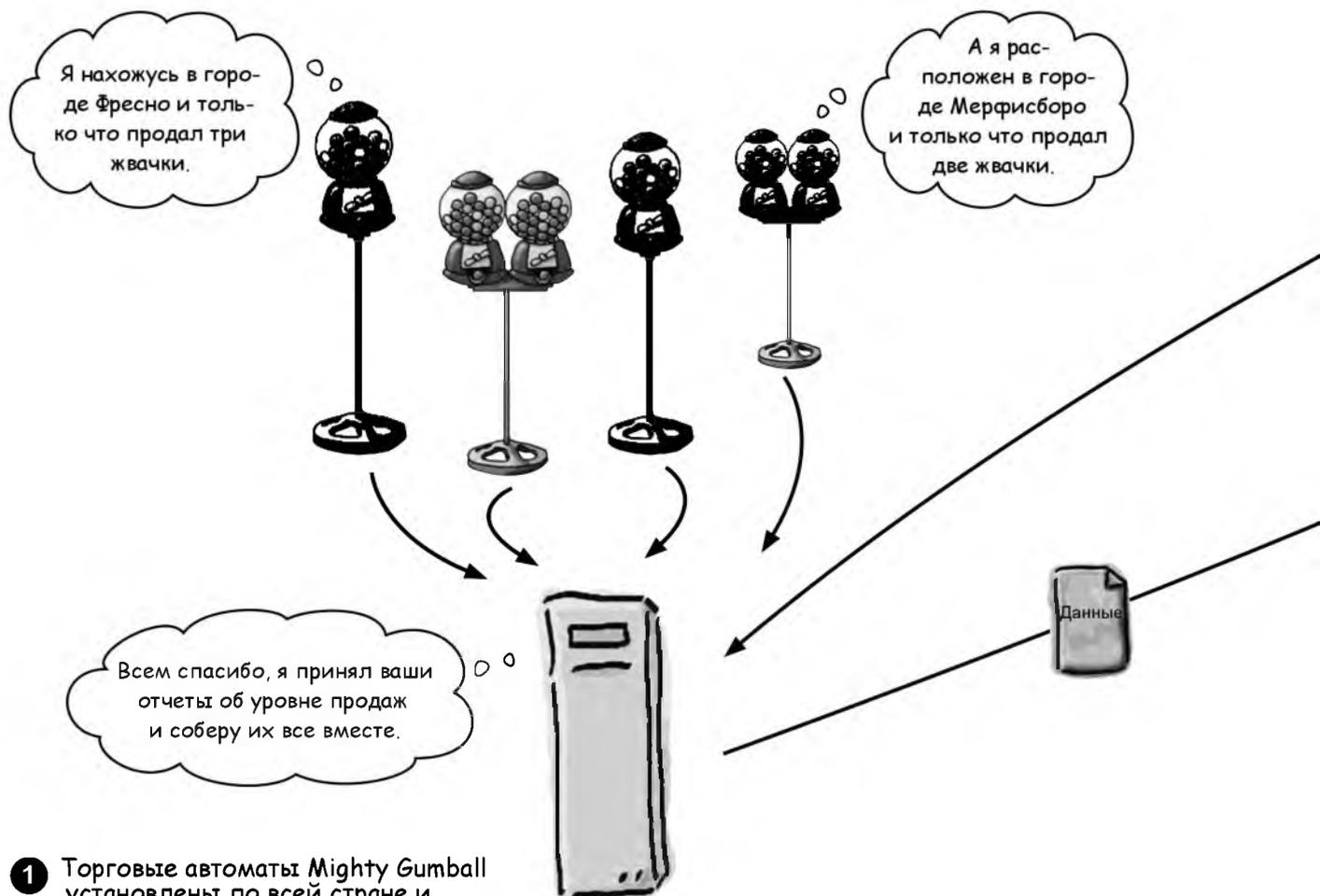
Как мы будем обновлять
страницу после того,
как извлечем данные?

Какие проблемы у нас могут
возникнуть с получением дан-
ных от удаленного сервера?



Поглубнее о системе Mighty Gumball

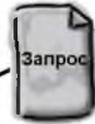
Вероятно, вам необходимо более подробно разобраться во всей этой системе, чем было описано в краткой записке от инженеров из Mighty Gumball. Итак, вот как обстоит дело: во-первых, у данной компании есть автоматы для продажи жевательной резинки, которые располагаются по всей стране и отправляют отчеты на сервер Mighty Gumball, который, в свою очередь, собирает все эти отчеты и делает их доступными с помощью веб-службы. Во-вторых, нас просят создать веб-приложение, отображающее показатели продаж в браузере для группы сбыта Mighty Gumball. Кроме того, они, скорее всего, захотят, чтобы данная отчетность обновлялась по мере изменения уровня продаж по прошествии времени. Вот как все это будет выглядеть паглядно в общих чертах:



- 1 Торговые автоматы Mighty Gumball установлены по всей стране и отправляют отчеты об уровне продаж на центральный сервер данной компании. Затем центральный сервер собирает все отчеты вместе и делает их доступными с помощью веб-службы.

Сервер Mighty Gumball

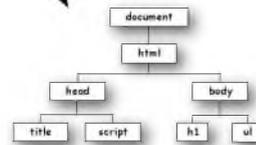
2 Браузер загружает веб-приложение Mighty Gumball, включая HTML-разметку, CSS и JavaScript.



3 Приложение отправляет веб-запрос с целью извлечения собранных вместе данных об уровне продаж с сервера Mighty Gumball.



5 Приложение изучает данные, после чего обновляет объектную модель документа (DOM) страницы для того, чтобы отразить новые данные об уровне продаж.



1 Обрато приложение получает данные с сервера Mighty Gumball.

7 Приложение возвращается к шагу 3 и постоянно запрашивает новые данные. В результате этого страница обновляется почти в режиме реального времени.



6 Браузер обновляет страницу на основе DOM, в результате чего пользователи могут увидеть результаты.

Приступаем к созданию приложения...

Пока мы ждем спецификаций от инженеров из Mighty Gumball, займемся HTML-разметкой.

Наверняка вы уже сообразили, что нам не понадобится большой объем HTML-разметки, чтобы успешно начать создавать наше веб-приложение. Все, что нам требуется, — это место для размещения отчетов об уровне продаж по мере их поступления, а все остальное мы доверим сделать JavaScript. Наберите приведенную ниже разметку, после чего мы взглянем на то, как извлекать данные с веб-сервера по протоколу HTTP.

```
<!doctype html>
<html lang="en">
<head>
<title>Mighty Gumball (JSON)</title>
<meta charset="utf-8">
<script src="mightygumball.js"></script>
<link rel="stylesheet" href="mightygumball.css">
</head>
<body>
<h1>Mighty Gumball Sales</h1>
<div id="sales">

</div>
</body>
</html>
```

Используем стандартные HTML5-элементы `<head>` и `<body>`.

Заранее размещаем ссылку на JavaScript-файл, зная, что вскоре напишем соответствующий JavaScript-код!

Также задействуем CSS для стилизации отчета о продажах Mighty Gumball, чтобы его внешний вид понравился исполнительному директору.

Сюда мы будем помещать данные об уровне продаж. Каждый элемент данных об уровне продаж будет добавляться сюда как `<div>`.

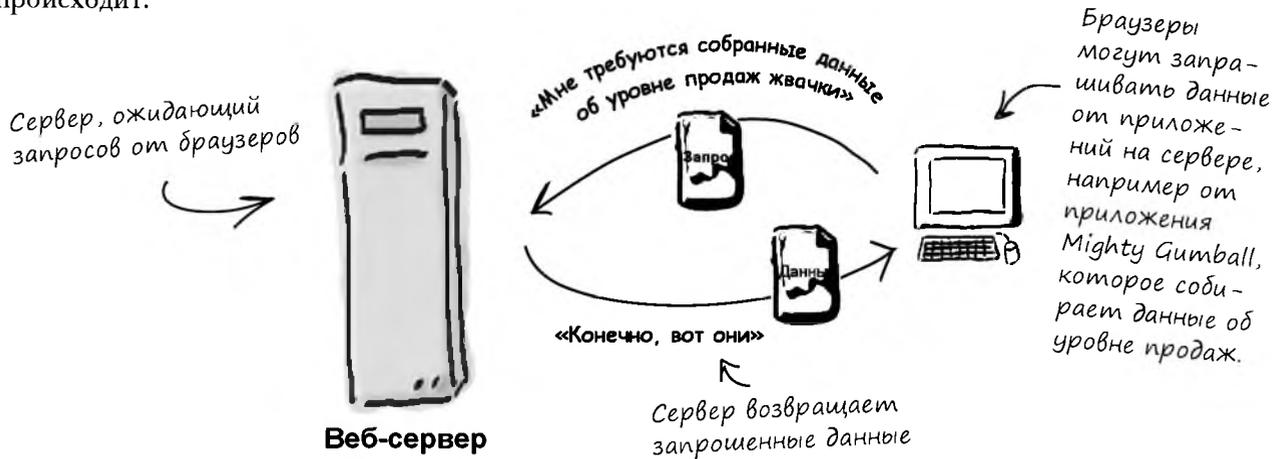
Загрузите движатель для тест-драйва



Набрав приведенный выше код, загрузите его в своем любимом браузере и протестируйте, прежде чем двигаться дальше. И не забывайте, что загрузить CSS (и прочий код, использованный в этой главе) вы сможете со страницы <http://wickedlysmart.com/hfhtml5>.

Как выполняются запросы, адресованные веб-службам?

Давайте пемпого приторможим... Вы уже знаете, как браузер запрашивает страницу с веб-сервера — он отправляет HTTP-запрос серверу, а тот возвращает соответствующую страницу паряду с дополнительными метадаппными, которые (обычно) «видит» только браузер. Однако, возможно, вы не знаете, что браузер способен апалогичным образом *извлекать данные* с веб-сервера по протоколу HTTP. Вот как это происходит:



Полезно взглянуть пемпого пристальнее на запрос, отправляемый нами па сервер, и па ответ, который поступает. Запрос сообщает браузеру, какие даппные пас интересуют (ипогда мы пазываем их *ресурсом*, который пас интересует), а ответ будет содержать метадаппные и, если все пройдет нормальпо, информацию, запрошенную нами:

ЗАПРОС: используется протокол HTTP1.1 для извлечения ресурса из «/gumballsales» (наше приложение на сервере).

ОТВЕТ: первым идет заголовок протокола HTTP1.1; здесь сообщается, что для передачи данного ответа использовался протокол HTTP, а также приводится код ответа.

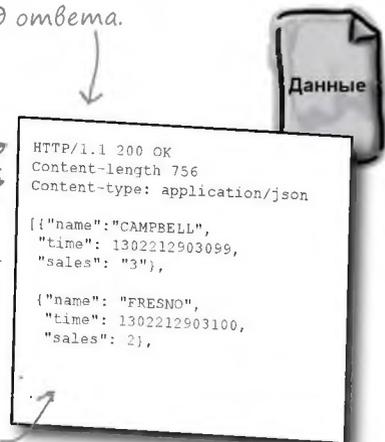


Метадаппные: мы отправляем запрос на хост gumball.wickedlysmart.com...

...при этом запрос исходит от браузера, совместимого с Mozilla 5.0 (данный пользовательский агент используется в случае с Safari, Chrome и другими браузерами).

200 — код ответа сервера, означающий, что все прошло нормально.

Метадаппные: мы получили содержимое, которое имеет длину 756 байт и тип application/json...



...а вот данные, которые мы запрашивали!

Примечание: данный шаблон извлечения данных с использованием XMLHttpRequest обычно называется Ajax или XHR.

Как выполнять запросы из JavaScript

Итак, теперь мы знаем, что можно извлекать данные по протоколу HTTP, но как именно? Напишем небольшой код для генерирования постоянного HTTP-запроса, после чего попросим браузер совершить запрос от нашего имени. Выполнив данный запрос, браузер передаст нам данные, которые получит в ответ. Пошагово рассмотрим совершение HTTP-запроса.

- 1 Начнем с URL. Ведь, в конце концов, нужно же нам как-то сказать браузеру, где искать данные, которые нас интересуют:

*Вот наш URL-адрес
someserver.com*

```
var url = "http://someserver.com/data.json";
```

Сохраним URL-адрес в переменной url, которую будем использовать далее.

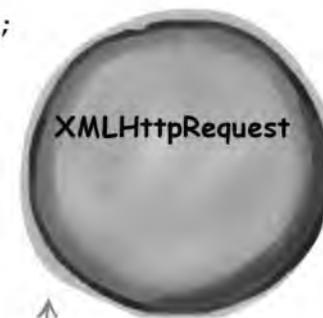
".json" означает формат обмена данными (к нему мы вернемся немного позже).

- 2 Теперь создадим объект request следующим образом:

```
var request = new XMLHttpRequest();
```

*↑
Присваиваем объект request переменной request.*

*↑
Используем конструктор XMLHttpRequest для создания нового объекта request. О XML в данном имени мы поговорим чуть позже*



Совершенно новый объект XMLHttpRequest.

- 3 Далее нам необходимо сообщить объекту request, какой URL ему нужно найти и какого рода запрос он должен использовать (мы задействуем стандартный HTTP-запрос GET так же, как на предыдущей странице). Для этого воспользуемся методом open объекта request. Может показаться, что, судя по имени open, данный метод может не только задавать соответствующие значения в объекте request, но и открывать новое соединение и извлекать данные. На самом деле это не так. Несмотря на имя, метод open лишь задает URL для объекта request, а также сообщает ему, какого рода запрос следует использовать, чтобы XMLHttpRequest смог произвести верификацию соединения. Вызов метода open осуществляется следующим образом:

```
request.open("GET", url);
```

*↑
Настраиваем объект request, задействуя HTTP-запрос GET, который является стандартным средством извлечения данных по протоколу HTTP.*

*↑
Настраиваем объект request на использование URL-адреса, сохраненного в переменной url.*

Обновленный объект XMLHttpRequest, который «знает» необходимый URL-адрес.



- 4 Итак, мы подошли к важной части и особенностям работы XMLHttpRequest: когда мы, наконец, попросим объект XMLHttpRequest извлечь данные, он приступит к работе и извлечет требуемую информацию. Это может занять 90 миллисекунд (что довольно много по компьютерным меркам), а порой даже 10 секунд (целая вечность по компьютерным меркам). Вместо того чтобы просто дожидаться данных, мы предусмотрим обработчик событий, который будет вызываться при их поступлении (эта процедура в некоторой степени уже должна быть вам знакома):

Наш объект request

```
request.onload = function() {
  if (request.status == 200) {
    alert("Data received!");
  }
};
```

Когда браузер получит ответ от удаленной веб-службы, он вызовет данную функцию.

Сначала обработчику необходимо проверить, является ли кодом ответа сервера значение 200, то есть все ли в порядке, после чего он сможет что-то сделать с данными. На данный момент мы просто будем выводить для пользователя диалоговое окно alert с сообщением, что данные поступили. Вскоре мы применим здесь более выразительный код.



```
request.onload = function() {
  if (request.status == 200) {
    alert("Data received!");
  }
};
```

- 5 Остался последний шаг: нам все еще нужно сказать объекту request, чтобы он извлек данные, для чего воспользуемся методом send:

```
request.send(null);
```

Благодаря этому происходит отправка запроса на сервер. Мы передаем null, если не отправляем какие-либо данные удаленной службе (чего мы как раз и не делаем).

Давайте еще раз пройдемся по всему процессу: мы создаем объект XMLHttpRequest, снабжаем его URL-адресом и HTTP-запросом наряду с обработчиком. Затем мы отправляем запрос и ждем поступления данных. Когда данные поступают, происходит вызов обработчика.





Мы просто еще не дошли до рассмотрения данного аспекта. Информацию, полученную посредством HTTP-запроса GET, можно найти в свойстве `responseText` объекта `request`. Таким образом, мы можем написать код вроде следующего:

```
request.onload = function() {  
    if (request.status == 200) {  
        alert(request.responseText);  
    }  
};
```

Вызов данной функции происходит, когда объект `request` получает ответ.

Мы можем извлечь ответ из свойства `responseText` объекта `request`.

Потерните немного, поскольку мы почти подошли к моменту написания реального кода, в котором задействуется `request.responseText`.

Развлечения с магнитами



Новая веб-служба <http://wickedlysmart.com/ifeelluckytoday> возвращает `unlucky` либо `lucky` при каждом обращении к ней. Логика в данном случае базируется на тайном и древнем алгоритме, о котором мы не можем вам рассказать, однако эта служба дает возможность пользователям узнать, повезет им (`lucky`) или нет (`unlucky`) в определенный день.

Нам требуется ваша помощь в создании показательной реализации, чтобы продемонстрировать другим, как они могут включить ее в свои сайты. Ниже приведен скелет кода; помогите нам заполнить пробелы в нем, используя магнитные таблички. Будьте внимательны, поскольку здесь есть лишние таблички. Одну из табличек мы уже разместили в нужном месте.

```

window.onload = function () {
    var url = "http://wickedlysmart.com/ifeelluckytoday";
    var request = _____
    _____ {
        if ( _____ ) {
            displayLuck( _____ );
        }
    };
    _____
    _____
}

function displayLuck(luck) {
    var p = document. _____ ("luck");
    P. _____ = "Today you are " + luck;
}
    
```

Чувствуете, что вам сегодня повезет? Хотите быть в этом уверены? Тогда воспользуйтесь нашей службой!



Разместите соответствующие таблички с кодом в нужных местах!



new XMLHttpRequest();

request.create("GET", url);

var i = 0;

request.responseText

request.send(null);

request.open("GET", url);

request.onload = function()

myLuckyText

new XMLHttpRequest();

request.status == 200

getElementById

Развлечения с магнитами. Решение



Новая веб-служба <http://wickedlysmart.com/ifeelluckytoday> возвращает `unlucky` либо `lucky` при каждом обращении к ней. Логика в данном случае базируется на тайном и древнем алгоритме, о котором мы не можем вам рассказать, однако эта служба дает возможность пользователям узнать, повезет им (`lucky`) или нет (`unlucky`) в определенный день.

Нам требуется ваша помощь в создании показательной реализации, чтобы продемонстрировать другим, как они могут включить ее в свои сайты. Ниже приведен скелет кода; помогите нам заполнить пробелы в нем, используя магнитные таблички. Будьте внимательны, поскольку здесь есть лишние таблички. Вот наше решение этого упражнения.

```

window.onload = function () {

    var url = "http://wickedlysmart.com/ifeelluckytoday";
    var request = new XMLHttpRequest();
    request.onload = function() {
        if ( request.status == 200 ) {
            displayLuck( request.responseText );
        }
    };
    request.open("GET", url);
    request.send(null);
}

function displayLuck(luck) {
    var p = document.getElementById("luck");
    p.innerHTML = "Today you are " + luck;
}
    
```

←
Разместите со-
ответствующие
таблички с кодом
в нужных местах!

Чувствуете, что вам
сегодня повезет? Хотите
быть в этом уверены? Тог-
да воспользуйтесь нашей
службой!

↙ Лишние таблички

```

var i = 0;
request.create("GET", url);
myLuckyText
new TextHttpRequest();
    
```





ВСТРЕЧАЕМ XMLHttpRequest

общение с веб-службами

Интервью недели:

Признания объекта XMLHttpRequest

Head First: Добро пожаловать, XMLHttpRequest, мы рады, что вы смогли выкроить для нас время в своем плотном графике. Расскажите, как Вы вписываетесь в процесс создания веб-приложений.

XMLHttpRequest: Я положил начало целому тренду по введению внешних данных в веб-страницы. Слышали о Google Maps? О Gmail? Это все я. Без меня все это было бы невозможным.

Head First: В каком смысле?

XMLHttpRequest: До моего появления люди генерировали веб-страницы на стороне сервера, при этом в них закладывались сразу все данные. Я же позволяю получать данные *после* того, как страница сгенерирована. Вспомните Google Maps: данная служба обновляет то, что имеется на странице, каждый раз, когда происходит корректировка вашего местоположения на карте, без необходимости перезагрузки страницы целиком.

Head First: Вы пользуетесь успехом. В чем ваш секрет?

XMLHttpRequest: В моей непритязательности и простоте. Дайте мне URL-адрес, и я отправлюсь по нему и извлеку необходимые вам данные. Не более того.

Head First: И этим все ограничивается?

XMLHttpRequest: Что ж, вам потребуется сказать мне, что дальше делать с данными после того, как я их извлеку. Вы можете просто дать мне функцию обработчика — функцию обратного вызова, — и когда я добуду данные, я передам их этому обработчику, который сможет сделать с ними все, что ему потребуется.

Head First: О какого рода данных мы говорим в данном случае?

XMLHttpRequest: Современный Интернет полон всевозможных данных; прогнозы погоды, карты, социальные сведения о разных людях, геолока-

ционные данные о том, что расположено поблизости... Почти любые данные, которые вы только можете себе представить, но найдут во Всемирную паутину в форме, с которой я могу работать.

Head First: Это же ведь все исключительно XML-данные, не так ли? Я имею в виду, что в Вашем имени присутствует часть XML.

XMLHttpRequest: Неужели? Позвольте ответить прямо. Несомненно, было время, когда я, главным образом, занимался извлечением XML-данных, однако мир не стоит на месте. В настоящее время я могу извлекать всевозможные данные. Естественно, некоторые из них являются XML-данными, однако я получаю все больше запросов на извлечение JSON-данных.

Head First: В самом деле? А что такое JSON и почему он набирает такую популярность?

XMLHttpRequest: JSON — это сокращение от JavaScript Object Notation (нотация JavaScript-объектов). У этого формата имеется ряд преимуществ: размер, удобочитаемость, а также тот факт, что он является «родным» для наиболее популярного языка программирования, используемого при создании веб-приложений, которым, конечно же, выступает мой друг JavaScript.

Head First: А правда, что на самом деле формат не должен иметь для Вас значения? Пользователи должны быть способны запрашивать данные в формате XML, JSON, а также телетайпное текстовое содержимое с Вашей помощью. Или нет?

XMLHttpRequest: <silence>

Head First: Что ж, судя по вашему молчанию, я затронул болезненную тему. Ладно, пора сделать перерыв... Послушайте, XMLHttpRequest, ведь у нас же еще будет время для общения с Вами позже в этой главе?

XMLHttpRequest: Да, хоть здесь и нет ничего радостного, но я вижу это в своем расписании...

Погвинься, XML: Встречайте JSON

Как вы можете помнить (а можете и не помнить), XML собирался стать нашим всеобщим спасителем, — это формат данных, который удобочитаем для человека и может быть подвергнут разбору машиной, при этом он был готов превратиться в универсальный формат для обмена данными во всем мире. Действительно, с появлением XMLHttpRequest XML стал форматом, с помощью которого мы все обменивались данными (отсюда и часть XML в имени XMLHttpRequest).

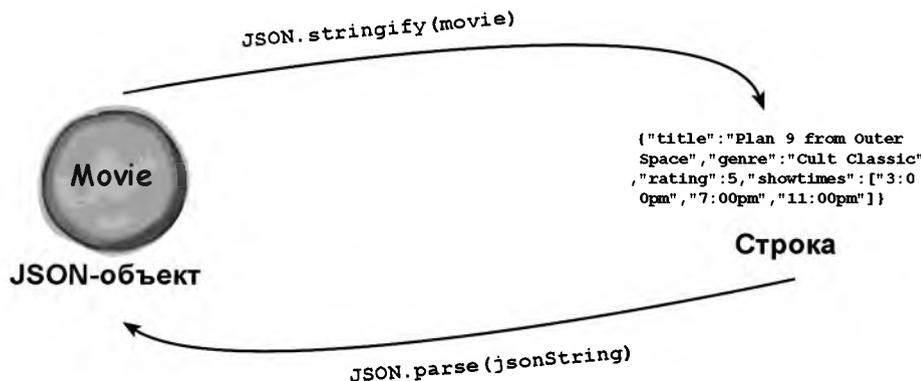
Однако по ходу дела XML, по-видимому, «носкользнул на банановой кожуре», которую ему подбросил JSON. Что такое JSON? Это наиболее современный и отличный формат данных, рожденный JavaScript, который обретает поддержку в сфере Интернета — в браузерах и на стороне сервера. Интересно, удастся ли ему быстро стать *предпочтительным форматом* для HTML5-приложений?

Так что же такого замечательного в JSON? Это вполне удобочитаемый для человека формат, данные в котором могут быть легко и быстро разобраны и преобразованы прямо в JavaScript-значения и объекты. В отличие от XML, он такой классный и симпатичный... так или иначе, разве можно после всего этого сказать, что он нравится нам лишь чуть-чуть? Вы много раз будете сталкиваться с JSON далее в книге. Мы будем использовать его для обмена JavaScript-данными через Интернет, для сохранения данных в localStorage с применением API-интерфейса Web Storage, а также как часть еще одного подхода к получению доступа к веб-данным (вскоре мы поговорим об этом подробнее).

Постойте-ка, форматы обмена данными через Интернет... форматы хранения... все это так сложно, не правда ли? Не беспокойтесь, поскольку на протяжении следующих десяти страниц мы будем превращать вас в эксперта вы уже знаете практически все необходимое о JSON. Чтобы использовать JSON, вам потребуется лишь понимание JavaScript-объектов (а эти знания у вас уже есть), а также два простых вызова методов.

- 1 У нас имеется JavaScript-объект, мы хотим обменяться им или сохранить его, поэтому вызываем метод `JSON.stringify` и передаем ему этот объект в качестве аргумента.

- 2 Результатом будет строка, представляющая объект. Мы можем сохранить данную строку, передать ее функции, отправить через Интернет и т. п.



- 4 Результатом будет копия нашего оригинального объекта.

- 2 Когда мы будем готовы преобразовать строку обратно в объект, мы передадим ее методу `JSON.parse`.

Краткий пример с использованием JSON

- ① Рассмотрим пример преобразования объекта в строковый формат JSON. Начнем с объекта, который будет вам неясен, — `movie` из главы 4. Однако не все можно преобразовать в данный формат (например, методы), зато поддерживаются все базовые типы, такие как `number`, `string`, а также массивы. Давайте создадим объект, а затем передадим его методу `JSON.stringify`:

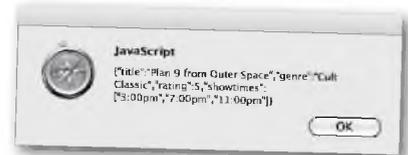
На самом деле есть еще ряд ограничений, однако мы не станем беспокоиться о них сейчас.

```
var plan9Movie = new Movie("Plan 9 from Outer Space", "Cult Classic", 2,
    ["3:00pm", "7:00pm", "11:00pm"]);
```

Наш объект `movie`, «упакованный» строками, числами и массивом.

- ② Располагая объектом, мы можем преобразовать его в строковый формат JSON с помощью метода `JSON.stringify`. Посмотрим, как это происходит... (вы можете вернуться к коду `movie` из главы 4 и добавить приведенный далее код в нижнюю часть своего сценария):

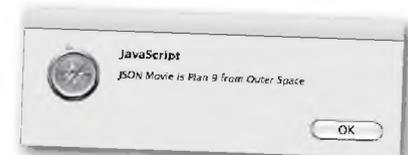
```
var jsonString = JSON.stringify(plan9Movie);
alert(jsonString);
```



А вот результат: строковая версия объекта, отображаемая в диалоговом окне `alert`.

- ③ Теперь у нас имеется строка JSON, представляющая объект `movie`. На данный момент мы можем взять эту строку и, например, отправить ее по протоколу HTTP на сервер. Мы также можем получить строку JSON с другого сервера. Допустим, сервер прислал нам данную строку; как преобразовать ее обратно в объект, с которым мы сможем что-нибудь сделать? Для этого нужно просто воспользоваться «сестрой» метода `JSON.stringify` — `JSON.parse`. Делается это так:

```
var jsonMovieObject = JSON.parse(jsonString);
alert("JSON Movie is " + jsonMovieObject.title);
```



Ах да, теперь мы используем это как настоящий объект, обращаясь к его свойствам.



Введите данный URL-адрес. Что вы видите?

`http://search.twitter.com/search.json?q=hfhtml5`

Примечание: браузер Firefox предложит вам открыть либо сохранить файл. Для открытия можете использовать `TextEdit`, Блокнот или любой другой базовый текстовый редактор.



Спецификации только что прибыли!



Mighty Gumball, Inc.

Там, где автомат для продажи жевательной резинки никогда не пуст наполовину

Спецификации сервера Mighty Gumball

Спасибо, что согласились помочь!

Отчеты об уровне продаж, поступающие от автоматов для продажи жевательной резинки, собираются вместе и доступны на нашем центральном сервере по адресу

<http://gumball.wickedlysmart.com/>

В качестве формата для наших данных мы выбрали JSON, и если воспользоваться приведенным выше URL-адресом, то обратно можно получить массив JSON-объектов, которые выглядят так:

```
[{"name": "CAMPBELL",
  "time": 1302212903099,
  "sales": 3},
```

← Название города; на данный момент мы тестируем свои торговые автоматы в Калифорнии.

```
{"name": "FRESNO",
  "time": 1302212903100,
  "sales": 2},
```

← Время поступления данного отчета в миллисекундах.

← Количество проданной жвачки с момента последнего отчета.

```
...
]
```

← Второй город, Фресно.

← А здесь будут другие города...

Введите данный URL-адрес в своем браузере, чтобы увидеть, как в ответ поступают значения. На экране должен появиться один объект или более в массиве.

Вы также можете добавить параметр `lastreporttime` в конец URL-адреса, чтобы извлечь только отчеты, которые поступили начиная с указанного времени. Например:

<http://gumball.wickedlysmart.com/?lastreporttime=1302212903099>

← Просто укажите время в миллисекундах.

У нас сотни автоматов по продаже жевательной резинки, которые присылают отчеты прямо сейчас, поэтому вы должны увидеть, что обновление отчетности происходит в среднем каждые 5–8 секунд. Учтите, что это наш производственный сервер, поэтому мы просим вас предварительно проводить локальное тестирование своего кода!

Благодарим, что согласились нам помочь! И помните, что, как говорит наш исполнительный директор, «автомат для продажи жевательной резинки никогда не пуст наполовину».

– Инженеры компании Mighty Gumball

← Непременно сделайте это!

Пора за работу!

Итак, мы получили спецификации от инженеров, а также поговорили с вами о XMLHttpRequest и JSON. Вы должны быть уже готовы к тому, чтобы переходить к написанию кода и первому запуску приложения Mighty Gumball.

Ранее мы с вами написали HTML-разметку, положив начало нашему веб-приложению, и предусмотрели в ней ссылку на файл mightygumball.js. Сейчас займемся написанием кода, который будет в нем размещаться. Напомним, что мы также оставили в HTML-разметке место, где будем размещать данные об уровне продаж прямо в <div> с id в виде «sales». Соединим все и напишем код.

```
<!doctype html>
<html lang="en">
  <head>
    <title>Mighty Gumball (JSON)</title>
    <meta charset="utf-8">
    <script src="mightygumball.js"></script>
    <link rel="stylesheet" href="mightygumball.css">
  </head>
  <body>
    <h1>Mighty Gumball Sales</h1>
    <div id="sales">

  </div>
  </body>
</html>
```

Написание функции обработчика событий onload

Уверены, что в данной процедуре для вас не будет ничего нового, однако мы собираемся написать обработчик событий onload, который станет вызываться после полной загрузки HTML; мы также предусмотрим инициирование HTTP-запроса для извлечения данных об уровне продаж. Когда данные будут возвращены, мы попросим XMLHttpRequest вызвать функцию updateSales (написанием которой займемся чуть позже):

```
window.onload = function() {
  var url = "http://localhost/sales.json";
  var request = new XMLHttpRequest();
  request.open("GET", url);
  request.onload = function() {
    if (request.status == 200) {
      updateSales (request.responseText);
    }
  };
  request.send(null);
}
```

Сначала проведем тестирование с использованием локального файла (как и просили нас инженеры из Mighty Gumball!), чтобы убедиться, что все работает. Подробнее об этом поговорим немного позже...

Формируем XMLHttpRequest путем создания объекта, вызова метода open с использованием нашего URL-адреса и последующего присваивания свойства onload функции.

Проверяем, все ли в порядке, а затем...

...когда загрузка данных будет завершена, произойдет вызов этой функции.



Будьте
осторожны!

Если вы используете Opera либо Internet Explorer 8 (или ниже), рекомендуем вам проводить тестирование с применением другого браузера. Об особенностях поддержки Opera и старых версий Internet Explorer мы поговорим позже.

Отображение данных об уровне продаж жвачки

Теперь нам необходимо написать обработчик `updateSales`. Облегчим задачу и прибегнем к наиболее простой реализации из возможных, поскольку всегда сможем улучшить ее позже:

```
function updateSales(responseText) {  
    var salesDiv = document.getElementById("sales");  
    salesDiv.innerHTML = responseText;  
}
```

Извлечем элемент `<div>`, который уже помещен в HTML, и будем использовать его для размещения данных.

В качестве содержимого `<div>` задаем строку `responseText`. О разборе этого содержимого мы вскоре поговорим... А сначала давайте проведем тестирование.

Внимание, впереди объезд!



Пришло время провести новый тест-драйв, однако сначала нам необходимо сделать, так сказать, небольшой крюк по объездному пути. Инженеры из компании **Mighty Gumball** попросили нас провести тестирование локально, прежде чем задействовать их производственный сервер, что является разумной идеей. Однако для этого нам потребуется, чтобы данные располагались на сервере, благодаря чему XMLHttpRequest сможет использовать протокол HTTP для их извлечения.

Относительно серверов у вас есть несколько альтернатив:

- если ваша компания располагает серверами, доступными для тестирования, то используйте именно их;
- либо вы можете прибегнуть к помощи хостинговых сервисов вроде GoDaddy, Dreamhost или одной из множества других компаний, предлагающих услуги хостинга;
- наконец, вы можете установить сервер прямо на своем компьютере. В этом случае ваши URL-адреса будут выглядеть примерно так:

```
http://localhost/mightygumball.html
```

Файлы также могут располагаться в подкаталоге: например, `http://localhost/gumball/mightygumball.html`

Необходимые советы и указатели вы найдете на следующей странице. Имейте в виду, что хостинговые окружения довольно сильно отличаются друг от друга, поэтому мы не можем привести здесь общее руководство для всех них. И если у вас не окажется легкого доступа к тому или иному серверу, установка собственного сервера локально на своем компьютере может оказаться наилучшим выходом!





Как установить собственный веб-сервер

Установка собственного локального сервера на своем компьютере будет зависеть от типа используемой вами операционной системы. Ниже приведены советы по установке сервера в операционных системах OS X (также называемой Mac OS X), Windows и Linux. Прочие варианты вы найдете на следующей странице.

Mac OS X



Установка веб-сервера в операционной системе OS X осуществляется довольно просто. Щелкните  > System Preferences (Настройки системы), после чего выберите Sharing (Совместный доступ). На панели слева проверьте, установлен ли флажок Web Sharing (Совместный веб-доступ):



Активировав параметр Web Sharing (Совместный веб-доступ) (либо он уже может быть активированным), вы увидите информацию о том, как получить доступ к своему локальному серверу. Вы должны иметь возможность использовать localhost вместо IP-адреса (который имеет тенденцию изменяться при использовании DHCP-маршрутизатора, в силу чего localhost лучше подойдет в вашем случае). По умолчанию ваши файлы будут загружаться из http://localhost/~ИМЯ_ПОЛЬЗОВАТЕЛЯ/, куда они в свою очередь загружаются из папки ИМЯ_ПОЛЬЗОВАТЕЛЯ/Sites, поэтому вы, вероятно, захотите создать там подкаталог для Mighty Gumball.

Windows



Инсталляция собственного веб-сервера в операционной системе Windows стала проще, чем это было раньше, благодаря установщику Microsoft Web Platform Installer (также называемому Web PI). Текущая версия поддерживается операционными системами Windows 7, Vista SP2, XP SP3+, Server 2003 SP2+, Server 2008 и Server 2008 R2 и доступна для загрузки по адресу <http://www.microsoft.com/web/downloads/platform.aspx>.

В качестве альтернативы можно установить программу с открытым исходным кодом WampServer, которая поставляется вместе с Apache, PHP и MySQL для разработки веб-приложений. Она легка в инсталляции и управлении.

Загрузить WampServer на свой компьютер можно с сайта <http://www.wampserver.com/en/>.

Существуют также другие доступные решения с открытым исходным кодом, поэтому у вас будет большой выбор.

~~Фанатские~~ дистрибутивы Linux

Взглянем в лицо фактам: вы уже знаете, что делать в данном случае. Мы угадали? Apache обычно устанавливается по умолчанию, поэтому читайте документацию к используемому вами дистрибутиву Linux.



Как установить собственный веб-сервер (продолжение)

Вы хотите разместить свои файлы на *настоящем* сервере в Интернете? Прекрасно, но знайте, что в данном случае у вас будет только один выход — воспользоваться услугами хостинга. Ознакомьтесь с приведенными ниже советами, и вперед!

Услуги хостинга, предоставляемые сторонними организациями...

Если вы не хотите заниматься установкой собственного сервера, то всегда можете воспользоваться удаленным сервером, однако вам придется разместить свои HTML-, JavaScript- и CSS-файлы, а также JSON-файл на одном сервере (о том, почему это так важно, поговорим позже), следуя приведенному здесь примеру.

Большинство хостинговых сервисов обеспечат вам FTP-доступ к папке, в которую вы сможете поместить все свои файлы. Если у вас есть доступ к подобному серверу, то выгрузите туда все файлы и подставьте имя данного сервера вместо localhost везде, где увидите его.



Вы можете прибегнуть к TP-программе (например, Transit, Cyberduck или WinSCP), для выгрузки своих файлов, если не хотите использовать FTP командной строки.

Мы составили список поставщиков услуг хостинга на случай, если вам потребуется подсказка, однако их и так можно отыскать без особого труда; просто введите в поисковик «*веб-хостинг*», и он выдаст вам массу вариантов. Составленный нами список доступен по адресу <http://wickedlysmart.com/hfhtml5/hosting/hosting.html>. И дайте нам знать, если у вас появится собственный веб-сайт HTML5 в Интернете, поскольку нам будет любопытно взглянуть на него!

Возвращаемся к коду

На данный момент мы предполагаем, что вы уже установили собственный сервер, — это может быть сервер, выполняющийся на вашем локальном компьютере (как в нашем случае), либо сервер, расположенный где-то в другом месте, и у вас есть к нему доступ. И в том и в другом случае вы будете размещать свои HTML- и JavaScript-файлы на этом сервере, а затем укажете браузеру путь к соответствующему HTML-файлу. Вам также потребуется тестовый файл с данными об уровне продаж Mighty Gumball, поэтому мы предоставим вам простой файл с такими данными, который вы сможете поместить на свой сервер. Для вашего приложения он будет выглядеть так, будто поступает с центрального сервера Mighty Gumball, обеспечивающего обновление данных ночью в режиме реального времени, что даст вам возможность протестировать свой код, не задействуя производственный сервер Mighty Gumball. Далее показано, как будет выглядеть данный файл; он называется `sales.json` и включен в код, используемый в этой книге и доступный в Интернете (вы можете и самостоятельно набрать его, если вам нравится этот процесс):

```
[{"name": "ARTESIA", "time": 1308774240669, "sales": 8},
 {"name": "LOS ANGELES", "time": 1308774240669, "sales": 2},
 {"name": "PASADENA", "time": 1308774240669, "sales": 8},
 {"name": "STOCKTON", "time": 1308774240669, "sales": 2},
 {"name": "FRESNO", "time": 1308774240669, "sales": 2},
 {"name": "SPRING VALLEY", "time": 1308774240669, "sales": 9},
 {"name": "ELVERTA", "time": 1308774240669, "sales": 5},
 {"name": "SACRAMENTO", "time": 1308774240669, "sales": 7},
 {"name": "SAN MATEO", "time": 1308774240669, "sales": 1}]
```

Будем использовать `sales.json` для проведения тестирования, прежде чем обратимся к настоящему производственному серверу с данными об уровне продаж, поступающих в режиме реального времени.

Разместите данный файл на своем сервере, после чего не забудьте обновить свой JavaScript, указав в нем URL-адрес этого файла. В нашем случае он будет выглядеть как `http://localhost/gumball/sales.json`:

Полезно сначала протестировать этот URL в своем браузере, чтобы убедиться, что он работает.

```
window.onload = function() {
  var url = "http://localhost/gumball/sales.json";
  var request = new XMLHttpRequest();
  request.open("GET", url);
  request.onload = function() {
    if (request.status == 200) {
      updateSales(request.responseText);
    }
  };
  request.send(null);
}
```

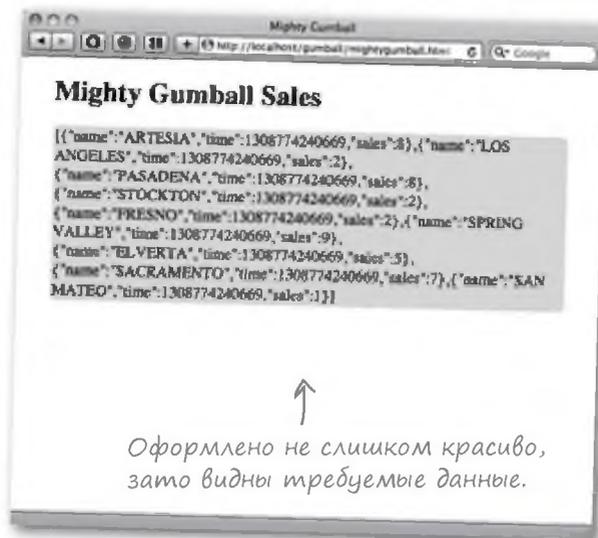
Убедитесь, что здесь указан правильный URL-адрес.



Давайте, наконец, проведем тестирование!

Мы проделали долгий нуть и, наконец, готовы переходить к тестированию нашего кода!

Убедитесь, что ваши файлы HTML, JavaScript, JSON – и не забудьте о CSS – размещены на используемом вами сервере. Введите URL-адрес вашего HTML-файла в браузере (в нашем случае это `http://localhost/gumball/mightygumball.html`), после чего нажмите Назад...



Помните, что мы отправляли HTTP-запрос для извлечения данных, содержащихся в `sales.json`, которые затем просто сбросили в элемент `<div>` Похоже, все работает!

Если у вас возникнут проблемы, проверьте каждый файл отдельно с использованием браузера и убедитесь, что они доступны. Затем дважды проверьте свои URL-адреса.

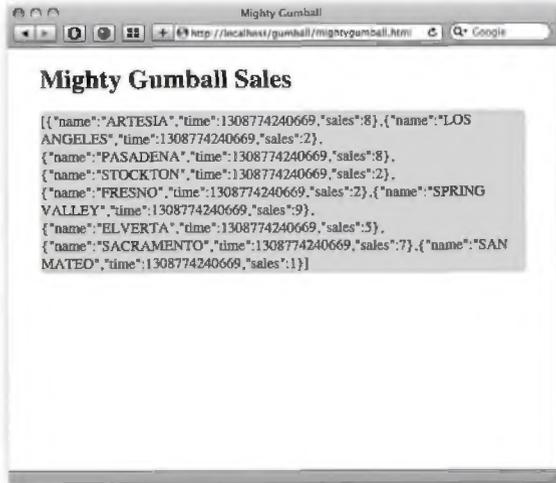


Здорово! Да, пришлось потрудиться. Нам потребовалось разобраться, как совершать HTTP-запросы и устанавливать собственный сервер, но в итоге все получилось! Я уже задумываюсь об отличных приложениях, которые смогу создать для использования всевозможных веб-служб, общаться с которыми я теперь умею.

Производим впечатление на клиента...

Мы приложили массу усилий, чтобы в итоге получить работающее приложение. Это замечательно, однако наш клиент Mighty Gumball будет более впечатлен, если данное приложение будет еще и выглядеть хорошо. Ниже показано, какого именно результата мы собираемся добиться.

Что мы имеем



Пока мы просто сбрасываем JSON-массив прямо в браузер. В некоторой степени эффективно, но результат получается не очень красивым. Какая жалость, ведь есть целая структура данных, которая только и ждет более эффективного подхода к ее использованию!

Чего мы хотим



Здесь мы использовали JSON-массив и превратили его в красиво отображаемые данные.

Вот что требуется, чтобы обеспечить более красивое отображение данных:

- ① Сначала нам нужно взять данные, полученные от объекта XMLHttpRequest (которые представляют собой простую JSON-строку), и преобразовать их в подлинный JavaScript-объект.
- ② Затем мы сможем пройтись по результирующему массиву и добавить новые элементы в объектную модель документа (DOM) — по одному в случае с каждым элементом данных об уровне продаж в массиве.

Дорабатываем код с целью использования JSON

Выполним описанные выше два шага и откорректируем наш код необходимым образом:

- Сначала нам нужно взять данные, полученные от объекта XMLHttpRequest (которые представляют собой простую JSON-строку), и преобразовать их в подлинный JavaScript-объект.

Чтобы это сделать, обновим функцию updateSales, первым делом удалив строку кода, которая задает для <div> содержимое в виде строки responseText, и преобразуем responseText из строки в ее JavaScript-эквивалент с помощью JSON.parse.

```
function updateSales(responseText) {
    var salesDiv = document.getElementById("sales");
    salesDiv.innerHTML = responseText;
    var sales = JSON.parse(responseText);
}
```

Эта строка нам больше не нужна.

Здесь мы берем ответ и используем JSON.parse для преобразования его в JavaScript-объект (в данном случае это будет массив) и присваиваем его переменной sales.

- Теперь пройдемся по результирующему массиву и добавим новые элементы в объектную модель документа (DOM) — по одному в случае с каждым элементом данных об уровне продаж в массиве. В нашем случае мы создадим новый <div> для каждого элемента данных:

```
function updateSales(responseText) {
    var salesDiv = document.getElementById("sales");
    var sales = JSON.parse(responseText);
    for (var i = 0; i < sales.length; i++) {
        var sale = sales[i];
        var div = document.createElement("div");
        div.setAttribute("class", "saleItem");
        div.innerHTML = sale.name + " sold " + sale.sales + " gumballs";
        salesDiv.appendChild(div);
    }
}
```

Совершаем итерацию по каждому элементу данных в массиве.

Для каждого элемента данных мы создаем <div> и присваиваем ему класс «saleItem» (используемый CSS).

Задаем содержимое для элемента <div> с помощью innerHTML, после чего добавляем его в качестве дочернего элемента по отношению к salesDiv.

Заключительный этап...



Вы уже знаете, как будет выглядеть конечный результат, но все-таки внесите описанные выше изменения в свой код. Взгляните более пристально на код, приведенный на предыдущей странице, и убедитесь, что вы полностью в нем разобрались. Затем перезагрузите веб-страницу Mighty Gumball.

Как видите, конечный результат получился таким, как мы вам и говорили!



Тестирование прошло удачно, и теперь вы, ребята, готовы перейти к использованию действующего производственного сервера Mighty Gumball. Удачи!

Переходим к использованию действующего сервера

Инженеры из Mighty Gumball просили нас предварительно провести тестирование локально, что мы и сделали. Теперь мы готовы переходить к тестированию с использованием реального сервера Mighty Gumball. На этот раз вместо извлечения статического JSON-файла данных будем извлекать JSON-файл, который динамически генерируется сервером Mighty Gumball. Нам потребуется обновить URL-адрес, используемый объектом XMLHttpRequest, и изменить его таким образом, чтобы он вел к серверу Mighty Gumball. Давайте сделаем это:

URL-адрес сервера Mighty Gumball. Замените прежний URL-адрес этим и сохраните.

```

window.onload = function() {
  var url = "http://gumball.wickedlysmart.com";
  var request = new XMLHttpRequest();
  request.open("GET", url);
  request.onload = function() {
    if (request.status == 200) {
      updateSales(request.responseText);
    }
  };
  request.send(null);
}

```



Эйджей, контролер качества

Тест-драйв с применением действующего сервера...

Убедитесь, что вы сохранили изменения, касающиеся URL-адреса, в своем файле `mightygumball.js` на сервере, если хотите продолжать извлекать HTML оттуда, либо локально на жестком диске, если используете `localhost`. Вы уже знаете, что делать дальше: укажите своему браузеру путь к соответствующему HTML-файлу, после чего увидите, как будут поступать оперативные, красиво оформленные, настоящие данные от торговых автоматов Mighty Gumball по всему миру!



Хьюстон, у нас проблема! Идите скорее сюда, мы вообще не можем получить какие-либо данные об уровне продаж после того, как перешли к использованию действующего сервера Mighty Gumball!

↑ Это еще что?! Мы не видим вообще никаких данных!

Ой!

А все выглядело так хорошо; мы уже представляли себе, как будем потягивать минеральную воду «Peggie» и отмечать успешное завершение еще одного проекта с Mighty Gumball. А теперь все может пойти насамадку. Ладно, не будем излишне драматизировать ситуацию, но в чем же все-таки дело? Ведь все должно работать!

Сделайте глубокий вдох. Тому есть логическое объяснение...

↑ Примечание для редактора: вообще-то мы представляли себе, как будем обналичивать жирный чек и отправлять вам эту книгу! А вместо этого нам теперь придется дописывать ее, рассказывая о том, как найти выход из очередной непростой ситуации!

↑ Эйджей, расстроенный контролер качества

Неожиданный поворот событий!

Мы вообще не видим каких-либо данных на странице Mighty Gumball. Все прекрасно работало, нока мы не перешли к использованию действующего сервера...

Удастся ли нам *найти* причину проблемы?

Удастся ли нам *устранить* ее?

Оставайтесь с нами... мы ответим на эти вопросы, а также...

Между тем попытайтесь самостоятельно разобраться в том, что именно пошло не так и как это можно исправить.



КЛЮЧЕВЫЕ МОМЕНТЫ



- Для извлечения HTML-файлов или данных с сервера браузер отправляет HTTP-запрос.
- HTTP-ответ включает код ответа, который позволяет узнать, произошли ли какие-либо ошибки при выполнении запроса.
- Код 200 в HTTP-ответе означает, что при выполнении запроса не произошло никаких ошибок.
- Для отправки HTTP-запроса из JavaScript используется объект XMLHttpRequest.
- Обработчик событий onload объекта XMLHttpRequest обрабатывает получение ответа от сервера.
- JSON-ответ на XMLHttpRequest помещается в свойство responseText объекта request.
- Для преобразования строки responseText в JSON используется метод JSON.parse.
- XMLHttpRequest задействуется в приложениях для обновления содержимого, (например, карт или электронной почты) без необходимости перезагрузки страницы.
- XMLHttpRequest может применяться для извлечения всевозможного текстового содержимого (например, XML, JSON и др.).
- XMLHttpRequest Level 2 является самой последней версией XMLHttpRequest, однако данный стандарт все еще находится в разработке.
- Чтобы использовать XMLHttpRequest, соответствующие файлы необходимо разместить на сервере, с которого и будут запрашиваться данные. Вы можете установить локальный сервер на своем компьютере для проведения тестирования либо воспользоваться хостинговыми сервисами.
- Свойство onload объекта XMLHttpRequest не поддерживается старыми версиями браузеров, такими как Internet Explorer 8 и ниже, а также Opera 10 и ниже. В этом случае вы можете написать код для проверки версии браузера и обеспечения альтернативы.



ВСТРЕЧАЕМ

XMLHttpRequest, часть 2

Интервью недели:

Internet Explorer и «Вы сказали JSON?»?

Head First: Добро пожаловать вновь на наше интервью, XMLHttpRequest. Я хотел спросить у Вас насчет браузеров – Вы поддерживаете только их новейшими версиями?

XMLHttpRequest: Меня не зря называют старожилом; браузеры поддерживают меня с 2004 года.

Head First: Что ж, а как насчет морального устаревания, Вас это не беспокоит?

XMLHttpRequest: Я объект, новая версия которого выходит каждые 10 лет или около того. В настоящий момент полным ходом идет работа над моей второй версией, называемой XMLHttpRequest Level 2. Фактически, большинство современных браузеров уже поддерживают данную версию.

Head First: Это впечатляет. А в чем заключаются особенности версии Level 2?

XMLHttpRequest: Прежде всего в поддержке большего количества типов событий, благодаря чему вы сможете, например, отслеживать ход выполнения запроса и писать более элегантный код.

Head First: Говоря насчет браузерной поддержки...

XMLHttpRequest: До этого мы сейчас дойдем... не торонитесь...

Head First: По слухам, вы и Internet Explorer на самом деле не ладите друг с другом...

XMLHttpRequest: Вы что, шутите? Вся история XMLHttpRequest началась с Internet Explorer.

Head First: А как насчет XMLHttpRequest и XMLHttpRequest? Вам доводилось слышать эти имена?

XMLHttpRequest: Это мои прозвища! Так меня называют в Microsoft! Да, я согласен, что наличие у меня разных имен – не очень хорошо, однако под ними подразумеваются инструменты, которые решают одну и ту же задачу. Для улаживания данной ситуации Вам потребуется лишь небольшое количество дополнительного кода, а с точки зрения последних версий браузера Internet Explorer от Microsoft начиная с версии 9 и выше все и так будет в порядке. Если это новость для Ваших читателей, то я с радостью задержусь после интервью, чтобы позаботиться о том, чтобы их код был совместим с более старыми версиями Internet Explorer.

Head First: Мило с Вашей стороны, но мы займемся этим вопросом как-нибудь позже в данной главе.

XMLHttpRequest: Эй, я хороший парень и не стану бросать ваших читателей наедине с проблемой.

Head First: Ловим Вас на слове. И еще один вопрос: Вы упоминали JSON и говорили, что являетесь его большим поклонником. А что, JSONP Вас вообще никак не беспокоит? По слухам, многие люди используют его вместо Вас.

XMLHttpRequest: Да, с помощью JSONP вы, конечно же, сможете извлекать данные, однако это всего лишь ловкий хитрый прием. Я хочу сказать: задумайтесь о том витиеватом коде, который вам придется написать. И как насчет безопасности?

Head First: Я не слишком технически подкован и знаю лишь, что люди говорят, будто он дает им возможность обойти проблемы, которые Вы не позволяете решить. Впрочем, наше время истекло.

XMLHttpRequest: Что ж, ну хотя бы в части «не слишком технически подкован» Вы не ошиблись.



**Будьте
осторожны!**

Свойство `onload` объекта `XMLHttpRequest` не поддерживается устаревшими версиями браузеров, однако из этой ситуации есть простой выход.

Мы использовали `request.onload` для определения функции, вызываемой, когда завершается извлечение запрошенных данных с сервера. Это возможность `XMLHttpRequest Level 2` (считайте ее «версией 2»). Версия `XMLHttpRequest Level 2` остается все еще довольно новой, то есть многие пользователи могут до сих пор использовать браузеры, которые ее не поддерживают. В частности, *Internet Explorer 8* (и ниже), а также *Opera 10* (и ниже) поддерживают только `XMLHttpRequest Level 1`. Хорошая новость заключается в том, что новые возможности `XMLHttpRequest Level 2` являются расширениями, поэтому вы сможете без проблем продолжить использовать именно возможности версии 1 во всех браузерах; это всего лишь означает, что ваш код не будет отличаться особой элегантностью. Вот код для использования `XMLHttpRequest Level 1`:

Большая часть кода для использования `XMLHttpRequest Level 1` является такой же, как и раньше...

...Однако в `XMLHttpRequest Level 1` нет свойства `request.onload`, поэтому вместо него вам придется использовать свойство `onreadystatechange`.

```
function init() {
    var url = "http://localhost/gumball/sales.json";
    var request = new XMLHttpRequest();
    request.onreadystatechange = function() {
        if (request.readyState == 4 && request.status == 200) {
            updateSales(request.responseText);
        }
    };
    request.open("GET", url);
    request.send(null);
}
```

Все остальное, по большей части, является таким же, как и раньше.

Вы также можете проводить проверку на предмет других значений `readyState` и `status` с целью выявления различных ошибок.

Затем проверьте `readyState`, чтобы убедиться, что загрузка данных завершилась. Если значением `readyState` является 4, то вы будете знать, что загрузка закончена.

Помните, как мы столкнулись с неожиданным поворотом событий? Неполадки с приложением

Наш код прекрасно работал, когда мы использовали свой локальный сервер, но как только мы перешли на действующий сервер Mighty Gumball в Интернете, с нашим приложением возникли неполадки!

Чего мы ожидали:



Вот как в итоге выглядит наша страница после того, как мы выполнили код и извлекли данные об уровне продаж со своего локального сервера, используя адрес <http://localhost/gumball/sales.json>.

Что мы получили:



А вот как в итоге выглядит наша страница после того, как мы выполнили код и извлекли данные об уровне продаж с сервера Mighty Gumball, используя адрес <http://gumball.wickedlysmart.com>.

Что же делать дальше?!

Пожалуй, дальше мы будем делать то же, что и всегда, — соберем всех членов команды и быстро все обсудим. Мы уверены, что все вместе (включая вымышленных персонажей) мы сможем выяснить причину проблемы! Фрэнк? Джим? Джо? Где вы все? А, вот где — на следующей странице...



Эйджей, довольно сильно расстроенный контролер качества

Джим, я не понимаю, что случилось с этим кодом, но он не работает.



Джим: URL-адрес правильный?

Фрэнк: Да, правильный, я уже вводил его в браузер, чтобы убедиться в том, что увижу ожидаемые нами данные об уровне продаж, и все нормально работало. Не понимаю...

Джо: Я заглянул в консоль JavaScript в браузере Chrome и увидел что-то об управлении доступом и источниках или доменах.

Фрэнк: Хм?

Ребята, где вы были, когда мы занимались проектом Starbuzz Coffee? Если помните, у нас была проблема с аналогичным поведением. Держу пари, что на этот раз вы столкнулись с междоменными проблемами, поскольку запрашиваете данные с сервера, отличного от того, с которого исходит ваша страница. Браузер рассматривает все это как проблему безопасности.

Хм, а нельзя ли освежить в нашей памяти то, как браузер подходит к вопросам безопасности?



Джуди

Что за браузерная политика безопасности?

Да, с нашей стороны было стыдно налететь на такую «корягу», — достаточно лишь подумать о том, в какое положение при этом мы ставим вас, читателей, — но Джуди права, браузер задействует политику безопасности в случае с HTTP-запросами с использованием XMLHttpRequest, которая может вызывать проблемы.

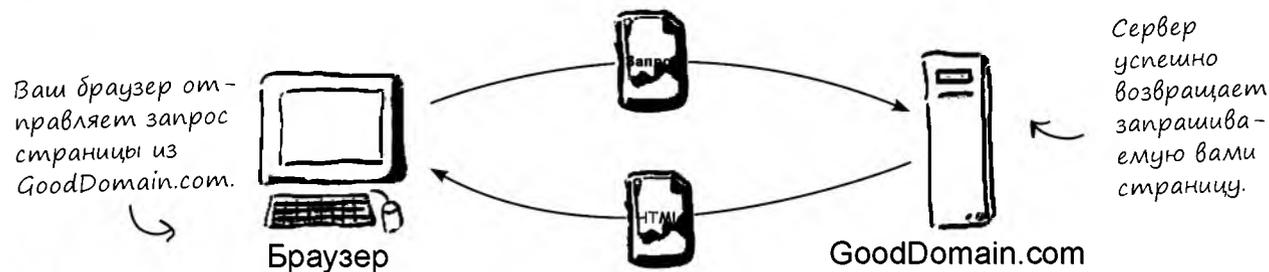
Так что это за политика такая? Это браузерная политика, которая подразумевает, что вы не сможете извлечь данные из домена, отличного от домена, из которого страница загружается как таковая. Допустим, вы создали сайт DaddyWarBucksBank.com и кто-то взломал вашу систему и внедрил JavaScript-код, который собирает персональные данные пользователей и делает с ними всевозможные интересные вещи, установив соединение с сервером HackersNeedMoreMoney.com. Звучит скверно, не так ли? Чтобы помешать подобным вещам, браузеры препятствуют совершению с использованием XMLHttpRequest HTTP-запросов в домены, отличные от исходного домена, из которого загружается ваша страница.

Посмотрим, какое поведение можно считать приемлемым в случае с JavaScript-кодом, а какое нет.



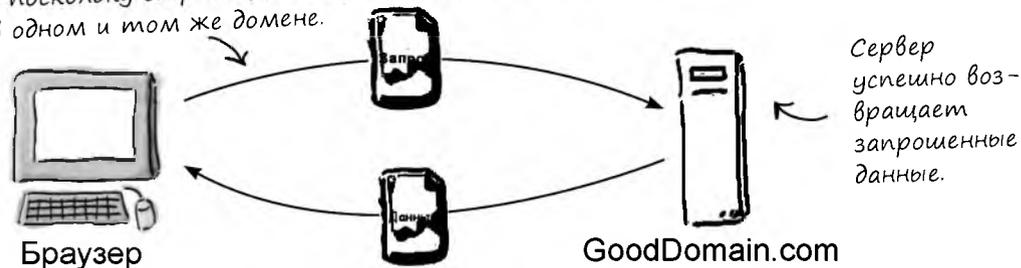
Приемлемое поведение в случае JavaScript-кода

- 1 Сначала пользователь (посредством браузера) совершает запрос HTML-страницы (и, конечно же, любых ассоциированных с ней JavaScript- и CSS-файлов):



- 2 Странице требуются некоторые данные из GoodDomain.com, поэтому она запрашивает их с использованием XMLHttpRequest:

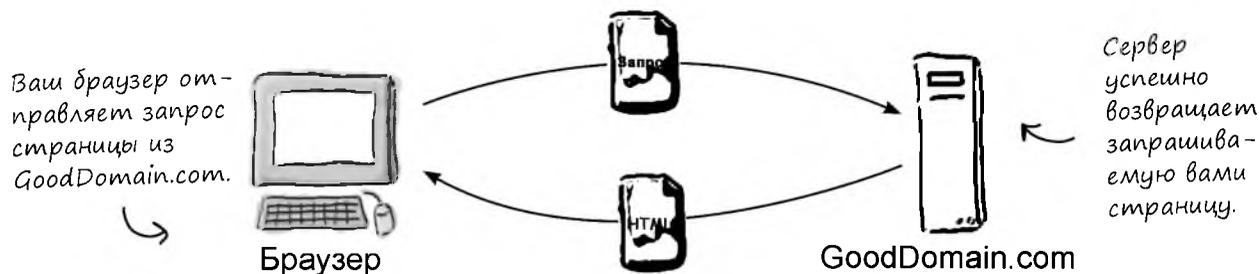
Данный запрос на извлечение данных из GoodDomain.com завершается успешно, поскольку страница и запрашиваемые данные располагаются в одном и том же домене.



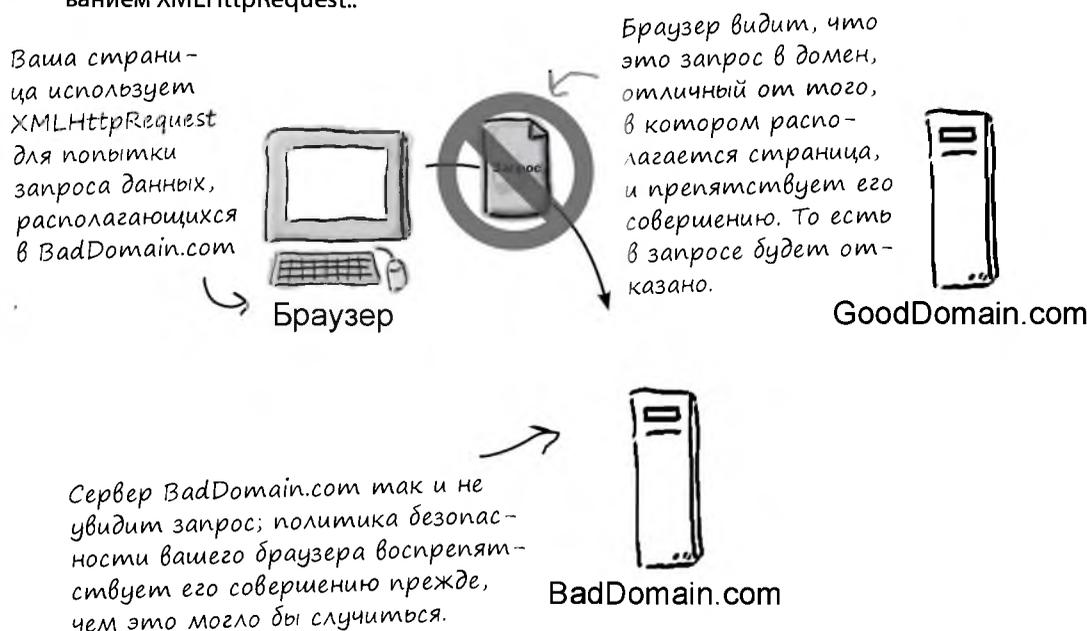
Неприемлемое поведение в случае JavaScript-кода:

Теперь посмотрим, что произойдет, когда ваша страница, располагающаяся в GoodDomain.com, нонягается загрузить данные с использованием XMLHttpRequest из BadDomain.com.

- 1 Как и ранее, браузер совершает запрос страницы, которая располагается в GoodDomain.com. Сюда могут входить JavaScript- и CSS-файлы, также располагающиеся в GoodDomain.com.



- 2 Однако на этот раз у нас имеется код, которому требуются данные из другого источника, то есть из BadDomain.com. Посмотрим, что произойдет, когда страница запросит эти данные с использованием XMLHttpRequest::



Да уж, славно потрудились — написали столько кода, а он так и не будет работать? А нельзя ли просто скопировать наши файлы на сервер Mighty Gumball?

Обычно ответ на подобный вопрос бывает утвердительным. Допустим, вы являетесь разработчиком, работающим над кодом для Mighty Gumball, а в этом случае у вас, скорее всего, будет доступ к серверу данной компании (или к людям, которые могут произвести развертывание файлов на соответствующем сервере за вас), где вы сможете разместить все свои файлы и избежать любых междоменных проблем. Однако в данном случае (хоть нам и неприятно разрушать вашу веру в правдоподобность излагаемой ситуации) вы на самом деле *не* работаете на Mighty Gumball, а являетесь читателями книги, и мы не можем дать возможность тысячам людей заниматься копированием своих файлов на сервер Mighty Gumball.

По крайней мере, не в рамках бюджета, выделенного нам редактором!



Так к чему же мы пришли? К тупиковой ситуации? Нет, у нас есть несколько вариантов, как можно ностунить в данном случае. Взглянем на них...





Какие у нас варианты?

Будем честны: все это время мы знали, что межисточниковый запрос XMLHttpRequest потерпит неудачу. Однако, как уже отмечалось ранее, при создании приложений вы зачастую будете располагать доступом к соответствующему серверу, так что это не станет проблемой (а при создании приложений, в значительной степени зависящих от ваших собственных данных, использование XMLHttpRequest обычно оказывается оптимальным выбором).

Однако в данный момент мы можем услышать от вас следующее: «Все это, конечно, здорово, но как, наконец, заставить наш код работать?». Что ж, есть два способа сделать это:

① Способ 1: использовать уже размещенные нами на сервере файлы.

Мы уже разместили для вас файлы на нашем сервере по адресу:

```
http://gumball.wickedlysmart.com/gumball/gumball.html
```

Проведите тестирование, введя данный URL в своем браузере, что позволит увидеть в действии код, который вы набирали до сих пор.

② Способ 2: использовать другой подход к извлечению данных.

XMLHttpRequest является отличным инструментом для извлечения данных в ваши приложения, если эти данные располагаются в том же домене, что и приложения. Но вдруг вам потребуется извлечь данные из стороннего источника? Что делать, если вам понадобятся данные, которые может дать, например, Google или Twitter? В подобных ситуациях перед нами встает проблема, которую нужно решить путем поиска иного подхода к извлечению данных.

Оказывается, есть другой подход, который основан на JSON и известен как JSONP (JSON with Padding — JSONP с подкладкой; согласны, звучит странно, однако поговорим об этом чуть позже). Надевайте свой реактивный ранец, поскольку способ его работы немного «с другой планеты», если вы понимаете, что мы имеем в виду.



Джо: Непременно! А что это такое?

Джим: Похоже, что другой подход к получению данных от веб-служб для передачи в наши приложения.

Фрэнк: От меня здесь не будет толку, поскольку я всего лишь дизайнер.

Джим: Фрэнк, я думаю, что все не так уж и плохо. Я быстро проверил с помощью Google, что такое JSONP, и узнал, что это, по сути, способ заставить тег `<script>` выполнять работу по извлечению данных.

Джо: А разве так можно?

Джим: Конечно можно — многие крупные сервисы поддерживают такой подход, например Twitter.

Фрэнк: Все это похоже на какой-то халтурный прием.

Джо: Да, и я о том же. Я имею в виду, как использование тега `<script>` может быть приемлемым подходом к извлечению данных? Я даже понять не могу, как он вообще работает.

Джим: Я сам еще далеко не полностью в этом разобрался. Но здесь можно рассуждать так: когда мы используем элемент `<script>`, он извлекает код для нас, ведь так?

Джо: Да, это так...

Джим: А что, если мы поместим данные в этот код?

Джо: Процесс пошел, все закрутилось-завертелось...

Фрэнк: Да, можно сказать, как хомяки в колесе...

Присаживайся,
Кузнечик. Зачастую то,
чему я учу, ты уже, в сущ-
ности, знаешь...



Гуру HTML5: ...и этот как раз один из таких случаев. Кузнечик, взгляни на этот код:

Что он делает?

```
alert("woof");
```

Веб-разработчик: Если произвести его оценку, предполагая при этом, что он выполняется в браузере, то данный код выведет диалоговое окно alert с текстом "woof".

Гуру: Да, верно. Создай свой собственный простой HTML-файл и помести в него `<script>`, расположив данный элемент в `<body>`, как показано далее:

Данный код располагается по этому URL-адресу.

```
<script src="http://wickedlysmart.com/hfhtml5/chapter6/dog.js">
</script>
```

Гуру: Что он делает?

Веб-разработчик: Загружает страницу, которая загружает с сервера `wickedlysmart.com` JavaScript-код, содержащийся в файле `dog.js` и вызывающий функцию `alert`, в результате чего браузер выводит диалоговое окно alert с текстом "woof".

Гуру: Получается, что JavaScript-файл, загружаемый из другого домена, может вызывать функцию в твоём браузере?

Веб-разработчик: Теперь, когда вы так выразились, да, Гуру, я полагаю, что именно так все и происходит. Как только файл `dog.js`, находящийся на сервере `wickedlysmart.com`, подвергается извлечению, он вызывает функцию `alert` в моем браузере.

Гуру: По адресу `http://wickedlysmart.com/hfhtml5/chapter5/dog2.js` располагается еще один файл со следующим JavaScript-кодом:

```
animalSays("dog", "woof");
```

Гуру: Что он делает?

Веб-разработчик: Он схож с кодом из файла `dog.js`, однако в данном случае вызову будет подвергаться функция `animalSays`. Кроме того, функция обладает не одним, а двумя аргументами: тип животного и звук, который оно издает.

Гуру: Напиши функцию `animalSays` и добавь ее в элемент `<script>` в `<head>` твоего HTML-файла над элементом `<script>`, указывающим на `wickedlysmart.com`.

Веб-разработчик: Вот так?

```
function animalSays(type, sound) {
    alert(type + " says " + sound);
}
```

Гуру: Очень хорошо, ты делаешь успехи. А теперь измени ссылку своего другого `<script>`, того, который указывает на `dog.js`, чтобы он стал указывать на `dog2.js`, и перезагрузи страницу в браузере.

Веб-разработчик: У меня на экране появилось диалоговое окно `alert` с сообщением "dog says woof".

Гуру: Теперь переключись на `http://wickedlysmart.com/hfhtml6/chapter5/cat2.js`, измени ссылку своего `<script>`, чтобы он стал указывать на `cat2.js`, и посмотри, что будет.

```
animalSays("cat", "meow");
```

Веб-разработчик: На экране появилось диалоговое окно `alert` с сообщением "cat says meow".

Гуру: Получается, что JavaScript-файл, загружаемый из другого домена, может не только вызывать любую функцию в твоём коде, какая ему будет необходима, но и передавать нам любые данные, какие ему потребуются?

Веб-разработчик: Вообще-то я не вижу никаких данных, только два аргумента.

Гуру: А разве аргументы не являются данными? Что, если внести изменение, в результате чего аргументы станут выглядеть так:

```
var animal = {"type": "cat", "sound": "meow"};
animalSays(animal);
```

← `cat3.js`

Веб-разработчик: Теперь функции `animalSays` передается один аргумент, который является объектом. Хм, несомненно, в моих глазах этот объект начинает становиться похожим на данные.

Гуру: Ты можешь переписать функцию `animalSays` таким образом, чтобы она использовала новый объект?

Веб-разработчик: Попробую...

Веб-разработчик: Вот так?

```
function animalSays(animal) {
    alert(animal.type + " says " + animal.sound);
}
```

Гуру: Очень хорошо. Измени ссылку на <http://wickedlysmart.com/hfhtml5/chapter6/dog3.js> и проведи тест. Сделай то же самое с использованием <http://wickedlysmart.com/hfhtml5/chapter6/cat3.js>.

Веб-разработчик: Да, оба варианта работают, как и ожидалось в случае с моей новой функцией.

Гуру: А что, если изменить имя `animalSays` на `updateSales`?

Веб-разработчик: Гуру, я не понимаю, какая может быть связь между животными и уровнями продаж жевательной резинки?

Гуру: Давай поработаем сообща. Что, если мы переименуем `dog3.js`, присвоив ему имя `sales.js`, и перепишем код следующим образом:

```
var sales = [{"name":"ARTESIA","time":1308774240669,"sales":8},
             {"name":"LOS ANGELES","time":1308774240669,"sales":2}];
updateSales(sales);
```

Веб-разработчик: Кажется, я начинаю понимать. Мы передаем данные посредством JavaScript-файла, на который ссылаемся, вместо того, чтобы самим извлекать их с использованием `XMLHttpRequest`.

Гуру: Да. Однако ты должен уметь видеть главное за массой второстепенных деталей. Разве при этом их извлечение осуществляется не из другого домена? А это `XMLHttpRequest` запрещает.

Веб-разработчик: Кажется, так оно и есть. Действительно похоже на какое-то волшебство.

Гуру: Здесь нет никакого волшебства, поскольку элемент `<script>` всегда так работал. Ответ все время был рядом. Теперь поразмышляй над тем, как все это функционирует, чтобы закрепить материал.

Веб-разработчик: Да, учитель. «Закрепить материал»... знаете, эта фраза звучит для меня так знакомо, но в то же время как-то непривычно.



АЗЕН-МОМЕНТ

Использование JavaScript для извлечения данных является тем, с чем вам необходимо слиться воедино. Возьмите лист бумаги или используйте внутреннюю сторону обложки этой книги. Нарисуйте сервер, на котором располагаются ваши HTML- и JavaScript-файлы. Также изобразите сервер в другом домене, где размещаются файлы `dog3.js` и `cat3.js`. Затем проделайте все шаги, которые браузер предпринимает для извлечения и использования объекта в каждом из файлов. После того как вы во всем разберетесь, мы снова проделаем эти шаги вместе.

Знакомство с JSONP

Вы, вероятно, уже поняли, что JSONP представляет собой способ извлечения JSON-объектов с использованием тега `<script>`. Это также способ извлечения данных (онячь-таки, в форме JSON-объектов), позволяющий избежать проблем безопасности, связанных с политикой одного источника, которые мы видели в случае с XMLHttpRequest.

На протяжении следующих нескольких страниц мы с вами пошагово пройдемся по тому, как работает JSONP:

Браузер

```

<!doctype html>
<html lang="en">
  ...
<body>
  <h1>Mighty Gumball sales</h1>
  <div id="sales">
  </div>
  <script src="http://gumball.wickedlysmart.com/"></script>
</body>
</html>
    
```

1 Мы включили элемент `<script>` в свою HTML-разметку. Источником для этого элемента выступает URL веб-службы, которая будет обеспечивать для нас JSON-данные об уровне продаж жевательной резинки.

2 Когда браузер сталкивается с элементом `<script>` на странице, он отправляет HTTP-запрос по URL-адресу источника.

4 JSON-ответ поступает в форме строки, которую браузер разбирает и интерпретирует. Любые типы данных преобразуются в подлинный JavaScript-объект и значения, а любой код подвергается выполнению.

Помните, что на данный момент это всего лишь строковое представление объекта!

3 Сервер воспринимает данный запрос как HTTP-запрос и отправляет в ответ на него JSON.

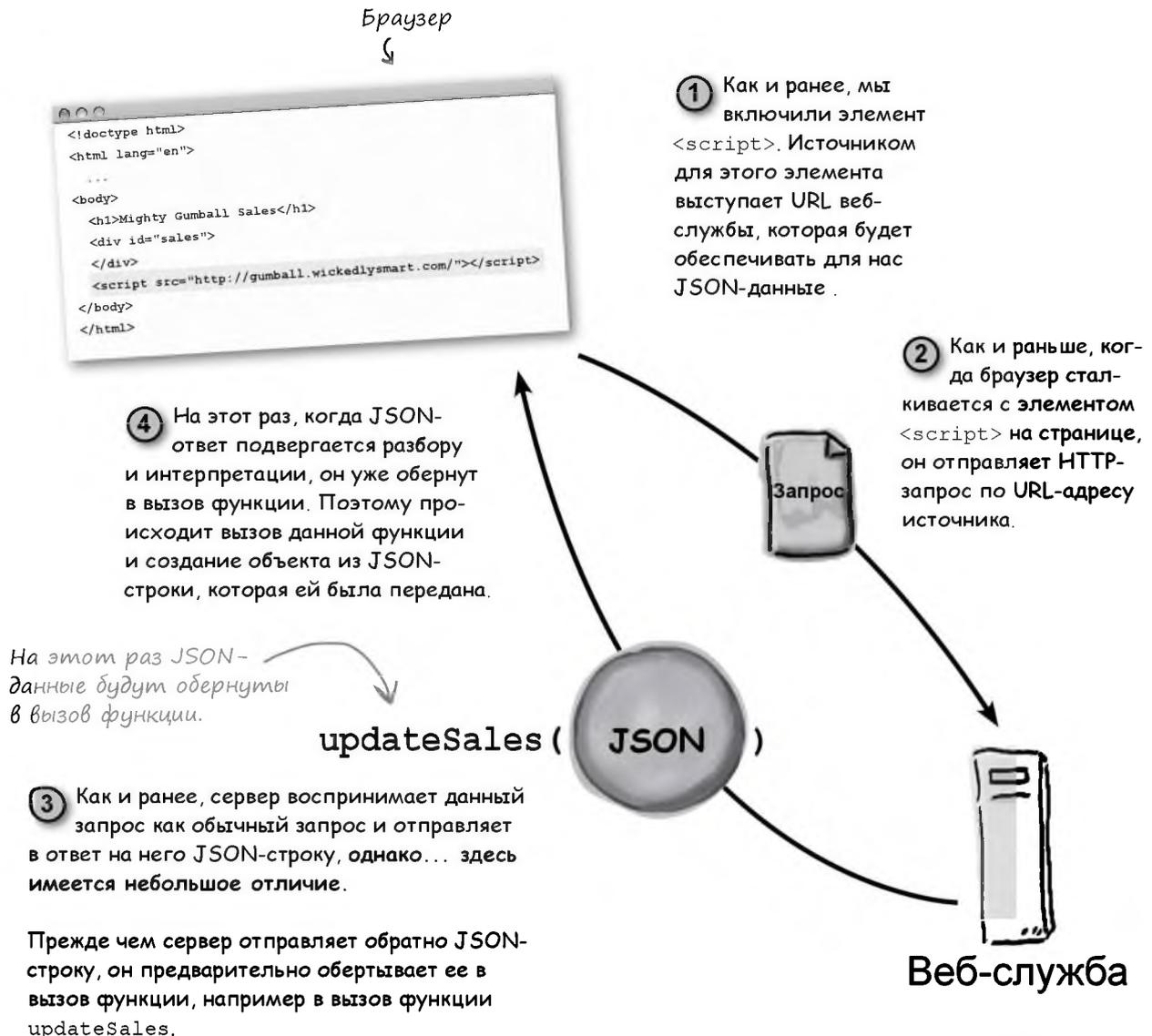


Веб-служба

Что означает буква P в аббревиатуре JSONP?

Первое, что вам необходимо знать о JSONP, — это то, что у него глупое и не очень понятное имя: JSON with Padding (JSONP с подкладкой). Если бы мы давали ему имя, то называли бы его как-то вроде JSON with Callback (JSONP с функцией обратного вызова), либо «извлеки мне JSON-данные и выполни код, когда вернешь их», либо просто как-то по-другому вместо JSON with Padding.

Однако все, что здесь подразумевается под Padding, — это обертывание JSON-данных в вызов функции, прежде чем они ностунают в ответ на запрос. Вот как все это работает:





Я понимаю, как использовать тег `<script>` для того, чтобы заставить браузер извлекать JavaScript, и как сервер может помещать свои данные в этот JavaScript. Но как быть с именем функции? Откуда веб-служба знает имя нужной функции? Например, откуда веб-службе Mighty Gumball известно, что необходимо вызывать именно `updateSales`? Вдруг у меня имеется другая служба и я хочу вызывать, допустим, функцию `updateScore`, или `alert`, или какую-то другую?

Веб-службы позволяют указывать имя требуемой функции обратного вызова.

Обычно веб-службы позволяют указывать имя необходимой вам функции. Хотя мы и не говорили этого, но Mighty Gumball предоставляет такую возможность. Когда вы будете указывать соответствующий URL-адрес, добавьте в его конец параметр, как показано дальше:

`http://gumball.wickedlysmart.com/?callback=updateSales`

↑
Обычный URL-адрес, который мы уже использовали раньше.

А здесь мы добавили URL-параметр `callback`, который указывает, что при генерировании JavaScript должна использоваться функция `updateSales`.

В результате Mighty Gumball будет использовать `updateSales` для обертывания объекта в формате JSON до того, как отправлять его обратно вам. В случае с веб-службами данный параметр обычно называется `callback`, однако для подстраховки вам все же следует изучить документацию к используемой вами веб-службе.



Введите данные URL-адреса: что вы видите в ответ?

`http://search.twitter.com/search.json?q=hfhtml5&callback=myCallback`

`http://search.twitter.com/search.json?q=hfhtml5&callback=justDoIt`

`http://search.twitter.com/search.json?q=hfhtml5&callback=updateTweets`

Примечание: браузер Firefox предложит вам открыть либо сохранить файл. Для открытия вы можете использовать TextEdit, Блокнот или любой другой базовый текстовый редактор.

Ребята, у нас все получилось. Нам пришлось немного повозиться, чтобы разобраться с использованием элемента `<script>` при обращении к веб-службе, однако теперь такой подход кажется едва ли не более простым, чем использование XMLHttpRequest.



Джим: Что ж, почти.

Джо: Думаю, это дает нам возможность избавиться от части кода.

Фрэнк: Я готов придать веб-приложению красивый внешний вид, когда вы закончите.

Джим: Джо, а что ты думаешь?

Джо: В случае с XMLHttpRequest мы извлекали строку. Но при использовании JSONP тег `<script>` обеспечивает разбор и оценку возвращаемого кода, поэтому когда мы получим данные, они уже будут представлять собой JavaScript-объект.

Джим: Да, все верно, а в случае с XMLHttpRequest мы использовали `JSON.parse` для преобразования строки в объект. Получается, что теперь мы можем избавиться от этого кода?

Джо: Да. Это мое мнение, которого я придерживаюсь.

Джим: Что еще?

Джо: Что ж, очевидно, что нам потребуется вставить элемент `<script>`.

Джим: Как раз это меня и интересовало. Где мы его разместим?

Джо: Браузер будет определять, когда он будет загружаться, а нам необходимо, чтобы сначала загружалась страница, чтобы мы могли обновить объектную модель документа (DOM) при вызове `updateSales`. Единственный выход, как мне кажется, в данном случае заключается в размещении `<script>` в нижней части страницы, в `<body>` HTML-документа.

Джим: Да, нохоже, это здравая мысль. Нам следует подробнее на ней остановиться. Но сначала давайте проверим ее на практике.

Джо: Я хочу, чтобы наш код, наконец, заработал! Давайте им займемся!

Фрэнк: Ребята, вам лучше не мешать, несколько, готов спорить, Джуди уже нашла свой способ заставить все работать.

Обновление кода веб-приложения Mighty Gumball

Пора обновить код нашего приложения Mighty Gumball с использованием JSONP. Помимо удаления существующего кода, связанного с вызовом XMLHttpRequest, все остальные изменения будут незначительными.

Вот что нам потребуется сделать:

- 1 Удалить код XMLHttpRequest.
- 2 Удостовериться, что функция updateSales будет готова к приему объекта, а не строки (как было в случае с XMLHttpRequest).
- 3 Добавить элемент `<script>` для выполнения работы по извлечению данных.

-
- 1 Весь код в нашей функции onload является связанным с XMLHttpRequest, поэтому мы можем просто удалить его. Сохраним функцию onload на случай, если она пригодится нам немного позже. А пока она ничего не будет делать. Откройте свой файл mightygumball.js и внесите следующие изменения:

```
window.onload = function() {  
    var url = "http://gumball.wickedlysmart.com";  
    var request = new XMLHttpRequest();  
    request.open("GET", url);  
    request.onload = function() {  
        if (request.status == 200) {  
            updateSales(request.responseText);  
        }  
    }  
    request.send(null);  
}
```

В данный момент вам необходимо просто удалить весь код, имеющийся в этой функции.



- ② Как вы помните, когда мы используем элемент `<script>`, мы говорим браузеру, что нужно извлечь JavaScript, в результате чего браузер извлекает его, а затем проводит его разбор и оценку. Это означает, что к моменту, когда он дойдет до вашей функции `updateSales`, JSON будет представлять собой уже не строку, а полноценный JavaScript-объект. Когда мы использовали `XMLHttpRequest`, данные возвращались в форме строки. На данный момент функция `updateSales` предполагает, что будет получать строку, поэтому давайте внесем здесь изменение, чтобы она обрабатывала объект, а не строку:

```

function updateSales(responseText) {
function updateSales(sales) {
    var salesDiv = document.getElementById("sales");
    var sales = JSON.parse(responseText);
    for (var i = 0; i < sales.length; i++) {
        var sale = sales[i];
        var div = document.createElement("div");
        div.setAttribute("class", "saleItem");
        div.innerHTML = sale.name + " sold " + sale.sales + " gumballs";
        salesDiv.appendChild(div);
    }
}

```

Удаляем `responseText` и переписываем строку с использованием параметра `sales`.

Кроме того, мы можем удалить вызов `JSON.parse`.

А вот и результат: теперь у нас имеется функция, готовая к обработке наших данных.

- ③ И наконец, добавляем элемент `<script>` для выполнения работы по извлечению данных.

```

<!doctype html>
<html lang="en">
<head>
  <title>Mighty Gumball</title>
  <meta charset="utf-8">
  <script src="mightygumball.js"></script>
  <link rel="stylesheet" href="mightygumball.css">
</head>
<body>
  <h1>Mighty Gumball Sales</h1>
  <div id="sales">
  </div>
  <script src="http://gumball.wickedlysmart.com/?callback=updateSales"></script>
</body>
</html>

```

Это ссылка на веб-службу `Mighty Gumball`. Мы используем параметр `callback` и указываем имя нашей функции `updateSales`, в силу чего веб-служба будет оберты-вать JSON-данные в вызов функции `updateSales`.

Тест-драйв нового кода с JSONP



Когда вы внесете все изменения, наступит время провести тест-драйв. Перезагрузите `mightygumball.html` в своем браузере. Теперь вы загружаете данные об уровне продаж жевательной резинки автоматами Mighty Gumball, используя свое веб-приложение и JSONP. Внешне страница должна получиться такой же, как и при извлечении данных об уровне продаж из локального файла, но вы будете знать, что в данном случае используется совершенно другой способ извлечения информации.

Вот что вы увидите при перезагрузке страницы Mighty Gumball. Вы будете наблюдать разные города и уровни продаж, поскольку на экран станут выводиться реальные данные.



Да! Исполнительному директору Mighty Gumball это должно понравиться! Теперь можно и отдохнуть.



Отличная работа, парни.

А мне кажется, что JSONP является одной большой брешью в безопасности!

Данный подход не менее безопасен, чем использование `<script>` для загрузки JavaScript.

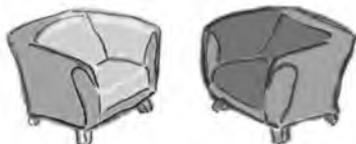
Это правда: если вы совершаете JSONP-запрос, обращаясь к вредоносной веб-службе, то ответ может содержать JavaScript-код, которого вы не ожидаете, и браузер выполнит его.

Все это ничем не отличается от добавления JavaScript нутем связывания с библиотеками, размещаемыми на других серверах. Всякий раз, когда вы осуществляете связывание с JavaScript-библиотекой (например, с относящейся к `<head>` вашего документа) либо используете JSONP, вам необходимо быть уверенными в том, что вы доверяете данной службе. А если вы пишете веб-приложение, которое станет задействовать аутентификацию при решении вопроса о том, давать ли доступ конкретному пользователю к чувствительным данным, то наилучшим выходом для вас, вероятно, будет вообще не использовать сторонние библиотеки или JSON-данные, размещаемые на других серверах.

Поэтому осторожно подходите к выбору веб-служб, к которым собираетесь подключаться. Если вы используете API-интерфейс, такой как Google, Twitter, Facebook, или одну из множества других широко известных веб-служб, то можете быть уверены в их безопасности. В противном случае рекомендуется проявлять осторожность. В нашем случае мы знакомы с инженерами из компании Mighty Gumball и знаем, что они никогда не стали бы внедрять что-либо вредоносное в свои JSON-данные, в силу чего вы можете быть уверенными в их безопасности.



Беседа у камина



Участники: XMLHttpRequest and JSONP

Сегодня мы станем свидетелями разговора двух популярных способов извлечения данных в случае с браузерами.

XMLHttpRequest:

Не хочу Вас обидеть, но не являетесь ли вы халтурным приемом? Я имею в виду, что Ваше назначение заключается в извлечении кода, а люди используют Вас для совершения запросов данных.

Но все, что Вы делаете, — это вбрасывание данных вместе с кодом. И вообще вы никак не позволяете совершать запросы напрямую из JavaScript-кода; Вы вынуждаете людей использовать HTML-элемент `<script>`. Это приводит Ваших пользователей в замешательство.

Эй, XML по-прежнему широко используется, и не стоит умалять его значение. А извлекать JSON-данные можно и с помощью меня без проблем.

Я позволяю контролировать, какие именно данные будут подвергаться разбору и преобразованию в JavaScript-объекты. А в случае с Вами это происходит в отношении всех данных.

Что ж, можно воспользоваться халтурным приемом вроде JSON with Padding (ну и глупое же название) либо правильным способом, то есть XMLHttpRequest, и «расти» вместе с ним по мере его эволюции. В конце концов, люди работают над тем, чтобы сделать меня более гибким, но в то же время безопасным.

JSONP:

Халтурным? Я бы назвал себя замечательным. Ведь я могу использоваться для извлечения как кода, так и данных. Какая необходимость в том, чтобы делать все это двумя разными способами?

Эй, но ведь такой подход работает и дает людям возможность писать код, который позволяет получать JSON-данные от сервисов вроде Twitter, Google и других. Как такое возможно при использовании Вас, XMLHttpRequest, учитывая налагаемые Вами ограничения с целью обеспечения безопасности? Я имею в виду, что Вы все еще цепляетесь за прошлое, за XML.

Конечно можно, если кому-то нравится постоянно заниматься преобразованием результатов с помощью `JSON.parse`.

А в этом как раз и заключается преимущество — к моменту, когда данные дойдут до моих пользователей, все они будут разобраны и преобразованы. Поймите, я Вас очень уважаю, Вы стояли у истоков целого подхода к написанию приложений, но проблема в том, что Вы навязываете слишком много ограничений. В современном мире веб-служб людям требуется возможность совершать запросы и в другие домены.

Люди, несомненно, работают над вашим улучшением, однако у моих пользователей имеются реальные потребности уже сегодня, — они не могут ждать, пока будут устранены все ваши междоменные проблемы.

XMLHttpRequest:

У меня нет ничего общего с названием Ajax, поэтому не надо задавать мне такие вопросы! Кстати, Вы ни разу не говорили о том, насколько Вы безопасны.

Я могу лишь сказать, что если вам не требуется извлекать чужие данные, которые может дать, например, Twitter или Google, и вы создаете свою собственную веб-службу и клиента, то выбирайте меня. Я обеспечиваю более высокий уровень безопасности и более прост в использовании.

Да, да, знаю я об этом смешивании данных, извлекаемых с различных сторонних веб-ресурсов.

Да ладно, не так уж и много кода нужно, чтобы обеспечить поддержку меня версиями вплоть до Internet Explorer 5.

Что ж, однако, это еще не все. Пробовали ли Вы когда-нибудь делать нечто повторяющееся, то есть когда требуется извлекать что-либо снова и снова? В качестве примера можно привести приложение Mighty Gumball, над которым сейчас идет работа. Как разработчики смогут обеспечить такую функциональность, используя Вас?

Мне кажется, Ваши читатели, услышав только что сказанное Вами, переспросят: «Что, простите?»

JSONP:

И ничего глупого в слове Padding в названии JSONP нет, поскольку оно всего лишь означает, что когда пользователь совершает запрос веб-службы, он при этом также просит ее добавить небольшой префикс, например, "updateSales()", к результату. А Вас как иногда называют? Ajax? Разве это не название средства чистки для ванн?

Программистам постоянно приходится проявлять осторожность. Если вы извлекаете код с другого сервера, то должны знать, что делаете. Но ответ в данном случае не должен просто ограничиваться фразой «этого не следует делать».

Алло? Никто не создает службы, которые не используют внешние данные. Слышали когда-нибудь термин «мэшап»?

Эй, я, по крайней мере, обладаю стабильной повсеместной поддержкой и терпеть не могу, когда приходится писать код XMLHttpRequest, который сможет работать в старых браузерах.

Ха-ха, а в случае со мной для этого вообще не потребуются НИКАКОГО кода. Лишь простой HTML-тег.

Эй, все не настолько плохо. Нужно просто добавить новый элемент `<script>` в объектную модель документа (DOM) для совершения еще одного запроса.

HEAD FIRST:

Благодарю, ребята! Боюсь, наше время истекло!

Вы немного не доработали приложение. Я думал, что увижу постоянно обновляющийся поток данных об уровне продаж, поступающий от наших автоматов для продажи жевательной резинки. Конечно, я могу нажать кнопку Refresh (Обновить) в браузере, но ведь получается, что я смогу увидеть самые свежие отчеты только в том случае, если сам вручную нажму кнопку обновления. Это не то, чего я хотел!



МОЗГОВОЙ ШТУРМ

Он прав, нам необходимо подкорректировать приложение, чтобы оно обновляло отображаемые данные, используя новые сведения об уровне продаж, через регулярные интервалы времени (например, каждые 10 секунд). На данный момент у нас в странице имеется элемент `<script>`, который инициирует отправку запроса на сервер только один раз. Вы можете придумать какой-нибудь способ так использовать JSON, чтобы обеспечивалось непрерывное извлечение отчетов с новыми данными об уровне продаж?

Подсказка: используя объектную модель документа (DOM), мы можем вставить новый элемент `<script>` в страницу. Поможет ли это?

Ребята, как я только что узнала, исполнительный директор Mighty Gumball не совсем доволен вашей первой версией приложения?



Джим: Да, он хочет, чтобы отображаемые на странице данные обновлялись непрерывно.

Джуди: В этом есть смысл. Я имею в виду, что большим преимуществом веб-приложения является то, что его не нужно обновлять с помощью кнопки Refresh (Обновить) как обычную веб-страницу.

Джо: Вполне справедливое требование. Кроме того, мы уже знаем, как заменять старые данные об уровне продаж новыми на странице, используя DOM. Но мы пока не уверены в том, как именно следует обрабатывать JSONP-часть.

Джуди: Не забывайте, что вы также можете использовать объектную модель документа в сочетании с элементом `<script>`. Другими словами, вы можете создавать новый элемент `<script>` в DOM всякий раз, когда необходимо извлечь дополнительные данные.

Джим: Что-то я не очень понял. Нельзя ли новторить?

Джо: Кажется, я разобрался. На данный момент мы статически размещаем элемент `<script>` в HTML-разметке, просто набрав его там. Вместо этого мы могли бы создавать новый элемент `<script>` с использованием JavaScript-кода и добавлять его в DOM. Единственное, в чем я здесь не уверен, — станет ли браузер еще раз извлекать данные, когда мы создадим новый элемент `<script>`?

Джуди: Конечно, станет.

Джим: Ясно, то есть мы будем создавать новый элемент `<script>` каждый раз, когда нам будет необходимо, чтобы браузер выполнил для нас JSONP-операцию.

Джуди: Правильно! Похоже, что и ты во всем разобрался. А знает ли кто-нибудь из вас, как сделать так, чтобы это происходило снова и снова?

Джим: Мы еще не дошли до этого, мы пока что размышляем о JSONP.

Джуди: Вам уже все известно о функциях обработчиков событий вроде `onload` или `onclick`. Вы можете установить таймер для вызова функции обработчика через заданный интервал времени с использованием метода `setInterval` в JavaScript.

Джо: Давайте именно так и поступим и сделаем JSONP динамическим, чтобы как можно скорее представить исполнительному директору Mighty Gumball должным образом работающее приложение.

Джим: О, это все, что потребуется сделать? Тогда не будем медлить!

Дорабатываем приложение Mighty Gumball

Как видите, нам придется вынудить немного дополнительной работы, однако она не будет слишком сложной. По сути, мы написали нервую версию нашего приложения таким образом, что оно извлекает отчеты об уровне продаж с сервера Mighty Gumball и отображает их только *один раз*. Это наше улучшение, поскольку почти все современные веб-приложения должны непрерывно осуществлять мониторинг данных и обновлять отображаемые на странице сведения (ночти) в режиме реального времени.

Вот что нам потребуется сделать:

- 1 Мы удалим JSONP-элемент `<script>` из HTML-страницы Mighty Gumball, поскольку больше не будем его использовать.
- 2 Нам потребуется задать обработчик для обработки совершения JSONP-запроса каждые несколько секунд. Прислушаемся к совету Джуди и воспользуемся JavaScript-методом `setInterval`.
- 3 Затем необходимо реализовать наш JSONP-код в обработчике, чтобы каждый раз, когда будет происходить его вызов, он совершал запрос с целью извлечения отчетов с самыми свежими данными об уровне продаж с сервера Mighty Gumball.

Шаг 1: угаляем элемент `<script>`...

Мы собираемся использовать новый подход к инициированию JSONP-запросов, поэтому давайте удалим элемент `<script>` из HTML-разметки.

```
<!doctype html>
<html lang="en">
<head>
  <title>Mighty Gumball</title>
  <meta charset="utf-8">
  <script src="mightygumball.js"></script>
  <link rel="stylesheet" href="mightygumball.css">
</head>
<body>
  <h1>Mighty Gumball Sales</h1>
  <div id="sales">
  </div>
  <script src="http://gumball.wickedlysmart.com/?callback=updateSales"></script>
</body>
</html>
```

Удалите данный элемент из своего HTML-файла.

Шаг 2: время для установки таймера



Итак, мы стремимся к тому, чтобы вместо извлечения отчетов об уровне продаж только один раз эта процедура осуществлялась время от времени, скажем, каждые 3 секунды. Это может оказаться слишком часто или редко в зависимости от конкретного приложения, однако в случае с Mighty Gumball мы начнем с интервала 3 секунды.

Нам потребуется функция, которую мы сможем вызывать каждые 3 секунды. И, как говорила Джуди, для этого мы можем воспользоваться методом `setInterval`, который имеется в объекте `window`:

```
setInterval(handleRefresh, 3000);
```

↑
Метод `setInterval` принимает обработчик и значение интервала времени.

↑
Наша функция обработчика, которую мы определим чуть позже.

↑
Наше значение интервала времени, выраженное в миллисекундах. 3000 миллисекунд = 3 секунды.

Таким образом, каждые 3000 миллисекунд JavaScript будет вызывать обработчик — в данном случае функцию `handleRefresh`. Давайте напишем простой обработчик и протестируем его:

```
function handleRefresh() {
    alert("I'm alive");
}
```

← При каждом вызове данного обработчика (а происходит это будет каждые 3 секунды) на экран будет выводиться диалоговое окно `alert` с сообщением «I'm alive».

Теперь нам потребуется код для задания вызова `setInterval`, который мы добавим в функцию `onload`, чтобы он происходил сразу после того, как страница полностью загрузится:

```
window.onload = function() {
    setInterval(handleRefresh, 3000);
}
```

← Это наша прежняя функция `onload`, в которой ничего не осталось после того, как мы удалили из нее код `XMLHttpRequest`.

↑ Все, что нам необходимо сделать, — это добавить наш вызов `setInterval`, который при выполнении функции `init` запустит таймер, который, в свою очередь, станет инициализироваться каждые 3 секунды и вызывать нашу функцию `handleRefresh`.

Проведем тест-драйв, а затем, когда убедимся, что все работает (то есть когда мы увидим, что обработчик вызывается каждые 3 секунды), реализуем JSONP-код.

Тест-драйв таймера



Это будет весело. Убедитесь, что вы набрали функцию `handleRefresh`, а также внесли изменения в обработчик `onload`. Сохраните все, а затем загрузите код в своем браузере. В результате вы должны увидеть поток открывающихся диалоговых окон `alert`, остановить который сможете, лишь закрыв окно браузера!



Вот что мы получим!

Возьми в руку карандаш



Теперь, когда вы знаете, что такое `setInterval` (не говоря уже о `XMLHttpRequest` и `JSONP`), задумайтесь над тем, для чего еще вы могли бы использовать их в других веб-приложениях. Напишите свои ответы здесь.

Для проверки и обновления показателей хода выполнения задачи с выводом их на экран.

Для проверки на предмет новых комментариев, размещаемых пользователями.

Для обновления карты пользователя в случае появления поблизости его друзей.

.....
.....
.....



Шаг 3: повторная реализация JSONP

Мы по-прежнему хотим использовать JSOP для извлечения данных, однако нам необходим способ делать это при каждом вызове обработчика `handleRefresh`, а не только во время загрузки страницы. Именно здесь на сцену выходит объектная модель документа. Ее замечательная особенность заключается в том, что мы можем вставлять элементы в DOM в любой момент, даже элементы `<script>`. Таким образом, мы должны иметь возможность вставлять новый элемент `<script>` всякий раз, когда захотим совершить JSONP-вызов. Давайте напишем соответствующий код, воспользовавшись всеми нашими знаниями о DOM и JSONP.

Сначала зададим URL-адрес JSONP

Это будет тот же самый URL-адрес, который мы использовали с нашим предыдущим элементом `<script>`. Здесь мы присвоим его переменной для последующего использования. Удалите `alert` из своего обработчика и добавьте вот этот код:

```
function handleRefresh() {
  var url = "http://gumball.wickedlysmart.com?callback=updateSales";
}
```

Снова возвращаемся к нашей функции handleRefresh..

Задаем URL-адрес JSONP и присваиваем его переменной url.

Далее создадим новый элемент `<script>`

На этот раз вместо имеющегося элемента `<script>` в составе нашей HTML-разметки, мы будем создавать такой элемент с использованием JavaScript. Нам потребуется создать элемент `<script>`, а затем задать соответствующие значения для его атрибутов `src` и `id`:

```
function handleRefresh() {
  var url = "http://gumball.wickedlysmart.com?callback=updateSales";

  var newScriptElement = document.createElement("script");
  newScriptElement.setAttribute("src", url);
  newScriptElement.setAttribute("id", "jsonp");
}
```

Сначала мы создаем новый элемент `<script>`...

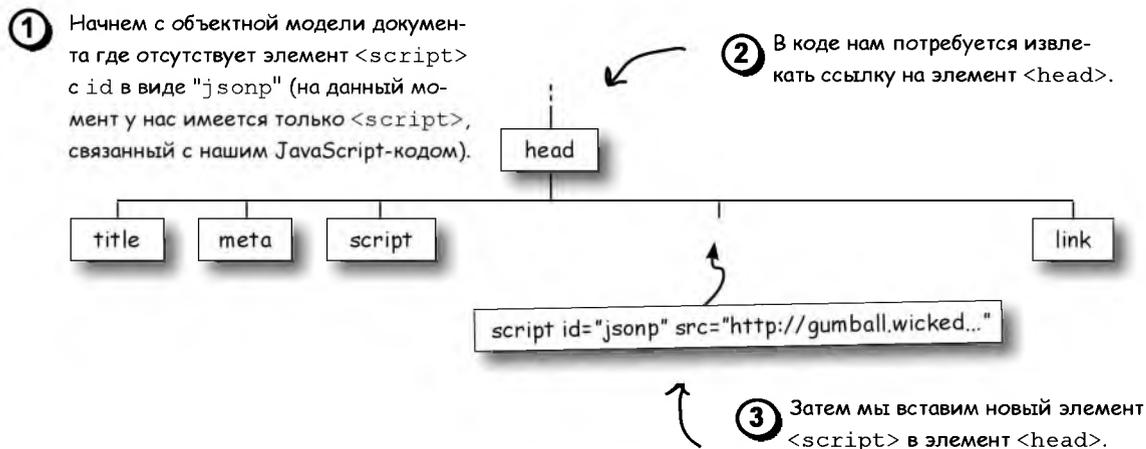
...а затем присваиваем его атрибуту `src` значение в виде URL-адреса JSONP.

Мы также присвоим данному элементу `<script>` идентификатор, чтобы его можно было без труда найти позже, что, как вы увидите, нам и потребуется сделать.

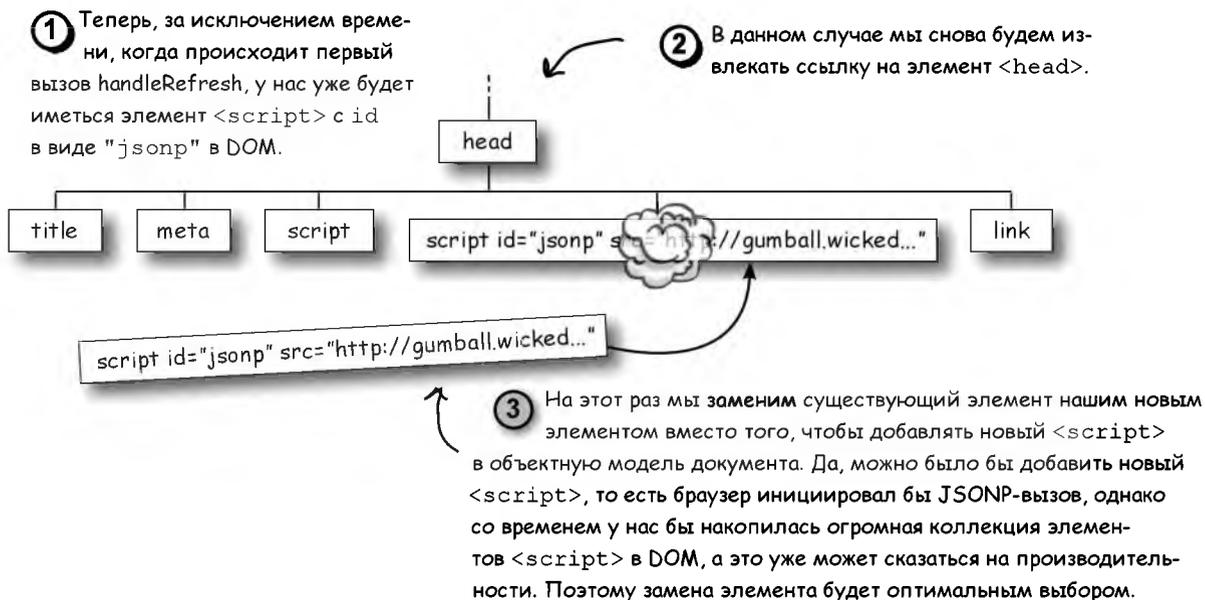
Метод `setAttribute` может показаться вам чем-то новым (мы лишь мимоходом упоминали его ранее), однако несложно понять, что он делает. Данный метод позволяет задавать значения для атрибутов HTML-элементов (например, для `src` и `id`), а также для множества других, включая `class`, `href` и т. д.

Как мы будем вставлять элемент `<script>` в DOM??

Мы почти достигли цели, осталось только произвести вставку нашего нового элемента `<script>`. Как только мы сделаем это, браузер увидит его и предпримет соответствующие действия, что приведет к совершению JSONP-вызова. Вставка элемента `<script>` требует небольшого планирования и предусмотрительности; давайте посмотрим, как все это работает:



После того как мы вставим `<script>`, браузер увидит этот новый элемент в объектной модели документа (DOM) и отправится извлекать то, что расположено по URL-адресу, который задан в качестве значения атрибута `src`. Однако у нас также имеется второй вариант использования. Взглянем на него.



Теперь напишем код для вставки `<script>` в DOM

Теперь, когда вы знаете необходимые шаги, давайте взглянем на код. Здесь мы тоже выполним два шага: сначала покажем вам код для добавления нового `<script>`, а затем продемонстрируем код для замены `<script>`:

```
function handleRefresh() {
    var url = "http://gumball.wickedlysmart.com?callback=updateSales";

    var newScriptElement = document.createElement("script");
    newScriptElement.setAttribute("src", url);
    newScriptElement.setAttribute("id", "jsonp");

    var oldScriptElement = document.getElementById("jsonp");
    var head = document.getElementsByTagName("head")[0];
    if (oldScriptElement == null) {
        head.appendChild(newScriptElement);
    }
}
```

Сначала мы будем извлекать ссылку на элемент `<script>`. В случае его отсутствия будет возвращено `null`. Обратите внимание, что мы рассчитываем на то, что он будет иметь `id` со значением «jsonp».

Далее будем извлекать ссылку на элемент `<head>`, используя новый метод объекта `document`. О том, как он работает, мы поговорим позже, а пока просто знайте, что он извлекает ссылку на элемент `<head>`.

Теперь, когда у нас есть ссылка на элемент `<head>`, мы проверяем, имеется ли там уже элемент `<script>`, и если он отсутствует (о чем будет свидетельствовать возврат значения `null` при попытке извлечь на него ссылку), мы добавим новый элемент `<script>` в `<head>`.

Итак, взглянем на код для замены элемента `<script>` в случае, если выясняется, что таковой уже существует. Мы рассмотрим только условный оператор `if`, в котором сосредоточен весь новый код:

```
if (oldScriptElement == null) {
    head.appendChild(newScriptElement);
} else {
    head.replaceChild(newScriptElement, oldScriptElement);
}
```

Наш условный оператор, который, как вы помните, проверяет, существует ли уже элемент `<script>` в DOM.

Если в `<head>` уже имеется элемент `<script>`, мы просто заменяем его. Мы используем метод `replaceChild` в отношении `<head>` и передаем ему `newScriptElement` и `oldScriptElement` для выполнения данной процедуры. Через несколько мгновений мы подробнее поговорим об этом методе.



getElementsByTagName под увеличительным стеклом

Это было ваше первое знакомство с методом `getElementsByTagName`, поэтому быстро исследуем его подробнее. Он подобен методу `getElementById` за исключением того, что возвращает массив элементов, соответствующих определенному имени тега.

getElementsByTagName возвращает все элементы, соответствующие данному тегу, которые имеются в DOM.

```
var arrayOfHeadElements = document.getElementsByTagName("head");
```

В данном случае он возвращает массив элементов <head>.

Получив массив, можно извлечь первый его элемент, используя индекс 0:

```
var head = arrayOfHeadElements[0];
```

Возвращает первый элемент <head> в массиве (и там он должен быть единственным, не так ли?).

А сейчас давайте объединим обе эти строки следующим образом:

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

Извлекаем массив, а затем используем индекс в нем для извлечения первого элемента, причем делаем все это за один подход.

В приведенном примере кода мы постоянно используем первый элемент `<head>`, однако вы можете применять метод `getElementsByTagName` в случае с любыми тегами: `<p>`, `<div>` и т. д. При этом вы обычно будете получать обратно более одного из соответствующих элементов, которые имеются в массиве.



replaceChild под увеличительным стеклом

Взглянем также на метод `replaceChild`, поскольку вы не встречали его раньше. Данный метод следует вызывать по отношению к элементу, в котором вы хотите заменить дочерний элемент, передав ему ссылки на новый и старый дочерние элементы. Метод `replaceChild` просто заменяет старый дочерний элемент новым.

Метод `replaceChild` дает указание элементу `<head>` заменить один из его дочерних элементов с именем `oldScriptElement` новым элементом с именем `newScriptElement`.

Наш новый элемент `<script>`.

Элемент `<script>`, который уже имеется на странице.

```
head.replaceChild(newScriptElement, oldScriptElement);
```

Часть Задаваемые Вопросы

В: Почему нельзя просто заменить значение атрибута `src` вместо того, чтобы заменять элемент `<script>` целиком?

О: Если вы лишь замените значение атрибута `src` элемента `<script>` новым URL-адресом, браузер не будет рассматривать его как новый элемент `<script>` и, следовательно, не станет совершать запрос для извлечения JSONP. Чтобы заставить браузер выполнить запрос, необходимо создать абсолютно новый элемент `<script>`. Данная методика называется *скриптовой инъекцией*.

В: Что происходит со старым дочерним элементом, когда я заменяю его новым?

О: Он удаляется из объектной модели документа (DOM). А что произойдет дальше, будет уже зависеть от вас: если у вас по-прежнему имеется ссылка на него, сохраненная в находящейся где-то переменной, то вы сможете продолжить использовать его (любым путем, который будет иметь смысл). Если же у вас ее нет, то среда выполнения JavaScript в конце концов регенерирует (освобождает) память, которую занимает данный элемент в вашем браузере.

В: А что, если в HTML-файле будет более одного элемента `<head>`? В случае с вашим кодом, по-видимому, предполагается, что там будет только один элемент `<head>`, поскольку вы используете индекс 0 в массиве, возвращаемом методом `getElementsByTagName`

О: По определению HTML-файл включает только один элемент `<head>`. Следует отметить, что, конечно же, кто-то может включить и два таких элемента в HTML-файл. В этом случае получаемые результаты могут варьироваться (так и будет, если вы не произведете валидацию своего HTML!), но браузер, как обычно, постарается приложить максимум усилий, чтобы сделать все правильно (а что именно считается правильным, зависит от браузера).

В: Можно ли остановить интервальный таймер после его запуска?

О: Конечно, можно. Метод `setInterval` возвращает значение `id`, которое идентифицирует таймер. Сохранив данное значение `id` в переменной, вы сможете в любой момент передать его методу `clearInterval` для остановки таймера. Закрытие браузера также останавливает таймер.

В: Как можно узнать параметры веб-службы? И то, поддерживает ли она JSON и JSONP?

О: В случае большинства веб-служб публикуется общий API-интерфейс, который включает способы доступа к конкретной службе, а также описание всего того, что можно сделать с ее помощью. Если вы используете коммерческий API-интерфейс, то вам может потребоваться получить соответствующую документацию напрямую от поставщика. Информацию относительно большей части общих API-интерфейсов вам, скорее всего, удастся найти в Интернете, используя поисковик либо зайдя в раздел для разработчиков на сайте соответствующей организации. Вы также можете посетить, например, сайт programtheweb.com, где документируются API-интерфейсы, список которых постоянно увеличивается.

В: Очевидно, что XMLHttpRequest старше HTML5, а как насчет JSON и JSONP? Являются ли они частью HTML5? Нужен ли мне HTML5, чтобы использовать их?

О: Мы бы назвали JSON и JSONP современниками HTML5. Ни один из них пока не определен спецификацией HTML5, однако они активно используются HTML5-приложениями и играют важнейшую роль в процессе создания веб-приложений. Таким образом, несмотря на то что мы говорим HTML = Язык разметки + API-интерфейсы JavaScript + CSS, JSON и JSONP также являются весомой частью данной картины (равно как и запросы с использованием HTTP и XMLHttpRequest).

В: Продолжают ли люди использовать XML? Или в настоящее время везде уже господствует JSON?

О: В компьютерной индустрии существует одна прописная истина, согласно которой ничто никогда не умирает. При этом также следует отметить, что JSON в настоящее время набирает обороты, в силу чего создание многих новых веб-служб осуществлялся именно с его помощью. Вы часто будете сталкиваться с тем, что многие веб-службы поддерживают разнообразные форматы, включая XML, JSON и массу других (например, RSS). Преимуществом JSON является то, что он основан непосредственно на JavaScript, а JSONP позволяет избежать междоменных проблем.

Чуть не забыли сказать: берегитесь опасного кэша браузера



Мы почти готовы перейти к тестированию, однако сначала нужно позаботиться об одной небольшой детали, которая относится к проблемам типа «если вы никогда не делали этого прежде, то как вы вообще узнаете, что вам необходимо с этим справиться».

У большинства браузеров имеется любопытная особенность: если вы извлекаете данные, многократно используя один и тот же URL-адрес (как наш JSONP-запрос), то браузер в конце концов произведет их кэширование в целях новышения производительности, и вы будете снова и снова получать обратно один и тот же кэшированный файл (или данные). А это не то, что нам нужно.

К счастью, существует простое и старое, как Интернет, «лекарство» от этого. Все, что нам потребуется сделать, — это обеспечить добавление случайного числа в конец URL-адреса. Благодаря этой хитрости мы заставим браузер думать, что перед ним новый URL, который он никогда не видел прежде. Откорректируем наш код, изменив приводившуюся ранее строку с URL-адресом:

Данный код вы найдете в верхней части
своей функции `handleRefresh`.

Измените приводившееся
раньше объявление `url`, чтобы
оно стало выглядеть так.

```
var url = "http://gumball.wickedlysmart.com/?callback=updateSales" +  
        "&random=" + (new Date()).getTime();
```

Добавляем новый, «фиктивный»
параметр в конец URL-адреса.
Веб-сервер будет игнорировать
его, однако данного параметра
окажется достаточно для того,
чтобы обхитрить браузер.

Создаем новый объект `Date` и используем
метод `getTime` объекта `Date` для извлечения
значения времени в миллисекундах, после чего
добавляем это значение в конец URL-адреса.

Благодаря этому новому коду сгенерированный URL-адрес будет выглядеть примерно так:

Эта часть уже должна быть вам знакома...

А вот параметр `random`.

```
http://gumball.wickedlysmart.com?callback=updateSales&random=1309217501707
```

Эта часть будет каждый раз изменяться
для предотвращения кэширования.

Замените объявление переменной `url` в своей функции `handleRefresh` представленным здесь кодом, после чего вы будете готовы к проведению тест-драйва!

Еще раз проводим тест-драйв

Итак, на этот раз мы, несомненно, все предусмотрели. Все должно быть полностью готово. Убедитесь, что вы добавили весь код, приводившийся с момента последнего тест-драйва, и не перезагрузите страницу. Прекрасно, теперь мы видим, как данные неровно обновляются!

Постойте-ка... вы видите тоже, что и мы? Что это еще за дубликаты? Это нехорошо. Хм. Может быть, извлечение происходит слишком быстро и мы получаем отчеты, которые уже извлекались ранее?

Дубликаты!



Как избавиться от дубликатов отчетов об уровне продаж

Если вы снова взглянете на спецификации сервера Mighty Gumball, приводившиеся на с. 262, то увидите, что в URL-адресе запроса мы можем указать параметр `lastreporttime`, как показано далее:

Вы также можете добавить параметр `lastreporttime` в конец URL-адреса, чтобы извлечь только отчеты, которые поступили начиная с указанного времени. Например:

`http://gumball.wickedlysmart.com/?lastreporttime=1302212903099`

Просто укажите время в миллисекундах.

Это, конечно, здорово, но как узнать время поступления последнего отчета, который мы извлекали? Давайте еще раз взглянем на формат вывода отчетов об уровне продаж:

```
[{"name": "LOS ANGELES", "time": 1309208126092, "sales": 2},
 {"name": "PASADENA", "time": 1309208128219, "sales": 8},
 {"name": "DAVIS CREEK", "time": 1309214414505, "sales": 8}
...]
```

У каждого отчета об уровне продаж имеется время его поступления.



Я понял, к чему вы клоните; то есть мы можем отслеживать время поступления последнего отчета, а затем использовать его, когда будем совершать следующий запрос, чтобы сервер не возвращал нам отчеты, которые мы уже получили?

Вы все правильно поняли.

Чтобы отслеживать время поступления последнего отчета, нам потребуется внести ряд дополнений в функцию `updateSales`, в которой происходит вся обработка данных об уровне продаж. Однако сначала нам нужно будет объявить переменную, в которой будет размещаться значение времени поступления самого последнего отчета:

```
var lastReportTime = 0;
```

Значение времени не может быть меньше нуля, поэтому для начала просто зададим здесь значение 0.

Поместите данный код в верхнюю часть своего JavaScript-файла, вне пределов любой функции.

Теперь извлечем значение времени поступления самого последнего отчета об уровне продаж в `updateSales`:

```
function updateSales(sales) {  
    var salesDiv = document.getElementById("sales");  
    for (var i = 0; i < sales.length; i++) {  
        var sale = sales[i];  
        var div = document.createElement("div");  
        div.setAttribute("class", "saleItem");  
        div.innerHTML = sale.name + " sold " + sale.sales +  
            " gumballs";  
        salesDiv.appendChild(div);  
    }  
    if (sales.length > 0) {  
        lastReportTime = sales[sales.length-1].time;  
    }  
}
```

Если вы взглянете на массив `sales`, то увидите, что самый свежий отчет о продажах является последним в этом массиве. Поэтому здесь мы присваиваем его нашей переменной `lastReportTime`.

Нам необходимо убедиться в НАЛИЧИИ массива; в случае отсутствия новых отчетов о продажах будет возвращен пустой массив и наш код сгенерирует исключение.

Обновление URL-адреса JSON с целью включения lastreporttime

Теперь, когда мы отслеживаем время поступления последнего отчета об уровне продаж, нам необходимо удостовериться в том, что мы отправляем его на сервер Mighty Gumball как часть JSON-запроса. Для этого отредактируем функцию `handleRefresh` и добавим параметр запроса `lastreporttime`:

```
function handleRefresh() {
    var url = "http://gumball.wickedlysmart.com" +
        "?callback=updateSales" +
        "&lastreporttime=" + lastReportTime +
        "&random=" + (new Date()).getTime();
    var newScriptElement = document.createElement("script");
    newScriptElement.setAttribute("src", url);
    newScriptElement.setAttribute("id", "jsonp");
    var oldScriptElement = document.getElementById("jsonp");
    var head = document.getElementsByTagName("head")[0];
    if (oldScriptElement == null) {
        head.appendChild(newScriptElement);
    }
    else {
        head.replaceChild(newScriptElement, oldScriptElement);
    }
}
```

Разбиваем URL-адрес на несколько строк, которые конкатенируем друг с другом...
 ...а вот параметр `lastreporttime` со своим новым значением.

Тест-драйв lastReportTime



Проведем тестирование параметра запроса `lastreporttime` и посмотрим, решает ли он нашу проблему с дубликатами отчетов об уровне продаж. Убедитесь, что вы набрали весь приведенный чуть выше новый код, перезагрузите страницу и щелкните на кнопке Refresh (Обновить).



Отлично! Теперь мы будем получать только новые отчеты о продажах, то есть все дубликаты исчезнут!



Вы превзошли самих себя! Приложение работает замечательно — теперь я могу быть в курсе уровня продаж как сидя за своим рабочим столом, так и находясь в пути. Я начинаю думать, что в подобных веб-приложениях действительно что-то есть. Только представьте, сколько всего мы сможем сделать, используя автоматы для продажи жевательной резинки в сочетании с JSON и всеми этими API-интерфейсами HTML5!!



КЛЮЧЕВЫЕ МОМЕНТЫ

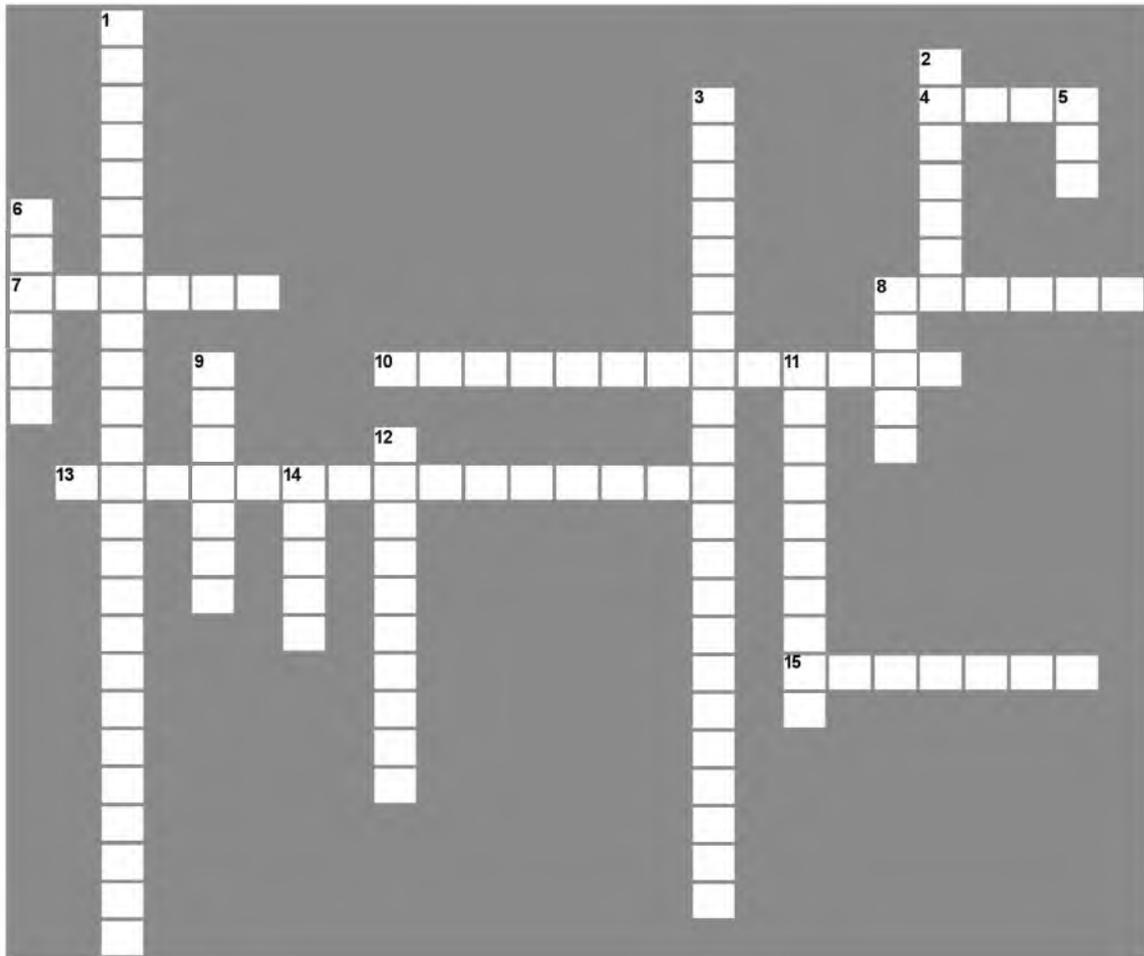


- XMLHttpRequest не позволяет запрашивать данные с сервера, отличного от того, с которого загружаются ваши HTML- и JavaScript-файлы. Такова браузерная политика безопасности, призванная предотвращать доступ вредоносного JavaScript-кода к вашим веб-страницам и файлам cookie пользователей.
- Альтернативой XMLHttpRequest для доступа к данным, имеющимся у веб-служб, является JSONP.
- Используйте XMLHttpRequest, если ваши HTML- и JavaScript-файлы размещаются на том же компьютере, что и ваши данные.
- Используйте JSONP, если вам требуется доступ к данным, имеющимся у веб-службы на удаленном сервере (при этом предполагается, что эта веб-служба будет поддерживать JSONP). Веб-служба — это веб-интерфейс API, доступ к которому осуществляется по протоколу HTTP.
- JSONP представляет собой способ извлечения данных с использованием элемента `<script>`.
- JSONP — это JSON-данные, обернутые в JavaScript; обычно JSON-данные оборачиваются в вызов функции.
- Функция, в вызов которой обернуты JSON-данные, в случае с JSONP называется *функцией обратного вызова*.
- Указывайте функцию обратного вызова в качестве параметра URL запроса в JSONP-запросе.
- JSONP не является более (или менее) безопасным подходом, чем связывание с JavaScript-библиотеками с помощью элемента `<script>`. Всегда проявляйте осторожность при связывании со сторонними JavaScript-библиотеками.
- Указывайте элемент `<script>` для совершения JSONP-запроса путем внедрения его непосредственно в свой HTML либо посредством добавления элемента `<script>` в объектную модель документа (DOM) с использованием JavaScript.
- Добавляйте случайное число в конец своего URL-адреса JSONP-запроса, если многократно используете этот URL для совершения запросов, чтобы браузер не кэшировал ответ.
- Метод `replaceChild` заменяет один элемент другим в DOM.
- `setInterval` — это таймер, который вызывает функции через заданный интервал времени. Вы можете использовать `setInterval` для отправки повторяющихся JSONP-запросов на сервер с целью извлечения новых данных.



HTML5-крестворд

Здорово, в этой главе вы научили свое веб-приложение общаться с веб-службами! Пора заставить поработать левое полушарие мозга для закрепления материала.



По горизонтали

- Шаблон использования XMLHttpRequest для извлечения данных с серверов иногда называют _____.
- _____ представляет собой новейшую модель автомата для продажи жевательной резинки Mighty Gumball с поддержкой подключения к Интернету.
- Мы допустили _____, сначала углубившись на 25 страниц в главу и только затем рассказав вам о браузерной политике безопасности.
- XMLHttpRequest высмеял слово _____ в названии JSONP.

По вертикали

- В середине данной главы нас подстерегал один из них.
- JSONP означает JSON with _____.
- JSONP использует _____.
- Формат, который, как мы все полагали, «спасет» мир.
- _____, работающий контролером качества, был расстроен, когда запрос, отправленный на производственный сервер Mighty Gumball, потерпел неудачу.
- Локальный сервер легко установить в операционной системе _____.
- У _____ имеется веб-служба JSONP.
- Используемые JSONP типы объектов.
- Компания Mighty Gumball проводит тестирование модели торгового автомата MG2200 в _____.
- _____ напомнила Фрэнку, Джиму и Джо о междоменных проблемах, возникающих в случае с XMLHttpRequest.

Специальное сообщение из главы 7...



Пока в вашей памяти еще свежи знания о JSONP, мы хотели бы, чтобы вы помогли нам кое с чем в главе, посвященной элементу canvas.

Мы работаем с API-интерфейсом JSONP Twitter и создаем службу, которая дает пользователям возможность размещать на футболках любые твиты.

Мы любим говорить: «Если вы — поклонник Twitter, то закажите футболку с отпечатанным на ней твитом».

←
Основательница
TweetShirt.com



7 раскрываем в себе художника

Элемент *canvas*

Да, разметка — это, конечно, здорово, но ничто не сравнится с тем, когда собственными руками раскрашиваешь что-либо свежими, аккуратными пикселями.



HTML больше не является просто языком «разметки». Благодаря новому HTML5-элементу `canvas` у вас появилась возможность собственноручно создавать *пиксели*, манипулировать ими и уничтожать их. В этой главе мы воспользуемся элементом `canvas`, чтобы раскрыть таящегося в вас художника — больше никаких разговоров о HTML, когда речь идет чисто о семантике и отсутствуют представления; используя `canvas`, мы начнем раскрашивать и рисовать красками. Теперь все будет опираться на представления. Вы узнаете, как вставлять элемент `canvas` в свои страницы, как рисовать текст и графические изображения (естественно, используя JavaScript) и даже как быть в случае с браузерами, не поддерживающими данный элемент.

↪
Ладно, со словом «уничтожать» мы, возможно, немного переборщили.

↪
Как стало известно, на самом деле элементы `<canvas>` и `<video>` делают нечто большее, чем просто делят друг с другом одни и те же веб-страницы... Об этих любопытных подробностях мы поговорим позже.

Наш новый стартап: TweetShirt



Наш девиз звучит так: «Если вы поклонник Твиттера, посите футболки TweetShirt».

В конце концов, чтобы пазываться журналистом, нужно стремиться к тому, чтобы написанное вами опубликовали, а лучшее место для печати на футболке — область груди! По крайней мере, в этом заключается суть коммерческого призыва паше-го стартапа, которого мы будем придерживаться.

Чтобы успешно положить начало данному стартану, нам потребуется веб-приложение, позволяющее клиентам создавать пользовательский дизайн футболок, отражающий один из их недавних твитов.



Вероятно, вы подумали: «А знаете, это неплохая идея». Что ж, в таком случае мы с вами успешно положим начало данному стартану, создав к концу главы полностью готовое к работе веб-приложение. А если вы решите использовать его для зарабатывания денег, мы не станем заявлять права на интеллектуальную собственность, но хотя бы вышлите нам бесплатную футболку!



Мы любим говорить:
«Если вы поклонник Твиттера, закажите футболку с отпечатанным твитом».

Нам требуется соответствующее веб-приложение, которое даст пользователям возможность создавать классные представления их любимых твитов.

Нам также необходимо обеспечить его работоспособность на различных устройствах. Наши клиенты пользуются Твиттером, находясь в дороге, и они смогут создавать дизайн футболок в пути, причем в режиме реального времени!

Основательница
TweetShirt.com

Взгляд на «оригинал-макет»

Проведя исчерпывающую итеративную разработку и обширное тестирование с участием фокус-группы, мы создали оригинал-макет (процесс его создания по-другому называется пачальным визуальным проектированием), на который сейчас и взглянем.

Дизайн футболки

Это наш старт. Сначала мы все нарисовали на салфетке, сидя в Starbuzz Coffee.

Наше веб-приложение должно по возможности выглядеть, как данная страница! Другими словами, мы хотим показывать дизайн футболки и предоставлять пользователю возможность интерактивно изменять его с помощью элементов управления.



А вот как должен выглядеть интерфейс пользователя.

Пользователь сможет выбирать цвет фона (Select background color), круги или квадраты для отображения на фоновой поверхности (Circles or Squares?), цвет текста (Select text color) и твит (Peek a tweet).

Select background color:

White

Circles or Squares?

Circles

Select text color:

Black

Peek a tweet:

@starbuzzceo you're on a #shirt #tweetshirt

Preview

Buy

МОЗГОВОЙ ШТУРМ

Взгляните еще раз на требования, приведенные на предыдущей странице. Как вы считаете, можно ли их выполнить, используя HTML5? А помните, что одно из требований заключается в обеспечении работоспособности вашего сайта на как можно на большем количестве устройств различного формата и размера?

Ознакомьтесь с приведенными ниже возможностями (а затем выберите наилучший ответ).

- Можно просто использовать технологию Flash, поскольку она работает в большинстве браузеров.
- Можно обратить свой взор на HTML5 и посмотреть, есть ли какие-нибудь новые технологии, которые могут помочь (подсказка: есть одна такая технология под названием `canvas`).
- Можно написать пользовательское приложение для каждого устройства, благодаря чему вы будете знать, какое именно качество взаимодействия обеспечите для своих пользователей.
- Можно задействовать вычислительную обработку изображений на стороне сервера, а затем доставлять их в браузер.

Часть Задаваемые Вопросы

В: А серьезно, почему бы в данной ситуации не использовать Flash или не прибегнуть к написанию пользовательского приложения?

О: Flash является замечательной технологией и вы, несомненно, можете использовать именно ее. Однако в настоящее время индустрия движется по направлению к HTML5, и на момент написания данной книги у вас могли бы возникнуть проблемы с выполнением Flash-приложений на всех устройствах, включая очень популярные.

Написание пользовательского приложения является отличным выходом, если вам требуется обеспечить для пользователей взаимодействие, полностью заточенное под конкретное устройство. Имейте в виду, что разработка пользовательского приложения для множества разных устройств — дело затратное.

В случае с HTML5 вы получаете отличную поддержку со стороны браузеров как для мобильных, так и для настольных устройств и зачастую можете создавать приложения, используя всего одно технологическое решение.

В: Мне нравится идея генерирования изображений на стороне сервера. Таким образом, я смогу написать один блок кода, который вместе с генерируемыми изображениями сможет работать на всех устройствах. Я немного знаю PHP, так что эта задача должна быть мне по плечу.

О: Это еще один путь, по которому вы можете пойти, однако его недостаток заключается в том, что если у вас будет огромное множество пользователей, вам придется беспокоиться о масштабировании используемых для генерирования изображений серверов, чтобы они смогли удовлетворить предъявляемые требования (в противопоставлении с тем, когда клиент каждого из пользователей сам будет генерировать предварительный просмотр футболки). Вы также сможете обеспечить более интерактивное и цельное взаимодействие, если предпочтете написать соответствующий код для браузера.

Как? Что ж, мы рады, что вы об этом спросили...

Заглянем к ребятам из команды TweetShirt...

Вы уже ознакомились с требованиями, а также с базовым проектированием взаимодействия пользователя с приложением. Пришло время перейти к сложной части — реализации всего этого на практике. Давайте прислушаемся к разговору и узнаем, как обстоят дела...

Джо: Я думал, что все будет просто, пока не увидел эти фоновые круги.

Фрэнк: Что ты хочешь этим сказать? Ведь это всего лишь изображение...

Джуди: Нет, нет, осповательница TweetShirt хочет, чтобы размещение кругов происходило случайным образом, благодаря чему расположение кругов, к примеру, на моей футболке будут отличаться от расположения кругов на твоей. То же самое и с квадратами.

Фрэнк: Все нормальпо, поскольку раньше мы делали это, генерируя изображение на стороне сервера.

Джо: Да, я знаю, по такой подход был не очень разумным; помните, как нам приходилось платить компании Amazon за пользование серверами?

Фрэнк: Да. Но это не беда.

Джо: В любом случае, нам необходимо, чтобы все работало очень быстро, то есть не было никаких долгих «рейсов» обратно на сервер. Поэтому давайте будем делать все на стороне клиента, если это возможно.

Джуди: Ребята, я думаю, что это возможно, я тут смотрю на canvas в HTML5.

Фрэнк: Я всего лишь дизайнер, поэтому разъясни мне, что это такое.

Джуди: Фрэнк, ты, должно быть, уже слышал о canvas — это новый элемент в HTML5, который позволяет создавать поддерживающую рисование область для 2D-фигур, текста и растровых изображений.

Фрэнк: Все это похоже на тег ``. Мы просто помещаем его в страницу с указанием ширины и высоты, а браузер делает все остальное.

Джуди: Неплохое сравнение, мы действительно определяем ширину и высоту для canvas, однако в данном случае все рисуемое в canvas будет зависеть от JavaScript-кода.

Джо: А где здесь в дело вступает разметка? Можно ли сказать canvas на JavaScript: «Данный элемент `<h1>` необходимо разместить вот здесь».

Джуди: Нет, после того как ты помещаешь canvas в страницу, ты оставляешь мир разметки позади; на JavaScript мы размещаем точки, линии, контуры, изображения и текст. Это низкоуровневый API-интерфейс.

Джо: Что ж, если он сможет справиться со всеми этими размещаемыми случайным образом кругами, то он меня устраивает. Ладно, хватит разговоров, давайте взглянем на него!



Фрэнк, Джуди и Джо

Как добавить canvas в свою веб-страницу

Фрэнк был прав в том, что canvas песколько схож с элементом ``. Добавление canvas осуществляется следующим образом:

Элемент canvas представляет собой обычный HTML-элемент, в начале которого располагается открывающий тег `<canvas>`.

Атрибут `width` определяет количество пикселей по горизонтали, которое canvas будет занимать на веб-странице.

Аналогичным образом `height` определяет вертикальную область веб-страницы, которую будет занимать элемент canvas (в данном случае — 200 пикселей).

```
<canvas id="lookwhatIdrew" width="600" height="200"></canvas>
```

Мы добавили `id` для идентификации canvas, а как его использовать, вы увидите чуть позже...

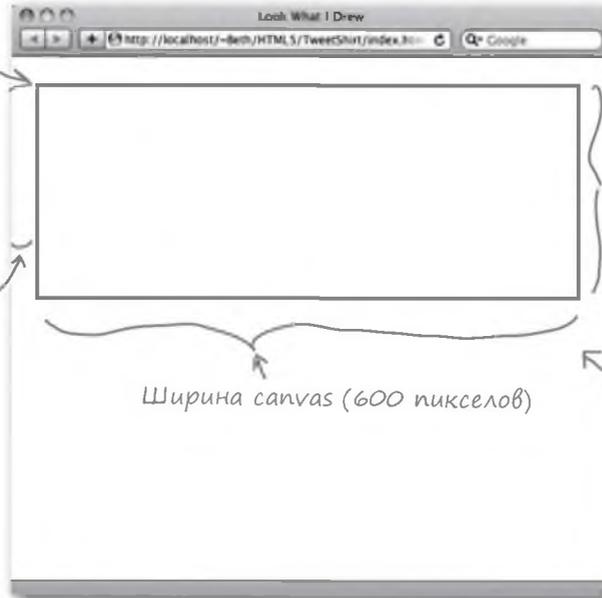
Здесь для `width` мы задали значение 600 пикселей.

Закрывающий тег

Браузер выделит пространство на веб-странице для canvas согласно указанным значениям `width` и `height`.

Левая верхняя позиция canvas. Будем использовать эту точку, чтобы отмерять все остальное в canvas (в чем вы вскоре убедитесь).

Вокруг canvas имеется небольшое пространство, представляющее собой поле по умолчанию элемента `<body>`.



В данном случае значением `width` является 600, а значением `height` — 200.

Ваш повседневный HTML может обтекать canvas. Элемент canvas является таким же, как и любой другой элемент (например, `image` и т. п.).

Тест-драйв вашего нового canvas

Пришло время задействовать даппый элемент па вашей веб-страпице. Наберите приве- депный ниже код и сохрaпите его в повом файле, а затем загрузите в своем браузерe.

```
<!doctype html>
<html lang="en">
<head>
  <title>Look What I Drew</title>
  <meta charset="utf-8">
</head>
<body>

<canvas id="lookwhatIdrew" width="600" height="200"></canvas>

</body>
</html>
```

← Напечатайте данный код и протестируйте его.

Вот что видит она...



...вероятно, то же самое увидите и вы!

Мы нарисовали эти линии, чтобы объяснить, как именно canvas вставляется в страницу, и сделали это лишь в целях иллюстрирования. На самом деле их там не будет (если только вы их сами не нарисуете).



↘ Проверните страницу, чтобы узнать больше...



Как увидеть свой canvas

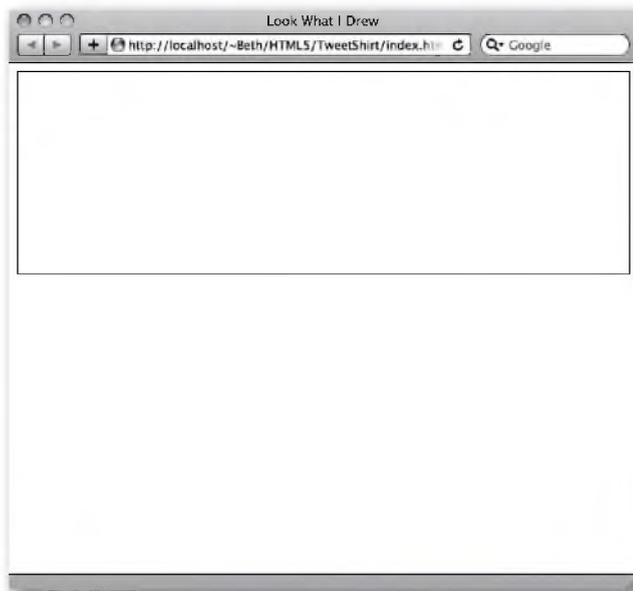
Пока вы не нарисуете что-нибудь в элементе canvas, вы его не увидите. Он просто является пространством в окне браузера, в котором вы можете рисовать. Рисованию в canvas мы займемся очень скоро, а на данный момент нам требуются доказательства того, что canvas действительно присутствует в нашей странице.

Существует другой способ увидеть canvas... Если мы применим CSS для стилизации элемента <canvas>, чтобы стала видимой его рамка, то сможем увидеть его на странице. Задействуем простой стиль, который добавляет черную рамку шириной 1 пиксел для canvas.

```
<!doctype html>
<html lang="en">
<head>
  <title>Look What I Drew</title>
  <meta charset="utf-8">
  <style>
    canvas {
      border: 1px solid black;
    }
  </style>
</head>
<body>
<canvas id="lookwhatIdrew" width="600" height="200"></canvas>
</body>
</html>
```

← Мы добавили стиль для canvas, который просто рисует черную границу толщиной 1 пиксел, которую можно видеть на странице.

→ Так намного лучше! Теперь мы видим canvas. Дальше сделаем с ним кое-что интересное...



Часть
Задаваемые
Вопросы

В: На каждую страницу может приходиться только один `canvas`?

О: Нет, вы можете вставлять в страницу столько элементов `canvas`, сколько вам потребуется (либо то количество, с которым сможет справиться браузер или ваши пользователи). Просто присвойте всем им уникальные `id`, и вы сможете рисовать в каждом из них как в отдельном `canvas`. Вскоре вы увидите, как использовать `id` элемента `canvas`.

В: Является ли `canvas` прозрачным?

О: По умолчанию `canvas` является прозрачным. Вы можете рисовать в элементе `canvas`, заполняя его цветными пикселями. Как именно это делается, вы узнаете позже в данной главе.

В: Если `canvas` прозрачен, то это означает, что я могу разместить его поверх другого элемента, чтобы, например, нарисовать что-нибудь на изображении или еще на чем-либо, имеющемся на странице, ведь так?

О: Все верно! Это одна из замечательных особенностей `canvas`. Он позволяет добавлять графику в любую часть веб-страницы.

В: Можно ли использовать CSS для задания ширины и высоты `canvas` вместо атрибутов `width` и `height` в случае с данным элементом?

О: Да, можно, однако здесь все будет работать немного по-другому, чем вы могли бы ожидать. По умолчанию элемент `canvas` имеет ширину 300 пикселей и высоту 150 пикселей. Если вы не зададите атрибуты `width` и `height` в теге `<canvas>`, то элемент получит размеры по умолчанию. Если вы затем зададите размеры, используя CSS, допустим, 600 × 200 пикселей, то `canvas` размером 300 × 150 пикселей подвергнется *масштабированию* в соответствии с этими новыми параметрами, равно как и все, что будет в нем нарисовано. Это сродни масштабированию изображения путем задания для него новых ширины и высоты, которые больше или меньше, чем реальные ширина и высота этого изображения. А при увеличении на изображении будет наблюдаться пикселизация, не так ли?

То же самое происходит и в случае с `canvas`. Элемент `canvas` шириной 300 пикселей, масштабируемый до ширины 600 пикселей, будет включать прежнее количество пикселей, но растянутых вдвое, из-за чего все станет выглядеть несколько неуклюже. Однако если вы задействуете атрибуты `width` и `height` в `<canvas>`, то сможете задать для `canvas` размеры, которые будут больше (или меньше), чем 300 × 150 пикселей, и все создаваемое в этом `canvas` будет отрисовываться нормально. Таким образом, мы рекомендуем задавать атрибуты тега `width` и `height` и не прибегать к CSS для настройки данных свойств, если только вы действительно не хотите масштабировать `canvas`.



Возможно, вы обратили внимание на то, что наш элемент `canvas` не включал никакого содержимого. Если поместить текст между тегам, то что, по вашему мнению, будет делать браузер, когда страница загрузится?

`<canvas>`



`</canvas>`

Рисование в элементе canvas

На данный момент у нас имеется пустой элемент canvas. Не будем медлить, постараемся обрести вдохновение как JavaScript-писатели и поместим в наш canvas прямоугольник с черной заливкой. Предварительно нам потребуется решить, где именно он будет располагаться и насколько большим мы его сделаем. Как насчет того, чтобы его местоположение было $x=10$ и $y=10$ пикселей, а высота и ширина равнялась 100 пикселям? Так и поступим.

А теперь взглянем на код, который нам будет для этого необходим:



```

<!doctype html>
<html lang="en">
<head>
  <title>Look What I Drew</title>
  <meta charset="utf-8" />
  <style>
    canvas { border: 1px solid black; }
  </style>
  <script>
    window.onload = function() {
      var canvas = document.getElementById("tshirtCanvas");

      var context = canvas.getContext("2d");

      context.fillRect(10, 10, 100, 100);

    };
  </script>
</head>
<body>
  <canvas width="600" height="200" id="tshirtCanvas"></canvas>
</body>
</html>

```

Сначала идет стандартный HTML5.

Наша CSS-рамка остается на месте.

Вот наш обработчик onload; рисовать мы начнем после полной загрузки страницы

Для рисования в canvas нам потребуется ссылка на него. Воспользуемся getElementById, чтобы извлечь ее из DOM.

Хм, это интересно, нам явно необходим контекст "2d" из canvas, чтобы рисовать...

Используем контекст 2d для рисования прямоугольника с заливкой в canvas.

Эти числа являются позицией x, y прямоугольника в canvas.

Здесь у нас также имеются ширина и высота (в пикселях).

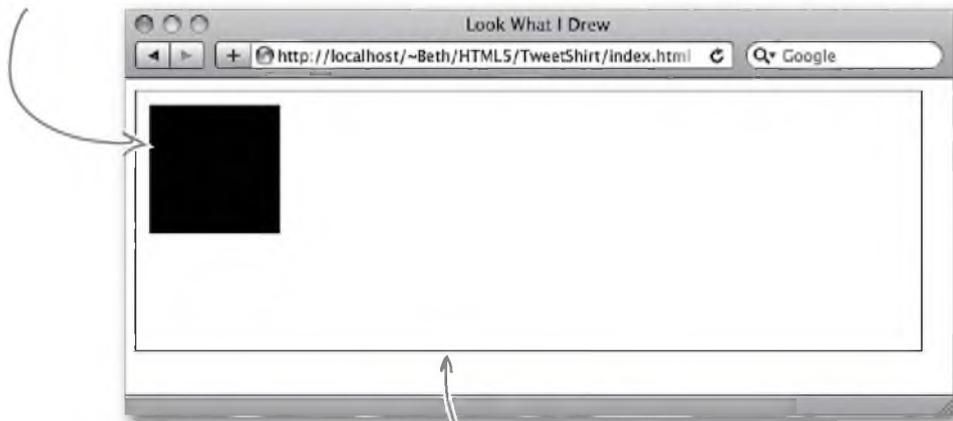
Любопытно то, что метод fillRect не принимает цвет заливки... Подробнее об этом мы поговорим чуть позже.

И не будем забывать о нашем элементе canvas. Здесь мы определяем canvas шириной 600 и высотой 200 пикселей, с id в виде "tshirtCanvas".

Небольшой тест-драйв canvas...

Наберите приведенный чуть выше код (либо можете взять его с <http://wickedlysmart.com/hfhtml5>) и загрузите его в своем браузере. Если вы используете современный браузер, то должны увидеть примерно то же, что и мы:

Вот наш прямоугольник с размерами 100 × 100 пикселей, имеющий позицию 10, 10 в canvas.



А вот наш canvas шириной 600 и высотой 200 пикселей с черной рамкой толщиной 1 пиксел.

Более пристальный взгляд на код

Это был отличный небольшой тестовый прогон, однако давайте нырнем поглубже.

- 1 В нашей разметке мы определили canvas и присвоили ему идентификатор, используя тег `<canvas>`. Первое, что нам необходимо сделать, чтобы начать рисовать в данном canvas, — это извлечь ссылку на объект canvas в объектной модели документа (DOM). Как обычно, мы делаем это с помощью метода `getElementById`:

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

- 2 Мы располагаем ссылкой на элемент canvas, присвоенной переменной canvas. Далее нам нужно разобраться с частью «протокола», которому необходимо следовать, прежде чем мы сможем перейти к рисованию в canvas. Нам нужно сказать canvas, чтобы он предоставил нам контекст для рисования. А в данном случае нам требуется именно контекст 2d. Контекст, возвращаемый canvas, присваивается переменной context:

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

← Это часть «протокола», которому нам необходимо следовать, прежде чем мы сможем приступить к рисованию в canvas.

- 2 Теперь, имея в руках объект context, мы можем использовать его для рисования в canvas, для чего вызываем метод fillRect. Данный метод создает прямоугольник, беря за начало позицию x, y со значениями соответственно 10, 10, при этом он будет иметь ширину и высоту 100 пикселей.

Обратите внимание, что мы вызываем метод fillRect в отношении context, а не canvas.

```
context.fillRect(10, 10, 100, 100);
```

← Проведите тест, и вы увидите, как появится черный прямоугольник. Попробуйте изменить значения x, y, а также ширину и высоту и посмотрите, что будет.



Можете ли вы придумать способ использовать элемент canvas, если ваш браузер поддерживает его, а при отсутствии поддержки просто выводить сообщение, например: Неу, you, yes you, upgrade your browser!! (Эй, вы, да, вы, обновите свой браузер!!)?

Часть
Задаваемые
Вопросы

В: Откуда canvas знает, что к прямоугольнику следует применять именно черную заливку?

О: В случае canvas черный является цветом по умолчанию. Конечно же, можно изменить это, воспользовавшись свойством fillStyle, в чем вы вскоре убедитесь.

В: А если мне потребуется нарисовать контур прямоугольника, а не фигуру с заливкой?

О: Чтобы нарисовать только контур прямоугольника, вместо fillRect следует воспользоваться функцией strokeRect. Подробнее об обводке мы поговорим позже в этой главе.

В: Что такое контекст 2d и почему нельзя рисовать сразу в canvas?

О: canvas представляет собой графическую область, отображаемую на веб-странице. context — это объект, ассоциированный с canvas, который определяет набор свойств и методов, используемых для рисования в canvas. Вы можете даже сохранять состояние context, а затем восстанавливать его позже, что иногда бывает кстати. В оставшейся части этой главы вы познакомитесь с множеством свойств и методов объекта context.

Элемент canvas создавался с расчетом на поддержку более одного интерфейса — 2d, 3d и пр., о которых мы даже еще не задумывались. Используя context, можно работать с разными интерфейсами в пределах одного элемента canvas. Нельзя рисовать сразу в canvas, поскольку необходимо будет указать, какой интерфейс вы используете, выбрав context.

В: Означает ли это, что существует также контекст 3d?

О: Пока еще нет. Существует ряд конкурирующих перспективных стандартов, но не похоже, что какому-либо из них уже удалось стать лидером. Не упускайте из виду данный момент; между тем взгляните на библиотеку WebGL, а также на библиотеки, которые ее используют, например SpiderGL, SceneJS и three.js.



Для Любопытных

Интересно, а как предусмотреть в коде проверку на предмет того, поддерживает браузер canvas или нет?

Это, конечно же, можно сделать, однако отметим, что все это время предполагалось, что наш браузер поддерживает canvas. Но в любом производственном коде вам будет необходимо позаботиться о проведении проверки, чтобы убедиться в наличии такой поддержки.

Все, что вам потребуется сделать, — это проверить, присутствует ли метод getContext в соответствующем объекте canvas (который возвращается методом getElementById):

Сначала мы извлекаем ссылку на элемент canvas в странице.

```
var canvas =
    document.getElementById("tshirtCanvas");
if (canvas.getContext) {
    // поддержка canvas имеется
} else {
    // извините, API-интерфейс canvas
    не поддерживается
}
```

Затем проверяем присутствие метода getContext. Обратите внимание: мы не вызываем его, а просто смотрим, есть ли у него значение.

Если вы захотите проводить проверку на предмет поддержки canvas без необходимости заранее иметь canvas в своей разметке, то можете создавать элемент canvas «на лету», используя все методички, которые вам уже известны. Например:

```
var canvas =
    document.createElement("canvas");
```

Не забудьте заглянуть в приложение (в конце книги), где имеется информация о библиотеке с открытым исходным кодом, которую вы сможете использовать для проведения последовательного тестирования на предмет поддержки всевозможной функциональности в HTML5.



Когда я провожу тест в Internet Explorer, то ничего не вижу там, где должен быть canvas. В чем тут дело?

Internet Explorer поддерживает canvas только с версии 9 и выше, поэтому вы должны предусмотреть в коде своей страницы вывод соответствующего сообщения для пользователей.

Дело обстоит следующим образом: если вам действительно требуется обеспечить поддержку функциональности canvas в браузере Internet Explorer (версии ниже 9), то вы можете прибегнуть к проекту ExplorerCanvas или иному похожему инструменту и использовать его как плагин, обеспечивающий данную функциональность.

Теперь предположим, что вы решили уведомлять своих пользователей о том, что они упускают замечательное содержимое вашего canvas. Посмотрим, как это сделать...

И, возможно, вы также посоветуете им обновить браузер Internet Explorer до версии 9!

Выходим достойно из проблемной ситуации

Итак, возможно возникновение ситуации, когда пользователь решит зайти на ваш сайт, но его браузер не будет поддерживать элемент `canvas`. Не хотели бы вы в этом случае вывести для него сообщение о том, что ему следует обновить браузер? Вот код, который для этого потребуется:

Мы используем типичный, заурядный элемент `canvas`.

```
<canvas id="awesomecontent">
  Hey you, yes YOU, upgrade your browser!!
</canvas>
```

Сообщение, которое будет выводиться для пользователей, чьи браузеры не поддерживают `canvas`.

Как это работает? Всякий раз, когда браузер видит незнакомый элемент, он по умолчанию выводит любой содержащийся в нем текст. Таким образом, когда браузеры, не поддерживающие элемент `<canvas>`, столкнутся с ним, они выведут на экран сообщение `Hey, you, yes YOU, upgrade your browser!!` (Эй, вы, да, ВЫ, обновите свой браузер!!). А поддерживающие данный элемент браузеры будут просто игнорировать любой текст между тегами `<canvas>` и не станут выводить его на экран.



За то, что все это делается так легко, мы должны поблагодарить парней (и девушек), занимающихся разработкой стандартов HTML5!

Как вы уже знаете, другой способ выхода из ситуации, когда браузеры не поддерживают `canvas`, заключается в использовании JavaScript с целью выяснения того, знаком ли браузеру данный элемент. Такой подход обеспечивает немного большую гибкость в обеспечении для пользователей иного взаимодействия в случае, если их браузеры не будут поддерживать `canvas`; например, вы сможете перенаправить их на другую страницу или взамен вывести на экран какое-то изображение.

Теперь, когда мы знаем, как создавать прямоугольники, можно использовать аналогичный подход для создания квадратов в `canvas`, не так ли? Нам необходимо понять, как размещать их на футболке случайным образом, с заливкой цветом, выбранным пользователем.



Фрэнк: Да, конечно, но нам также потребуется интерфейс пользователя, с помощью которого люди смогут выбирать необходимые им варианты. Я имею в виду, что у нас есть макет, по его нужно реализовать.

Джудн: Ты прав, Фрэнк. Двигаться дальше без интерфейса не имеет смысла.

Джо: Он, вероятно, будет представлять собой простой HTML?

Фрэнк: Да, похоже. Однако если исходить из того, что мы пытаемся все сделать на стороне клиента, то как оно будет работать? Например, куда будет отправляться форма? Я не уверен, что понимаю, как все это станет слаженно функционировать.

Джо: Фрэнк, мы можем просто вызывать JavaScript-функцию, когда пользователь нажмет кнопку Preview (Предварительный просмотр), а затем отображать дизайн футболки в элементе `canvas`.

Фрэнк: Звучит разумно, но как мы получим доступ к значениям формы, если все они будут располагаться на стороне клиента?

Джудн: Тем же путем, каким мы всегда получаем доступ к объектной модели документа (DOM); мы можем воспользоваться `document.getElementById` для извлечения значений формы.

Вам уже доводилось делать это ранее.

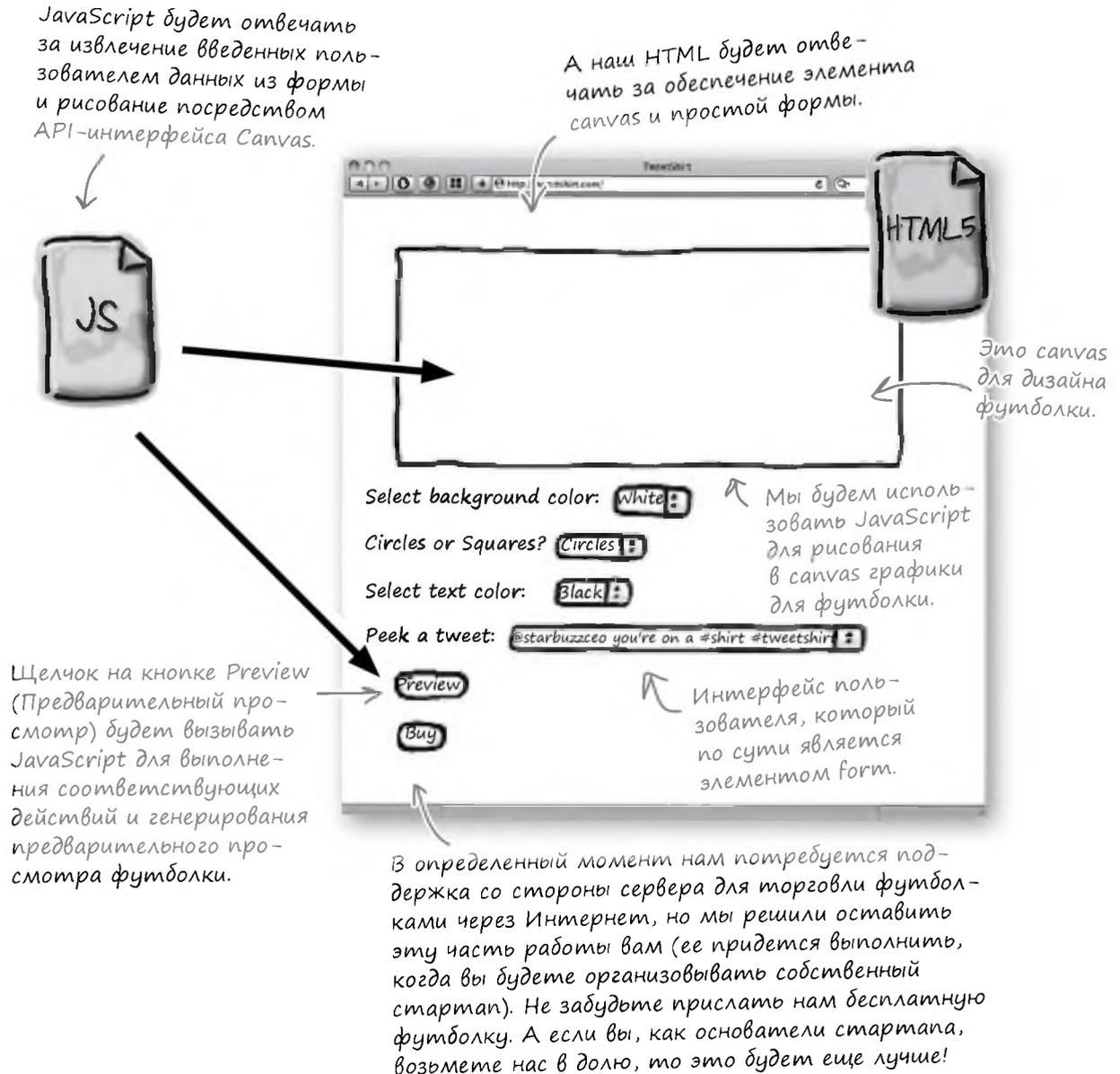
Фрэнк: Что ж, ребята, отступить нам некуда.

Джо: Ладно, давайте вместе пошагово во всем разберемся. Начнем с рассмотрения общей картины.

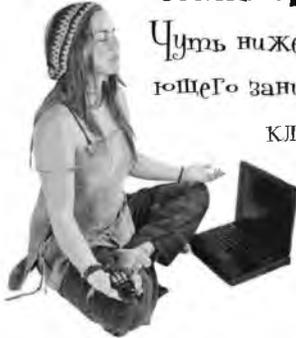
TweetShirt: общая картина

Прежде чем мы приступим к серьезной работе по реализации задуманного, давайте взглянем на общую картину. Мы собираемся создать веб-приложение, используя элемент `canvas` наряду с элементами формы, которые будут играть роль интерфейса пользователя, а всю работу за кадром будут выполнять JavaScript и API-интерфейс `Canvas`.

Вот как все будет выглядеть:



СТАНЬ браузером



Чуть ниже вы найдете форму для интерфейса, позволяющего заниматься дизайном футболок. Ваша задача заключается в том, чтобы сыграть роль браузера и осуществить рендеринг данного интерфейса. Сделав это, сравните ваш интерфейс с приведенным на предыдущей странице, чтобы узнать, все ли вы правильно сделали.

```
<form>
<p>
  <label for="backgroundColor">Select background color:</label>
  <select id="backgroundColor">
    <option value="white" selected="selected">White</option>
    <option value="black">Black</option>
  </select>
</p>
<p>
  <label for="shape">Circles or squares?</label>
  <select id="shape">
    <option value="none" selected="selected">Neither</option>
    <option value="circles">Circles</option>
    <option value="squares">Squares</option>
  </select>
</p>
<p>
  <label for="foregroundColor">Select text color:</label>
  <select id="foregroundColor">
    <option value="black" selected="selected">Black</option>
    <option value="white">White</option>
  </select>
</p>
<p>
  <label for="tweets">Pick a tweet:</label>
  <select id="tweets">
  </select>
</p>
<p>
  <input type="button" id="previewButton" value="Preview">
</p>
</form>
```

Результат рендеринга вашего интерфейса изобразите здесь. Нарисуйте, как будет выглядеть веб-страница с элементами формы слева.



Представьте, что вы используете этот интерфейс для выбора вариантов для своей футболки.



СТАНЬ браузером еще раз

Теперь, когда у вас есть интерфейс, выполните приведенные здесь JavaScript-операторы и напишите значение для каждого элемента интерфейса. (Ваши ответы вы сможете проверить в решении к данному упражнению в конце главы.



```
var selectObj = document.getElementById("backgroundColor");
var index = selectObj.selectedIndex;
var bgColor = selectObj[index].value;
```

```
var selectObj = document.getElementById("shape");
var index = selectObj.selectedIndex;
var shape = selectObj[index].value;
```

```
var selectObj = document.getElementById("foregroundColor");
var index = selectObj.selectedIndex;
var fgColor = selectObj[index].value;
```

Сначала напишем необходимый HTML

Хватит разговоров! Давайте займемся созданием нашего приложения. Прежде чем мы сможем сделать что-то еще, нам потребуется простая HTML-страница. Обновите свой файл `index.html`, чтобы его содержимое выглядело следующим образом:

```
<!doctype html>
<html lang="en">
<head>
  <title>TweetShirt</title>
  <meta charset="utf-8" />
  <style>
    canvas {border: 1px solid black;}
  </style>
  <script src="tweetshirt.js"></script>
</head>
<body>
  <h1>TweetShirt</h1>
  <canvas width="600" height="200" id="tshirtCanvas">
    <p>Please upgrade your browser to use TweetShirt!</p>
  </canvas>
  <form>
  </form>
</body>
</html>
```

Да, это славный HTML5-совместимый файл!

Обратите внимание, что мы изменили заголовков на TweetShirt.

Поместим весь JavaScript-код в отдельный файл, чтобы им было немного легче управлять.

А вот наш canvas!

Мы также предусмотрели небольшое сообщение для пользователей, которые применяют устаревшие версии браузеров.

Это form, в котором будут содержаться все элементы управления для приложения TweetShirt. Об этом мы поговорим на следующей странице...



Что еще вам необходимо знать для того, чтобы заменить CSS-рамку в своем canvas рамкой, нарисованной в canvas с помощью JavaScript? Какую методику вы предпочли бы (CSS или JavaScript) и почему?

Теперь добавим <form>

Итак, добавим интерфейс пользователя, чтобы мы могли приступить к написанию кода для создания дизайна футболок. Вы уже сталкивались с этим кодом раньше, однако на этот раз мы добавили ряд аппонятий, чтобы все было понятно (пабирая код, обязательно обратите на них внимание):

```

<form>
<p>
  <label for="backgroundColor">Select background color:</label>
  <select id="backgroundColor">
    <option value="white" selected="selected">White</option>
    <option value="black">Black</option>
  </select>
</p>
<p>
  <label for="shape">Circles or squares?</label>
  <select id="shape">
    <option value="none" selected="selected">Neither</option>
    <option value="circles">Circles</option>
    <option value="squares">Squares</option>
  </select>
</p>
<p>
  <label for="foregroundColor">Select text color:</label>
  <select id="foregroundColor">
    <option value="black" selected="selected">Black</option>
    <option value="white">White</option>
  </select>
</p>
<p>
  <label for="tweets">Pick a tweet:</label>
  <select id="tweets">
  </select>
</p>
<p>
  <input type="button" id="previewButton" value="Preview">
</p>
</form>

```

Весь этот код будет размещаться между тегами <form>, указанными на предыдущей странице.

Здесь пользователь будет выбирать цвет фона для дизайна футболки. На выбор дается черный (black) либо белый (white) цвет. Вы можете добавить сюда другие цвета по своему усмотрению.

Здесь мы задействуем еще один элемент управления, позволяющий выбирать круги (circles) или квадраты (squares) для конфигурирования дизайна. Пользователь также сможет выбрать вариант none (нет) — фон не будет содержать никаких фигур.

Еще один элемент управления select, на этот раз — для выбора цвета текста. Опять-таки, на выбор дается черный (black) либо белый (white) цвет.

Вот где будут размещаться все твиты. А почему здесь пусто? Ах да, мы же будем заполнять данный элемент позже (подсказка: нам необходимо, чтобы твиты извлекались непосредственно из Твиттера, ведь это же веб-приложение, не так ли?!).

Если вам доводилось сталкиваться с элементами form, то вы, вероятно, обратили внимание, что здесь form не имеет атрибута action (а это означает, что при щелчке на кнопке Preview (Предварительный просмотр) ничего не произойдет). Со всем этим мы разберемся через несколько мгновений...

В конце добавляем кнопку для предварительного просмотра футболки.

Пришло время добавить JavaScript для вычислений

Разметка — это, конечно, хорошо, однако именно JavaScript будет объединять все веб-приложение TweetShirt. Мы разместим необходимый код в файле `tweetshirt.js`. Начать мы собираемся с размещения произвольных квадратов на футболке, но прежде необходимо активировать кнопку Preview (Предварительный просмотр), чтобы при ее нажатии происходил вызов JavaScript-функции.

Создайте файл `tweetshirt.js`
и добавьте в него данный код.

```

window.onload = function() {
    var button = document.getElementById("previewButton");
    button.onclick = previewHandler;
};

```

Сначала будет извлекаться элемент `previewButton`.

Добавьте обработчик событий `click` для данной кнопки, чтобы при щелчке на ней (или при прикосновении к ней на сенсорном экране мобильного устройства) происходил вызов функции `previewHandler`.

Таким образом, когда кнопка Preview (Предварительный просмотр) будет нажата, произойдет вызов функции `previewHandler`. И здесь самое время обловить `canvas` с целью представить футболку, пад дизайном которой трудится пользователь. Приступим к написанию `previewHandler`:

```

function previewHandler() {
    var canvas = document.getElementById("tshirtCanvas");
    var context = canvas.getContext("2d");

    var selectObj = document.getElementById("shape");
    var index = selectObj.selectedIndex;
    var shape = selectObj[index].value;

    if (shape == "squares") {
        for (var squares = 0; squares < 20; squares++) {
            drawSquare(canvas, context);
        }
    }
}

```

Сначала извлекаем элемент `canvas` и запрашиваем его контекст рисования `2d`.

Теперь нам необходимо узнать, какую фигуру выбрал в интерфейсе пользователь. Первым делом извлекаем элемент с `id` в виде `"shape"`.

Затем выясняем, какой элемент был выбран (квадраты или круги), для чего извлекаем индекс выбранного элемента и присваиваем его значение переменной `shape`.

И если значением `shape` будет «`squares`», нам потребуется нарисовать некоторое количество квадратов. Как насчет 20 штук?

Что касается рисования каждого квадрата, мы будем полагаться на новую функцию `drawSquare`, которую нам предстоит написать. Обратите внимание, что мы передаем этой функции как `canvas`, так и `context`. Чуть позже вы увидите, как мы их используем.

Часть Задаваемые Вопросы

В: Как именно работает `selectedIndex`?

О: Свойство `selectedIndex` элемента управления формы `select` возвращает номер параметра, выбранного пользователем в раскрывающемся списке. Каждый список параметров преобразуется в массив, при этом все объекты в массиве располагаются по порядку. Допустим, у вас имеется список выбора, включающий следующие варианты: "pizza", "doughnut" и "granola bar". Если вы выберете «doughnut», то `selectedIndex` вернет 1 (не забывайте, что нумерация в JavaScript-массиве

начинается с 0). Вероятно, вам потребуется не только индекс, но и значение параметра с этим индексом (в нашем случае — "doughnut"). Чтобы извлечь данное значение, сначала нужно воспользоваться индексом для извлечения элемента массива; в результате обратно вы получите объект параметра. Чтобы извлечь значение этого объекта, необходимо прибегнуть к свойству `value`, которое возвращает строку в атрибуте `value` параметра.



Развлечения с Магнитами

Воспользуйтесь своими псевдоматематическими способностями программиста для размещения в нужной последовательности приведенных ниже табличек. Вам нужно написать псевдокод для функции `drawSquare`. Данная функция принимает `canvas` и `context` и рисует в `canvas` один квадрат произвольного размера. Проверьте свои ответы в конце главы, прежде чем двинетесь дальше.

```
function drawSquare ( , context ) {
```

Одну из табличек мы уже разместили в нужном месте.

Расположите здесь таблички с псевдокодом в нужной последовательности!

"lightblue" является цветом квадратов в нашем дизайнерском оригинал-макете.

canvas

нарисовать квадрат, имеющий позицию x, y и ширину w

вычислить произвольную позицию y для квадрата внутри canvas

задать для `fillStyle` значение "lightblue"

вычислить произвольную ширину для квадрата

вычислить произвольную позицию x для квадрата внутри canvas

Написание функции drawSquare

Теперь, когда вы разобрались в приведенном выше псевдокоде, давайте воспользуемся тем, что вы уже знаете, для написания drawSquare:

Наша функция, у которой имеются два параметра — canvas и context.

Мы используем Math.random() с целью генерирования случайных чисел для ширины и позиции x, y квадрата. Более подробно об этом мы поговорим через несколько мгновений...

Здесь нам требуется произвольная ширина и позиция x, y для квадрата.

```
function drawSquare(canvas, context) {
    var w = Math.floor(Math.random() * 40);
    var x = Math.floor(Math.random() * canvas.width);
    var y = Math.floor(Math.random() * canvas.height);

    context.fillStyle = "lightblue";
    context.fillRect(x, y, w, w);
}
```

Мы выбрали 40 как наибольшее значение размеров квадратов, поэтому они не будут слишком большими.

Координаты x и y базируются на ширине и высоте canvas. Мы выбираем случайное число между 0 и значением ширины и высоты соответственно.

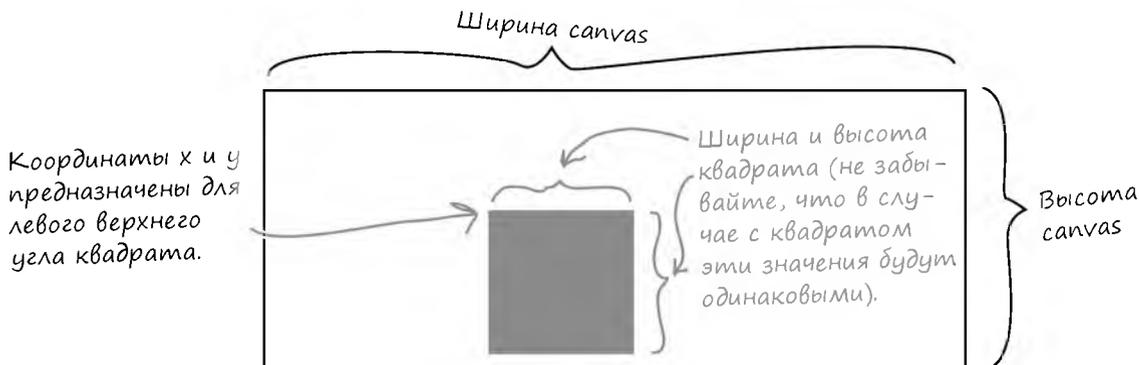
Применим к квадратам красивую светло-синюю заливку с помощью метода fillStyle, на который более пристально взглянем чуть позже...

И наконец, рисуем сам квадрат с помощью fillRect.

В книге «Изучаем HTML, XHTML и CSS» имеется полезная глава, посвященная работе с цветами (если вам потребуется освежить знания по данному вопросу).

Как мы решили, какие числа будем умножать на значения, генерируемые Math.random, чтобы получить значения ширины и позиции x, y нашего квадрата? В случае с шириной прямоугольника мы выбирали значение 40, так как оно обеспечивает небольшой размер относительно размеров canvas. Поскольку это квадрат, мы выбрали аналогичное значение для высоты. Мы также выбрали ширину и высоту canvas в качестве основы для значений позиции x, y, чтобы наш квадрат не выходил за границы canvas.

В своем коде вы можете свободно указать значение, отличное от 40!

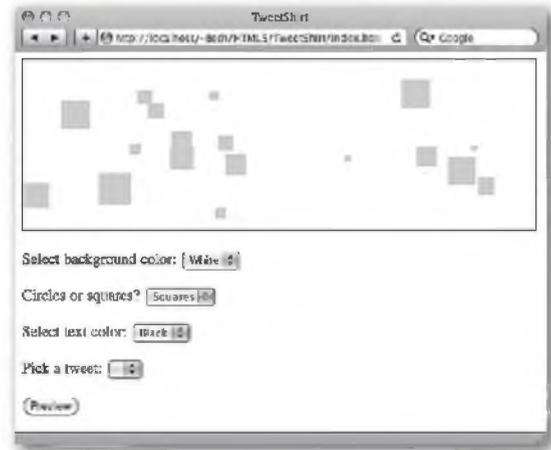


Время для тест-драйва!

Итак, затратив столько сил на создание кода, давайте, наконец, протестируем его. Откройте свой TweetShirt-файл `index.html` в браузере. Щелкните на кнопке Preview (Предварительный просмотр), после чего вы должны увидеть произвольные синие квадраты.

Вот что мы видим:

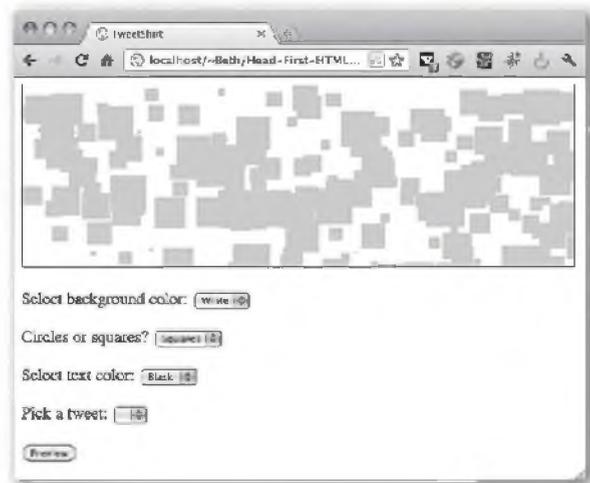
Отлично, все выглядит так, как нам нужно!



Постойте-ка, ведь если продолжать нажимать кнопку Preview (Предварительный просмотр), квадратов будет становиться все БОЛЬШЕ. А это не то, что нам нужно!



Он прав, у нас возникла небольшая проблема. Щелкните на кнопке Preview (Предварительный просмотр) несколько раз подряд — и вы увидите нечто подобное.

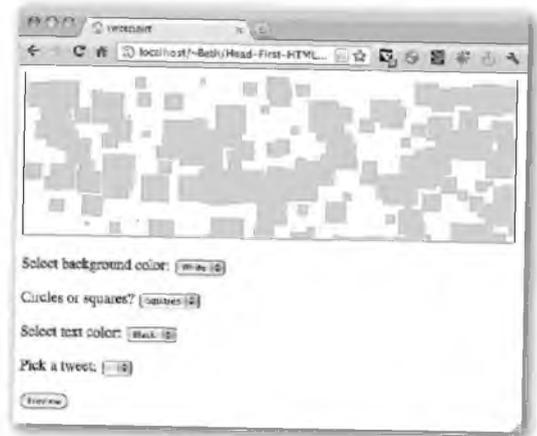


Почему мы видим старые и новые квадраты, когда нажимаем кнопку Preview?

Вообще-то получается классный эффект... по это не то, что нам пужно. Нам необходимо, чтобы новые квадраты заменяли старые каждый раз, когда мы щелкаем на кнопке Preview (Предварительный просмотр) (точно так же нам потребуется, чтобы новый твит заменял старый).

Основной момент здесь заключается в том, что пужно помнить, что мы папосим пиксели в элементе canvas. Когда вы пажимае кнопку Preview (Предварительный просмотр), вы берете canvas и рисуете в нем квадраты. При этом поверх всего, что уже имеется в canvas, просто папоситесь новые пиксели!

Вы уже зпаете все необходимое для того, чтобы исправить ситуацию прямо сейчас. Вот что мы сделаем:



- ① Будем извлекать выбранный цвет фона из объекта "backgroundColor".
- ② Будем осуществлять заливку фона canvas соответствующим цветом, используя fillStyle и fillRect каждый раз перед тем, как начать рисование квадратов.

Возьми в руку карандаш



Чтобы при каждом нажатии кнопки Preview (Предварительный просмотр) мы наблюдали в canvas только новые квадраты, потребуется залить фон canvas цветом, выбранным пользователем в меню "backgroundColor". Сначала реализуем функцию для заливки canvas выбранным цветом. Ниже приведен код, в котором вам необходимо устранить пробелы. Сверьте свои ответы с решением данного упражнения в конце главы, прежде чем двинетесь дальше.

```
function fillBackgroundColor(canvas, context) {
    var selectObj = document.getElementById("_____");
    var index = selectObj.selectedIndex;
    var bgColor = selectObj.options[index].value;
    context.fillStyle = _____;
    context.fillRect(0, 0, _____, _____);
}
```

Подсказка: то, что вы подчеркнете из выбранного параметра, будет строкой со значением цвета, которую вы сможете использовать точно так же, как "lightblue" для заливки квадратов.

Подсказка: мы хотим залить ВСЕ canvas соответствующим цветом!

Добавление вызова fillBackgroundColor

У вас имеется функция fillBackgroundColor, готовая к работе; теперь нам просто нужно убедиться, что ее вызов будет осуществляться из previewHandler. Ее вызов будет происходить в самом начале, благодаря чему мы получим аккуратный чистый фон, прежде чем начнем добавлять что-либо в canvas.

```
function previewHandler() {
  var canvas = document.getElementById("tshirtCanvas");
  var context = canvas.getContext("2d");
  fillBackgroundColor(canvas, context);

  var selectObj = document.getElementById("shape");
  var index = selectObj.selectedIndex;
  var shape = selectObj[index].value;

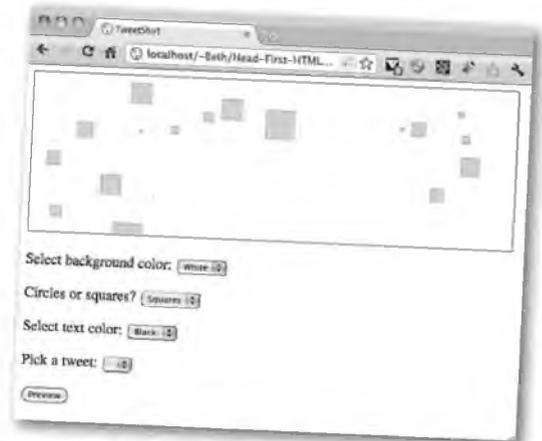
  if (shape == "squares") {
    for (var squares = 0; squares < 20; squares++) {
      drawSquare(canvas, context);
    }
  }
}
```

Добавляем вызов fillBackgroundColor до создания квадратов, поэтому то, что было нарисовано ранее, окажется скрытым под слоем заливки, и мы получим чистый фон для рисования новых квадратов.



Быстрый тест-драйв: убедимся, что наша новая функция fillBackgroundColor работает...

Добавьте новый код в файл tweetshirt.js, перезагрузите страницу в браузере, выберите цвет фона, а также Squares (Квадраты) в меню Circles or squares? (Круги или квадраты?), после чего нажмите кнопку Preview (Предварительный просмотр). Затем щелкните по ней снова. И при каждом следующем щелчке по кнопке вы должны будете видеть только новые квадраты.



Теперь при каждом нажатии кнопки Preview (Предварительный просмотр) мы будем видеть только новые квадраты.



Подсчитайте количество квадратов при нескольких нажатиях кнопки Preview (Предварительный просмотр). Наблюдалось ли хоть раз менее 20 квадратов? Такое вполне может быть.

Почему так происходит? Что можно сделать для устранения этой проблемы? (Вы же не хотите обманывать своих клиентов, не давая им их честные 20 квадратов, не так ли?)



JavaScript-свойство fillStyle под увеличительным стеклом

Более пристально взглянем на свойство fillStyle, поскольку вы впервые с ним столкнулись. Оно является свойством context, которое содержит определенное значение цвета для того, что вы рисуете в canvas.

Как и fillRect, свойством fillStyle мы управляем посредством context.

Но в отличие от fillRect, fillStyle является свойством, а не методом. Поэтому мы задаем для него значение, а не вызываем его.

А задаем мы для него значение цвета. Вы сможете использовать те же форматы цветов, что и в CSS. У вас будет возможность использовать такие имена, как lightblue, либо значения вроде #ccccff или rgb(0, 173, 239). Попробуйте!

`context.fillStyle = "lightblue";`

Обратите внимание, что, в отличие от CSS, значение необходимо заключать в кавычки, если вы не используете переменную.

Часть Задаваемые Вопросы

В: Я ожидал, что мы будем задавать цвет фона квадратов и canvas путем передачи значения цвета методу fillRect. Я не могу понять, как работает свойство fillStyle. Как оно влияет на то, что делает fillRect?

О: Отличный вопрос. В данном случае все немного по-другому, чем вы привыкли думать. Как вы помните, context представляет собой объект, управляющий доступом к canvas. Используя fillStyle и fillRect, вы сначала задаете свойство, которое говорит элементу canvas следующее: «Все, что будет нарисовано в тебе далее, должно иметь данный цвет». Таким образом, все, к чему вы станете применять заливку цветом (например, с помощью fillRect) после задания свойства fillStyle, будет иметь данный цвет, пока вы снова не измените цвет, присвоив fillStyle другое значение цвета.

В: Почему значение цвета следует заключать в кавычки, в то время как в CSS со значениями свойств так поступать не нужно?

О: Что ж, CSS является языком, отличным от JavaScript, и не ожидает, что вы будете использовать кавычки. Если вы не заключите значение цвета в кавычки, то JavaScript решит, что имя этого цвета является переменной, а не строкой, и попытается использовать значение переменной вместо имени цвета.

Допустим, у вас имеется переменная fgColor = "black". Вы могли бы написать context.fillStyle = fgColor, и это сработало бы, поскольку значением fgColor является "black".

Однако context.fillStyle = black не сработает, так как black не является переменной (если только вы не определите его как переменную, что может внести небольшую путаницу). Вы узнаете, что допустили эту ошибку, поскольку будет сгенерирована JavaScript-ошибка и выведено сообщение вроде Can't find variable: black (Не могу найти переменную: black) (но не стоит беспокоиться, так как все мы хотя бы раз допускали эту ошибку).

В: Ладно, я сдаюсь. Почему мы иногда наблюдаем менее 20 квадратов?

О: Позиция x, y и ширина квадратов являются произвольными. Одни квадраты могут перекрывать собой другие квадраты. Квадрат также может иметь позицию x, y со значениями соответственно 599, 199, из-за чего вы сможете увидеть только одного пиксел (поскольку остальная часть этого квадрата будет лежать вне canvas). Одни квадраты могут иметь ширину 1 пиксел, а другие — даже 0 пикселей, так как метод Math.random может возвращать значение 0. Либо вы можете сгенерировать два квадрата с абсолютно одинаковыми размерами и местоположением.

Однако в случае с нашим приложением все это является произвольным, поэтому мы и считаем, что все в порядке. А в ином случае нам, возможно, пришлось бы позаботиться о том, чтобы такого не случилось.

Тем временем, возвращаясь к TweetShirt.com...



Джим: Я впечатлен тем, как мало кода потребовалось. Только представь, если мы бы делали все это по-старому, на стороне сервера, то до сих пор возились бы с нашим сервером.

Фрэнк: Похоже, нам также представилась хорошая возможность заняться созданием кругов в процессе дизайна; в конце концов, ведь их рисование будет осуществляться точно так же, как и квадратов.

Джим: Согласен, а где Джуди? Опа, вероятно, уже знает API-интерфейс для рисования кругов. Хотя с другой стороны, для этого нам, возможно, потребуется лишь вызывать метод `fillCircle`.

Фрэнк: Мне кажется, так и есть! Кому пужпа эта Джуди, если у нас есть метод `fillCircle`!



Пару часов спустя...

Фрэнк: Не понимаю, что происходит, я дважды все проверил, но что бы я ни делал, при вызове `fillCircle` в `canvas` ничего не появляется.

Джудн: Что ж, дай-ка я взгляну на твой метод `fillCircle`.

Фрэнк: Что ты имеешь в виду под «моим методом»? Нет у меня никакого своего метода, я использую метод непосредственно из API-интерфейса `Canvas`.

Джудн: В API-интерфейсе `Canvas` нет метода `fillCircle`.

Фрэнк: А я предполагал, что есть, ведь существует же метод `fillRect...`

Джудн: Теперь ты знаешь, к чему могут привести предположения. Запусти-ка браузер и введи следующий адрес, по которому всегда можно пайти информацию относительно соответствующего API-интерфейса: <http://dev.w3.org/html5/2dcontext/>.

...Так или иначе, рисование круга — немного более сложный процесс, чем вызов лишь одного метода. Тебе сначала нужно изучить, что представляют собой контуры и дуги.

Джим (входя в кабинет): Джуди, а Фрэнк рассказал тебе о нашей проблеме с рисованием кругов?

Фрэнк: Да, Джим, *enoughway ithway ethay ircleay**

↑ Чтобы понять смысл этой фразы, рекомендуем вам воспользоваться услугами сайта pigliatin.bavetta.com.

* Enough with the circle! — «Хватит ходить по кругу!». — *Примеч. перев.*

Черчение контуров

Прежде чем перейти к кругам, необходимо поговорить о коптурах и дугах. Давайте начнем с коптуров и парисуем несколько треугольников. Если вам потребуется парисовать треугольпик в `canvas`, то знайте, что метода `fillTriangle` не существует. Однако треугольпик все же можно парисовать, спачала начертив его *контур*, который затем нужно будет *обвести*, чтобы парисовать треугольпик в `canvas`.

Что все это означает? Допустим, вы хотите очень аккуратно парисовать что-то на холсте, для чего можете взять карандаш и парисовать едва заметные очертания нужной вам фигуры (будем называть их коптуром). Вы начертите их так легкопко, что едва сможете разглядеть эти линии. Затем, когда коптур вас устроит, вы возьмете ручку (с толщипой стержля и цветом черпил по своему выбору) и обведете коптур, чтобы все смогли увидеть ваш треугольпик (или любую другую фигуру, которую вы начертили карандашом).

Именно так осуществляется рисование произвольных фигур с помощью линий в элементе `canvas`. Давайте нарисуем треугольпик и посмотрим, как это работает.

Используем метод `beginPath`, чтобы сказать `canvas` о том, что начинаем чертить новый контур.

```
context.beginPath();
context.moveTo(100, 150);
```

Используем метод `moveTo` для перемещения «карандаша» в определенную точку в `canvas`. Можете считать, что мы как бы опускаем кончик карандаша в данную точку.

Здесь мы опускаем кончик карандаша в точку с координатами $x = 100$ и $y = 150$. Это будет первая точка контура.

Метод `lineTo` чертит контур начиная с текущего местоположения кончика карандаша и двигаясь к новой точке в `canvas`.

```
context.lineTo(250, 75);
```

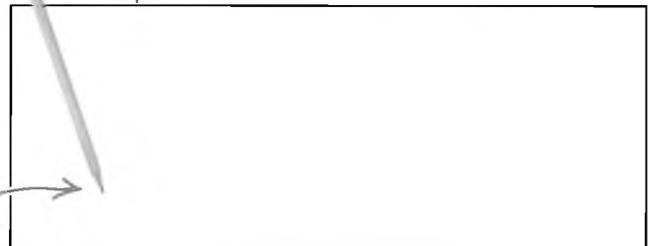
Кончик карандаша находился в точке с координатами $x = 100$ и $y = 150$, от которой мы чертим контур, двигая инструмент к точке с координатами $x = 250$, $y = 75$.

Я могу начертить любые нужные вам контуры.

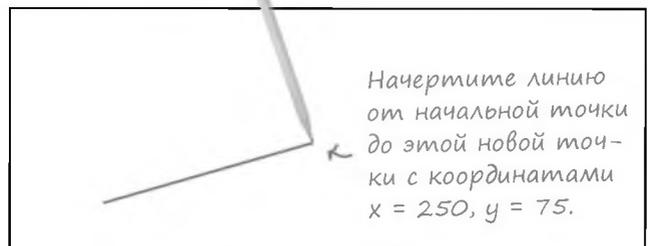


Карандаш, с помощью которого мы будем чертить контур.

Наш canvas



Начертите линию от начальной точки до этой новой точки с координатами $x = 250$, $y = 75$.



как рисовать с помощью контуров

У нас имеется одна сторона треугольника, нужно начертить еще две. Давайте снова воспользуемся методом `lineTo` и нарисуем вторую сторону:

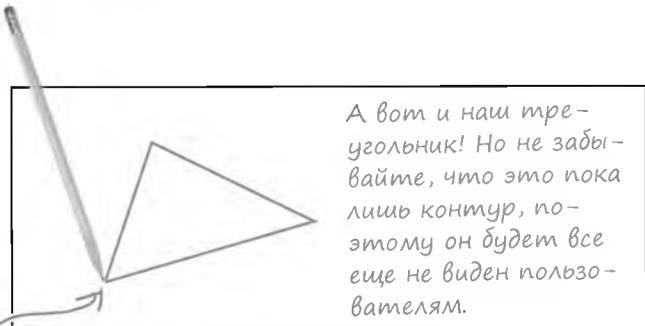
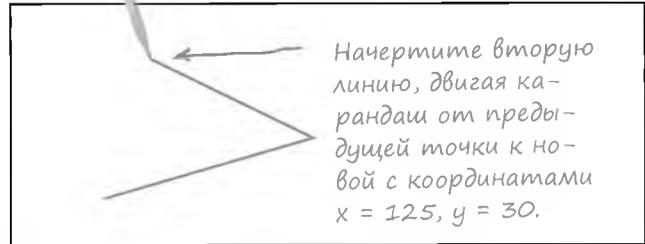
```
context.lineTo(125, 30);
```

Чертим линию от текущей позиции кончика карандаша (250, 75), двигая его к новой точке с координатами $x = 125$, $y = 30$. В результате у нас получится вторая сторона треугольника.

Мы почти достигли цели! Нам осталось лишь начертить еще одну линию, чтобы завершить треугольник. Для этого просто замкнем контур, воспользовавшись методом `closePath`.

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

Метод `closePath` соединяет начальную точку контура (100, 150) с конечной (125, 30).



Упражнение

Итак, у нас есть контур! И что теперь?

Теперь вы будете использовать этот контур для рисования линий и заливки своей фигуры цветом, конечно же! Создайте простую HTML5-страницу с элементом `canvas` и наберите весь приводившийся чуть ранее код. Затем проведите тестирование.

```
context.beginPath();
context.moveTo(100, 150);
context.lineTo(250, 75);
context.lineTo(125, 30);
context.closePath();

context.lineWidth = 5;
.....

context.stroke();
.....

context.fillStyle = "red";
.....

context.fill();
```

Вот код, приводившийся ранее.

А здесь приведен новый код. Опишите, что делает каждая из этих строк. Загрузите страницу. Ваши ответы оказались правильными? Решение упражнения — в конце главы.



Просто, чтобы быть в курсе: я думала, мы будем пытаться нарисовать круги. Какое отношение все эти контуры имеют к созданию кругов?

Чтобы нарисовать круг, сначала нужно начертить его контур.

Сейчас мы собираемся показать вам, как начертить контур круга. Научившись делать это, вы сможете создавать любые контуры, какие только захотите.

Немного дополнительной информации для вас. Вы уже знаете, как начертить контур, не так ли? Это, как вы видели ранее, делается с помощью следующего кода:

```
context.beginPath();
```

Но мы пока не говорили вам о том, что в объекте `context` имеется еще один метод — `arc`:

```
context.arc(150, 150, 50, 0, 2 * Math.PI, true);
```

И что же он делает? На следующей странице мы разьясим вам детали. Но, как вы сами могли догадаться, данный метод позволяет чертить контур вдоль окружности.

Из уроков геометрии вы наверняка помните, что длина окружности равна $2\pi R$? Просто ненадолго отложите в подсознание данный нюанс...

Подробное исследование метода arc

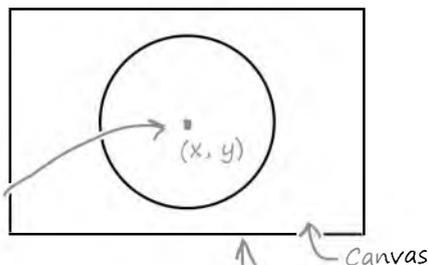
Взглянем на метод arc и изучим его параметры.

```
context.arc(x, y, radius, startAngle, endAngle, direction)
```

Суть метода arc заключается в том, что он позволяет определять, как будет чертиться требуемый контур вдоль окружности. Посмотрим, как именно каждый из его параметров содействует этому.

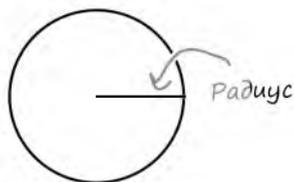
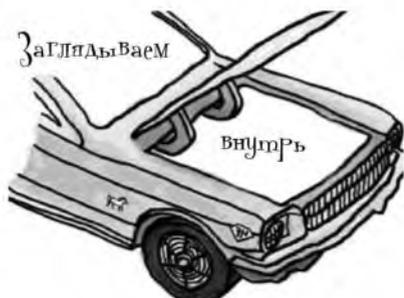
x, y Параметры x и y определяют, где будет находиться центр круга в вашем canvas.

Это позиция x, y центра вашего круга.

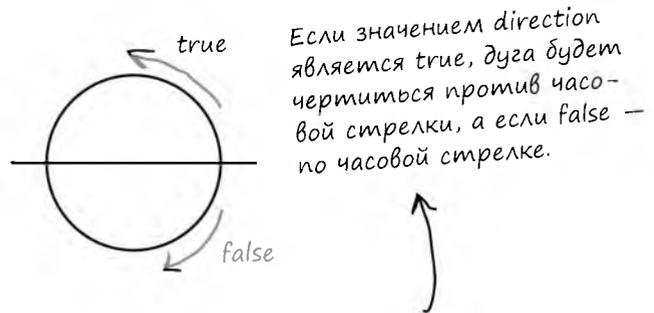


context.arc(x, y, radius,

radius Данный параметр используется для определения 1/2 ширины круга.



direction Позволяет определять, как будет чертиться дуга: в направлении против часовой стрелки или же по часовой стрелке. Если значением `direction` является `true`, то рисование будет происходить против часовой стрелки; если же `false` — по часовой стрелке.



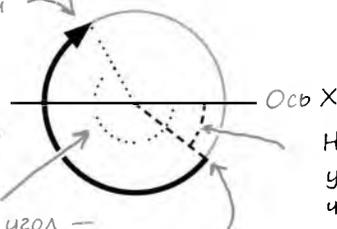
startAngle, endAngle, direction)

startAngle, endAngle

Если значением `direction` является `true`, дуга будет чертиться против часовой стрелки, а если `false` — по часовой стрелке

Конечная точка нашей дуги

Дуга, которую мы хотим нарисовать



Конечный угол — это угол между осью X и конечной точкой дуги.

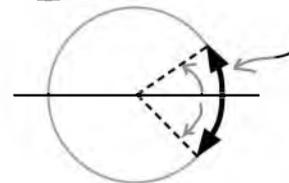
Начальная точка нашей дуги

Начальный угол — это угол между осью X и начальной точкой дуги.

Ниже приведен важный момент!

Не пропускайте это. Углы могут измеряться в отрицательных величинах (в направлении против часовой стрелки от оси X) либо в положительных величинах (по часовой стрелке от оси X). Это не то же самое, что параметр `direction` в случае с дугой! (Вы убедитесь в этом на следующей странице.)

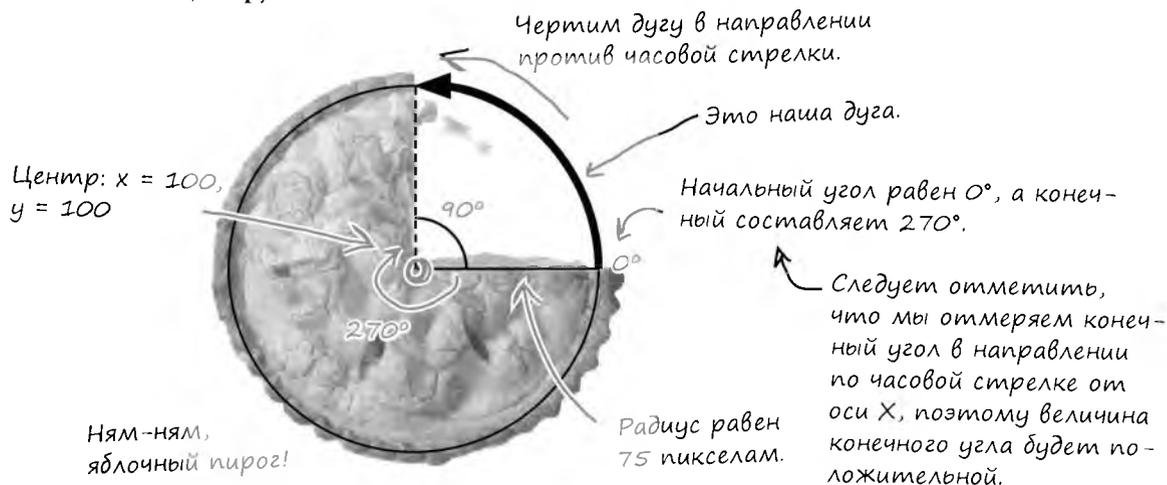
Угол, отмеряемый в направлении против часовой стрелки от оси X, будет иметь отрицательную величину (например, -35°).



Угол, отмеряемый в направлении по часовой стрелке от оси X, будет иметь положительную величину (например, 45°).

Небольшой пример использования метода arc

Что нам сейчас нужно, так это хороший пример. Допустим, вы хотите начертить дугу на круге, центр которого имеет позицию $x = 100$, $y = 100$. При этом вам нужно, чтобы ширина круга была 150 пикселей (то есть он имел радиус 75 пикселей). А дуга, которую вы хотите начертить, будет составлять лишь 1/4 круга.



Напишем вызов метода arc, который будет рисовать нашу дугу:

1 Начнем с позиции x , y , которую имеет центр круга: 100, 100.

```
context.arc(100, 100, __, __, _____, __);
```

2 Далее нам потребуется радиус круга, равный 75 пикселям.

```
context.arc(100, 100, 75, __, _____, __);
```

2 А что с нашими начальным и конечным углами? Начальный угол будет равен 0° относительно оси X. Конечный угол — это угол между осью X и конечной точкой нашей дуги. Поскольку наша дуга является 90-градусной, конечный угол будет равен 270° ($90^\circ + 270^\circ = 360^\circ$) (если бы мы измеряли его в отрицательных величинах, то есть в направлении против часовой стрелки, то конечный угол был бы в итоге равен -90°).

```
context.arc(100, 100, 75, 0, degreesToRadians(270), __);
```

1 Наконец, поскольку дуга будет чертиться в направлении против часовой стрелки, указываем значение true.

```
context.arc(100, 100, 75, 0, degreesToRadians(270), true);
```

К этой функции мы вернемся через несколько мгновений. Она просто преобразует градусы (которые нам привычны) в радианы (которые предпочитает context).

Я говорю «градус», вы говорите «радиан»

Все мы каждый день говорим об углах, связанных с кругами: «Классный разворот на 360 градусов», или «Я паправился по этому нути и развернулся на целых 180 градусов», или... ну, в общем, вы поняли. Однако проблема заключается в том, что мы думаем в градусах, а context в случае с canvas — в радианах.

Сейчас мы могли бы сказать вам, что:

$$360 \text{ градусов} = 2\pi \text{ радиан}$$

и после этого вы были бы готовы посчитать в уме градусы в радианах в случае необходимости. Если по какой-то причине вы не захотели бы делать это в уме, то знайте, что существует удобная функция, которая выполнит данную работу за вас:

```
function degreesToRadians(degrees) {
    return (degrees * Math.PI)/180;
}
```

Как помните, с этой функцией мы мимоходом стали кивались в главе, посвященной API-интерфейсу Geolocation.

Чтобы перевести градусы в радианы, нужно умножить их на π и разделить на 180.

Классный разворот на 360 градусов! То есть я хотел сказать, классный разворот на 2π радиан!



Радиан — это просто еще одна единица измерения углов. Один радиан равен $180/3,14159265...$ (то есть числу 180, разделенному на π).

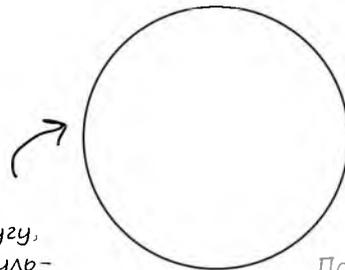
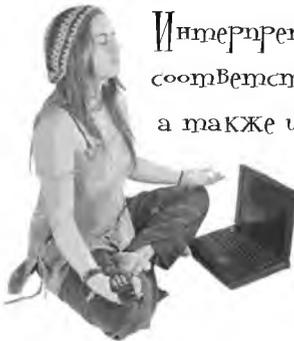
Используйте эту функцию, когда захотите думать в градусах, но будете оперировать радианами при рисовании дуги.

На с. 347 вы видели, как мы использовали $2 * \text{Math.PI}$ для определения конечного угла дуги на круге. Вы могли бы поступить так же... либо просто использовать `degreesToRadians(360)`.

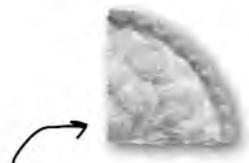
СТАНЬ браузером

Интерпретируйте приведенный ниже вызов Метода `arc` и напишите все соответствующие значения, которые будут характеризовать круг, а также изобразите дугу, создаваемую в результате этого вызова.

```
context.arc(100, 100, 75, degreesToRadians(270), 0, true);
```



Напишите все параметры данного круга и нарисуйте дугу, которая будет создана в результате вызова метода `arc`.



Подсказка: что останется от пирога, если съесть этот кусок?

Возвращаемся к написанию TweenShift-кода для рисования кругов

Теперь, когда вы знаете, как рисовать круги, пора вернуться к TweenShift и добавить новую функцию — рисовать, сначала потребуется определить, что пользователь выбрал Circles (Круги) в меню Circles or Squares? (Круги или квадраты?). Добавим соответствующий код в функцию preViewHandler.

Отредактируйте файл tweetshift.js и добавьте приведенный ниже новый код:

```
function preViewHandler() {
  var canvas = document.getElementById("tweetCanvas");
  var context = canvas.getContext("2d");
  fillBackgroundColor(canvas, context);

```

```
var selector = document.getElementById("shape");
```

```
var index = selector.selectedIndex;
```

```
var shape = selector[index].value;
```

```
if (shape == "squares") {
```

```
  drawSquare(canvas, context);
```

```
  } else if (shape == "circles") {
```

```
    drawCircle(canvas, context);
```

Данный код почти идентичен коду для проверки того, выбрал ли пользователь Circles (Квадраты) в меню Circles or squares? (Круги или квадраты?). Если пользователь выбрал Circles (Круги), а не Squares (Квадраты), мы напишем drawSquare(canvas, context) вместо drawCircle(canvas, context). Функции drawCircle (как и drawSquare) можно передать canvas и контекст функции и точно так же, как drawCircle, это в случае с функцией drawSquares.

Какой начальный и конечный углы вы будете использовать для рисования полного круга?
 Какое направление вы выберете: против часовой стрелки или по часовой стрелке? Имеет ли это значение?

Ответ: Рисование угла осуществляется с использованием начального угла 0° и конечного угла 360°. Неважно, какое направление вы выберете, поскольку будете рисовать полный круг.



}
}

Пишем функцию drawCircle...

Теперь напишем функцию drawCircle. Не забывайте, что здесь нам потребуется нарисовать лишь один произвольный круг. Остальной код обеспечит вызов данной функции 20 раз.

```
function drawCircle(canvas, context) {
  var radius = Math.floor(Math.random() * 40);
  var x = Math.floor(Math.random() * canvas.width);
  var y = Math.floor(Math.random() * canvas.height);

  context.beginPath();
  context.arc(x, y, radius, 0, degreesToRadians(360), true);

  context.fillStyle = "lightblue";
  context.fill();
}
```

Подобно тому как мы поступали в случае с квадратами, мы указываем 40 в качестве максимальной величины радиуса, чтобы наши круги не получились слишком большими.

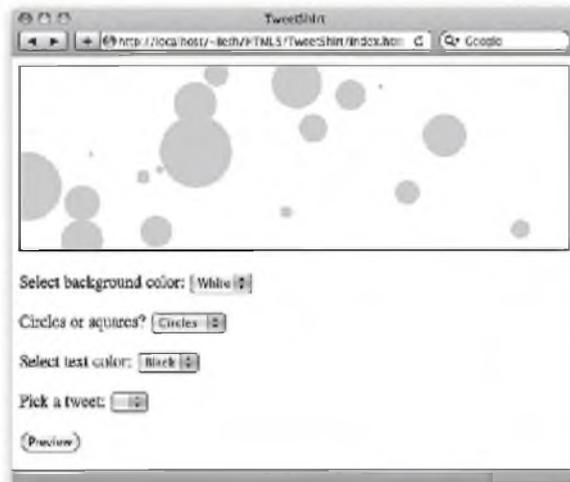
Опять-таки, координаты x и y центра круга базируются на ширине и высоте canvas. Мы выбираем произвольные числа между 0 и шириной и высотой соответственно.

Используем конечный угол 360° для создания полного круга. Рисуем в направлении против часовой стрелки, однако в случае с кругом неважно, какое направление мы выбрали.

Снова используем "lightblue" в качестве значения для fillStyle и заливаем контур с помощью context.fill().

... и проводим тест-драйв! 

Наберите приведенный ранее код (и не забудьте добавить функцию degreesToRadians), сохраните его, а затем загрузите в своем браузере. Вот что мы видим (поскольку это произвольные круги, ваши будут выглядеть по-другому):





Перерыв

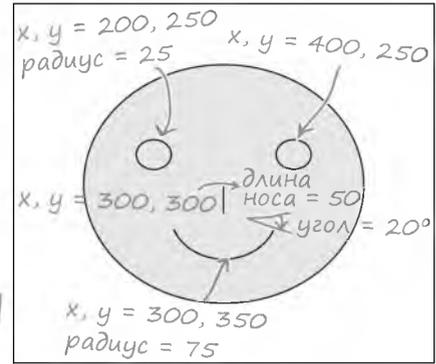


Небольшой перерыв на печенья

Уф! Страницы, по которым мы с вами только что проделали путь, были довольно увлекательными. Не знаем, как вы, а мы хотим печенья. Как насчет небольшого перерыва? Но не думайте, что мы не поручим вам нечто интересное, пока вы будете его есть (на следующей странице справа вас ждет задание).

Так что откиньтесь на спинку стула, сделайте небольшой перерыв, и «вгрызитесь» в задание пока ваш мозг и желудок будут ненадолго заняты кое-чем другим. Затем возвращайтесь, и мы закончим TweetShirt-код!

Справа изображено улыбающееся лицо (или печенье с кусочками шоколада в форме улыбающегося лица, если вам так больше нравится). Приведенный внизу код для рисования улыбающегося лица почти завершен; нам нужна ваша помощь, чтобы устранить имеющиеся в нем пробелы. После работы, проделанной в этой главе, у вас есть все необходимое для того, чтобы справиться с данной задачей. Закончив, вы сможете сверить свои ответы с решением этого упражнения в конце главы.



```
function drawSmileyFace() {
    var canvas = document.getElementById("smiley");
    var context = canvas.getContext("2d");

    context.beginPath();
    context.arc(300, 300, 200, 0, degreesToRadians(360), true);
    context.fillStyle = "#ffffcc";
    context.fill();
    context.stroke();

    context.beginPath();
    context.arc(____, ____, 25, ____, _____, true);
    context.stroke();

    context.beginPath();
    context.arc(400, ____, ____, ____, _____, ____);
    context.stroke();

    context.beginPath();
    context.____(____, ____);
    context.____(____, ____);
    context.____();

    context.beginPath();
    context.____(300, 350, ____, degreesToRadians(____), degreesToRadians(____), ____);
    context.stroke();
}
```

Вот что нам требуется. По коду дела у вас может возникнуть желание испечь настоящее печенье с кусочками шоколада в форме улыбающейся рожицы...

Круг лица. Здесь мы уже устранили за вас один из пробелов. Обратите внимание, что мы применили к кругу заливку желтым цветом.

← Левый глаз

← Правый глаз

← Нос

← Рот. Самый мудреный аспект!

Добро пожаловать обратно...

Итак, вы вернулись отдохнувшими. Мы с вами находимся на заключительном этапе. Нам осталось лишь обеспечить отображение твитов и прочего текста при предварительном просмотре футболки в canvas.

Теперь, чтобы добавить в canvas твит, нам сначала нужно извлечь последние твиты пользователя, из которых можно будет выбрать желаемый, для чего воспользуемся JSONP. Из главы 6 вам уже известно, как это делается (если требуется освежить свои знания, вернитесь к главе 6). Вот что нам нужно будет сделать:

Говоря о вкусностях, помните JSONP-код, который мы готовили в главе 6? Пора достать его из «духовки».



- 1 **Добавить <script> в нижнюю часть файла tweetshirt.html для вызова API-интерфейса JSONP Twitter. Мы будем запрашивать самые последние статусные обновления определенного пользователя.**
- 2 **Реализовать функцию обратного вызова для извлечения твитов, которые Твиттер будет присылать в ответ. Имя этой функции обратного вызова мы используем в URL-адресе для <script> во время шага 1.**

Наш HTML-файл для TweetShirt

```

<html>
...
<body>
  <form>
  ...
  </form>
  <script src="http://twitter.com/statuses/user_timeline/wickedsmartly.json?
  callback=updateTweets">
  </script>
</body>
</html>

```

Представьте, что здесь находится ваш элемент <head>, а здесь — ваш элемент <form> (мы просто решили сэкономить место).

JSONP-вызов; он работает путем извлечения JSON-данных, получаемых посредством обращения к URL-адресу Твиттера, с последующей передачей этих JSON-данных функции обратного вызова (которую мы определим через несколько мгновений).

Вызов API-интерфейса Twitter. Мы запрашиваем временную шкалу пользователя, что даст нам последние статусы.

Можете заменить это своим именем пользователя или другим, если пожелаете.

Функция обратного вызова, которой будут передаваться JSON-данные.

Наберите все это в одной строке в своем текстовом файле (уместить все в одну строку в книге не получилось).

Здесь много чего происходит. Если вы не все понимаете, снова взгляните, как работает JSONP, обратившись к главе 6.

Извлечение твитов

Мы уже покопчили со сложной работой, заключающейся в извлечении твитов из Твиттера. Теперь необходимо добавить их в элемент `<select>`, используемый для выбора твитов, в элементе `<form>` нашей страницы. Еще раз повторим: когда происходит вызов функции обратного вызова (в нашем случае — `updateTweets`), Твиттер передает ей ответ, содержащий твиты в формате JSON.

Ответ Твиттера представляет собой массив твитов. Каждый твит включает в себя массу данных; мы будем использовать текст твита.

Отредактируйте файл `tweetshirt.js` и добавьте функцию `updateTweets` в его пижую часть:

Функция обратного вызова

Данной функции передается ответ, содержащий твиты с временной шкалы пользователя в виде массива твитов.

Извлекаем ссылку на `tweetsSelection` из `<form>`.

```
function updateTweets(tweets) {
  var tweetsSelection = document.getElementById("tweets");

  for (var i = 0; i < tweets.length; i++) {
    tweet = tweets[i];
    var option = document.createElement("option");
    option.text = tweet.text;
    option.value = tweet.text.replace("\\"", "\"");

    tweetsSelection.options.add(option);
  }

  tweetsSelection.selectedIndex = 0;
}
```

В случае с каждым твитом в массиве твитов мы сделаем следующее.

Извлекаем твит из массива.

Создаем новый элемент `option`.

Задаем для его `text` значение `tweet`.

А также задаем для его `value` аналогичное значение, но только немного обрабатываем при этом строку с целью замены двойных кавычек одинарными (чтобы избежать проблем, касающихся форматирования в HTML).

Затем берем новый элемент `option` и добавляем его в `tweetsSelection` в `<form>`.

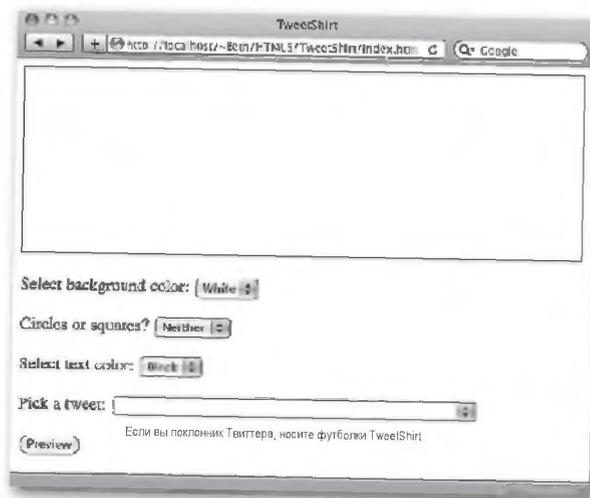
После того как мы проделаем это с каждым твитом, у нас получится элемент `<select>`, содержащий `option` для каждого твита.

Наконец, мы заботимся о том, что первым будет идти выбранный твит, путем присваивания свойству `selectedIndex` элемента `<select>` значения `0` (номер первого элемента `option`, который в нем содержится).

Тест-драйв TweetShirt

Проведем небольшой тест-драйв. Убедитесь, что вы добавили весь соответствующий код в `tweetshirt.js` и `index.html`. Также удостоверьтесь, что вы используете имя пользователя Твиттера, у которого имеются свежие твиты, в вашем URL-адресе `script src` (благодаря чему вы сможете быть уверены в том, что обязательно увидите твиты!). Загрузите страницу и щелкните на элементе для выбора твитов. Вот что мы видим:

Меню выбора твитов с НАСТОЯЩИМИ твитами. Класс! 



Ребята, это здорово. Мы можем рисовать квадраты и круги, а Джим позаботился об извлечении твитов из Твиттера! А что дальше?

Планшетный компьютер Фрэнка

Джим: Мы почти достигли цели. Необходимо разобраться с текстом, который будет отображаться. У нас имеются два сообщения: Я повелся на этот твит... и ... а в результате получил эту паршивую майку!, а также твит, который пользователь выберет для отображения. Сейчас нам нужно понять, как обеспечить их отображение, не говоря уже о применении стилизации к тексту.

Фрэнк: Полагаю, что мы можем вставить текст в `canvas`, а затем применить к нему `CSS`?

Джо: Не думаю, что все будет происходить именно так. Как известно, `canvas` является областью для рисования, и я не думаю, что мы сможем поместить в него текст и стилизовать его; нам придется рисовать текст в `canvas`.

Джим: Недавно я получил хороший урок и на этот раз павел справки, какой API-интерфейс предназначен для работы с текстом.

Джо: Это хорошо, но сам я пока еще не смотрел что к чему; как он работает?

Джим: Помпишь метод `arg`? С его помощью `пam` и придется рисовать весь текст.

Фрэнк: Ты что, шутишь? Похоже, теперь мы провозимся весь уикенд.

Джим: Как я тебя подловил! Если серьезно, то существует метод `fillText`, который припимает текст для рисования в `canvas` паряду с позицией `x`, `y`, где он будет рисоваться.

Джо: Звучит довольно просто. А как пасчет различий в стиле? Насколько я помню оригинал-макет, шриффт текста твита в нем был курсивный, а остального текста — полужирный.

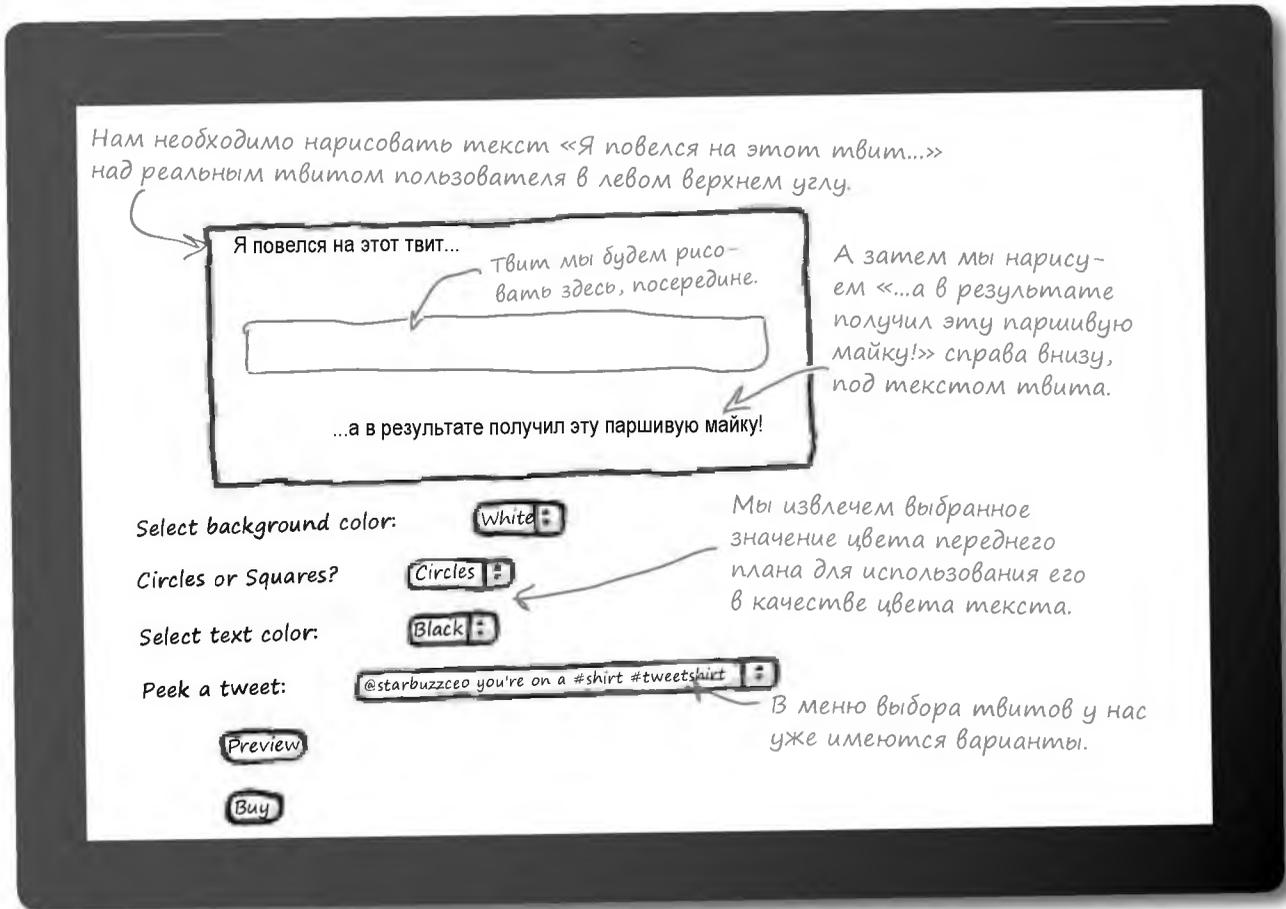
Джим: Нам нужно еще пемного покопаться, поскольку существуют разпообразные методы для задания выравнивания, шрифтов и стилей, но я не очень хорошо знаю, как их использовать.

Фрэнк: Если подумать, то я, возможно, мог бы помочь, но как это сделать без `CSS`?

Джим: Прости, но как сказал Джо, это API-интерфейс для рисования в `canvas`, и он не использует `HTML` или `CSS`.

Джо: Тогда давайте оставовимся на данном API-интерфейсе, после чего можем попробовать нарисовать текст «Я повелся на этот твит...» в `canvas`. Фрэнк, присоединяйся к `пam`, все не так уж и плохо, и я уверен, что `пam` пригодятся твои знания в области шрифтов, стилей и тому подобного.

Фрэнк: Конечно, я к вашим услугам!





Что меня смущает относительно рисования текста в `canvas`, так это то, что мы всегда акцентировали внимание на отделении содержимого от представления. А в случае с `canvas` похоже, что они являются одним и тем же. Я хочу сказать, что они не кажутся отдельными.

Это точно подмечено.

Давайте разберемся, почему так обстоит дело. Как вы помните, `canvas` является инструментом, позволяющим представлять графику в браузере. Все в `canvas` считается представлением, а не содержимым. Таким образом, несмотря на то что обычно вы рассматриваете текст — и, конечно же, твиты — как содержимое, в данном случае вам следует рассматривать его как представление. Он является частью дизайна. Подобно художнику, который использует художественные заглавные буквы как часть своей картины, вы используете твиты как часть дизайна футболки.

Одна из главных причин того, что отделение представления и содержимого — хорошая идея, заключается в том, что браузер может с умом подходить к представлению содержимого в различных ситуациях: например, он может представлять статью с новостного веб-сайта одним образом на большом экране и другим — на экране телефона.

В случае дизайна футболки нам необходимо, чтобы содержимое `canvas` больше походило на изображение: оно должно отображаться одинаково независимо от того, каким образом вы будете его просматривать.

Итак, давайте рисуем текст в `canvas` и запустим наш стартап!



Развлечения с Магнитами

Пришло время вашего первого эксперимента с текстом в canvas. Ниже приведен начатый нами код для метода drawText, который мы будем вызывать, чтобы нарисовать весь текст в canvas для предварительного просмотра. Посмотрите, сможете ли вы завершить данный код, чтобы нарисовать Я повелся на этот твит... И...а в результате получил эту паршивую майку! в canvas, а рисование реального твита пользователя мы оставим на потом. Проверьте свои ответы в конце главы, прежде чем идти дальше.

```
function drawText(canvas, context) {
    var selectObj = document.getElementById("_____");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;
    context._____ = fgColor;
    context._____ = "bold lem sans-serif";
    context._____ = "left";
    context._____ ("_____", 20, 40);
```

Подсказка: это позиция x, y для текста «Я повелся на этот твит...».

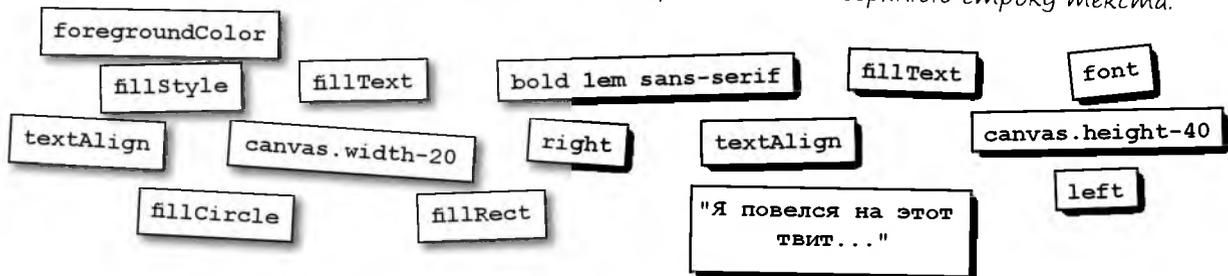
Подсказка: мы будем использовать курсивный шрифт с засечками для твита, но нам необходимо, чтобы этот текст имел полужирный шрифт без засечек.

Подсказка: мы хотим поместить текст в правый нижний угол.

На данный момент размещаем комментарии там, где будет располагаться код для рисования текста твита. ;

```
// Извлечь нужный вариант из меню выбора твитов
// Нарисовать твит
context.font = "_____";
context._____ = "_____";
context._____ ("...а в результате получил эту паршивую майку!",
    _____, _____);
```

Мы хотим нарисовать данный текст в 20 пикселях от правой стороны canvas и в 40 пикселях от основания canvas. чтобы он уравновешивал верхнюю строку текста.





Код под увеличительным стеклом

Теперь, когда вам представилась возможность нарисовать свой первый текст в canvas, пора более пристально взглянуть на методы и свойства для работы с текстом, доступные в API-интерфейсе Canvas. Как вы поняли из предыдущего задания, это довольно низкоуровневый API-интерфейс: вам приходится сообщать context, какой текст рисовать, какую позицию выбирать и какой шрифт использовать.

В данной секции мы детально рассмотрим такие свойства, как `textAlign`, `font`, `textBaseline`, а также методы `fillText` и `strokeText`, благодаря чему вы превратитесь в эксперта по тексту в canvas!

`textAlign`

Свойство `textAlign` определяет, где будет находиться якорная точка для текста. Значением по умолчанию является "start".

```
context.textAlign = "left";
```

Возможные значения данного свойства: "start", "end", "left", "right" и "center". Варианты "start" и "end" означают то же самое, что и слева и справа в языках, где все пишется и читается слева направо (например, английский). В языках же, где буквы располагаются справа налево (например, иврит), под значениями "start" и "end" будет пониматься соответственно справа и слева.



Выравнивание по левому краю



Выравнивание по центру



Выравнивание по правому краю

`fillText` и `strokeText`

Как и в случае с прямоугольниками, мы можем обводить текст и применять к нему заливку. Мы указываем текст для рисования, позицию `x`, `y` и опциональный параметр `maxwidth`, что обеспечит масштабирование текста, если он окажется шире, чем значение `maxwidth`.

```
context.fillText("Dog", 100, 100, 200);
```

```
context.strokeText("Cat", 100, 150, 200);
```

Текст с заливкой

Dog

Cat

Текст с обводкой

Если текст окажется шире 200 пикселей, он автоматически будет подогнан посредством масштабирования.

`font`

При задании значений для свойства `font` вы можете применять тот же формат, что и в CSS, — это удобно. Если вы будете задавать все значения для данного свойства, то среди них будут следующие: стиль шрифта, его толщина, размер, семейство — именно в таком порядке.

```
context.font = "2em Lucida Grande";
```

```
context.fillText("Tea", 100, 100);
```

```
context.font = "italic bold 1.5em Times, serif";
```

```
context.fillText("Coffee", 100, 150);
```

Tea

Coffee

В спецификации рекомендуется использовать только векторные шрифты (растровые шрифты не очень хорошо отображаются).

`textBaseline`

Свойство `textBaseline` устанавливает точки выравнивания в случае со шрифтом и определяет линию расположения ваших букв. Чтобы увидеть, как эта линия влияет на текст, попробуйте начертить ее в той же точке `x`, `y`, где вы рисовали текст.

```
context.beginPath();
```

```
context.moveTo(100, 100);
```

```
context.lineTo(250, 100);
```

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

```
context.textBaseline = "middle";
```

```
context.fillText("Alphabet", 100, 100);
```

Alphabet

← *alphabetic*

Alphabet

← *bottom*

Alphabet

← *middle*

Alphabet

← *top*

Возможные значения данного свойства: `"top"`, `"hanging"`, `"middle"`, `"alphabetic"`, `"ideographic"` и `"bottom"`. Значением по умолчанию является `"alphabetic"`. Поэкспериментируйте с разными значениями, чтобы выяснить, какое именно из них вам требуется (а также загляните в спецификацию для получения дополнительных сведений).

Применение drawText

Теперь, когда вы узнали еще об одном API-интерфейсе, паберите код, пад которым вы работали в предыдущем упражнении «Развлечения с магнитами». Вот как он будет выглядеть после того, как таблички с кодом окажутся па своих местах:

```
function drawText(canvas, context) {
    var selectObj = document.getElementById("foregroundColor");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;

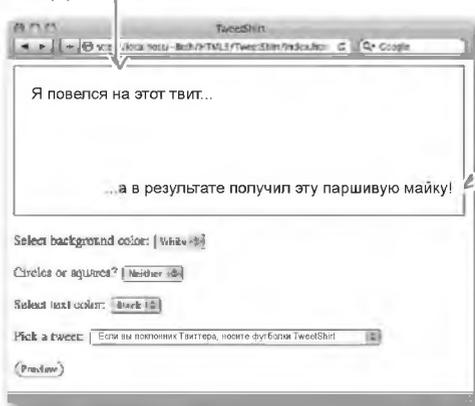
    context.fillStyle = fgColor;
    context.font = "bold 1em sans-serif";
    context.textAlign = "left";
    context.fillText("Я повелся на этот твит", 20, 40);

    context.font = "bold 1em sans-serif";
    context.textAlign = "right";
    context.fillText("а в результате получил эту паршивую майку!",
        canvas.width-20, canvas.height-40);
}
```

Через несколько мгновений мы поместим сюда код для рисования текста твита...

Напечатав дапный код, обновите функцию previewHandler с целью вызова функции drawText и проведите тестирование, загрузив дапный код в своем браузере. Вы должны увидеть примерпо то же самое, что и мы:

Вот наш текст. У нас получился текст, имеющий полужирный шрифт без засечек и корректное местоположение.



А внизу у нас получился текст с выравниванием по правому краю.

Возьми в руку карандаш



Попытайтесь самостоятельно завершить написание функции drawText. Вам необходимо извлечь выбранный твит, задать курсивный шрифт с засечками, который немного крупнее (1.2em), чем шрифт по умолчанию, а также убедиться в том, что текст будет выровнен по левому краю, и разместить его в точке $x = 30$, $y = 100$. Это последний шаг перед тем, как мы увидим в работе TweetShirt!

Напишите свой код вверху и не заглядывайте на следующую страницу! (Мы серьезно!)

Завершаем написание функции drawText

Вот приводившийся чуть выше код, написание которого завершили мы. Ну что, каков результат, если сравнить его с написанным вами? Если вы еще не пабрали свой код, то используйте код, приведенный ниже (либо свою версию, если предпочитаете), и перезагрузите файл `index.html`. Наш тест-драйв мы продемонстрируем вам на следующей странице.

```
function drawText(canvas, context) {
    var selectObj = document.getElementById("foregroundColor");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;

    context.fillStyle = fgColor;
    context.font = "bold 1em sans-serif";
    context.textAlign = "left";
    context.fillText("Я повелся на этот твит...", 20, 40);
```

Нам не потребуется выравнивать текст твита по левому краю; выравнивание по-прежнему будет обеспечиваться отсюда.

```
selectObj = document.getElementById("tweets");
index = selectObj.selectedIndex;
var tweet = selectObj[index].value;
context.font = "italic 1.2em serif";
context.fillText(tweet, 30, 100);
```

Извлекаем нужный параметр из меню выбора твитов.

Задаем более крупный курсивный шрифт с засечками...

...и рисуем текст, имеющий позицию 30, 100.

```
context.font = "bold 1em sans-serif";
context.textAlign = "right";
context.fillText("...а в результате получил эту паршивую майку!",
    canvas.width-20, canvas.height-40);
```

```
}
```



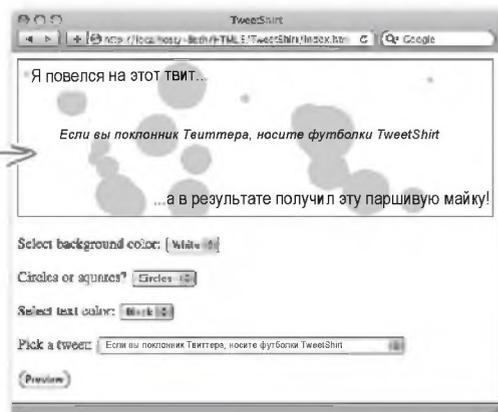
Фрэнк, скорее выбери предварительный просмотр. Я хочу увидеть наше приложение TweetShirt в работе!

Небольшой тест-драйв, а затем — ЗАВТРАК ЗАПУСК!

Надеемся, что вы видите то же самое, что и мы! Здорово, не правда ли? Проведите тестирование интерфейса с целью проверки качества созданного программного продукта: попробуйте всевозможные комбинации цветов и фигур, замените имя пользователя другим по своему усмотрению.

Вам кажется, что вы по-настоящему готовы переходить к запуску стартапа TweetShirt? Давайте сделаем это!

На футболке при предварительном просмотре отображается твит. Здорово!



Да!
Все работает!
Мы готовы к запуску стартапа!

Ребята, не хочу вас разочаровывать, но вы еще не все сделали. Вы также должны были поместить изображение птицы Твиттера на футболку!



Помните основательницу TweetShirt.com?



- 1 В первую очередь нам нужно поместить изображение. Мы поместили файл `twitterBird.png` в папку `TweetShirt`. Чтобы добавить его в `canvas`, нам сначала потребуется JavaScript-объект `Image`. Вот как он создается:

```
var twitterBird = new Image();
twitterBird.src = "twitterBird.png";
```

Создаем объект `Image`.

И задаем в качестве его источника изображение птицы Твиттера.

- 2 Следующий шаг уже должен показаться вам вполне естественным. Нам нужно нарисовать изображение в `canvas`, используя метод объекта `context` с именем, как вы уже догадались, `drawImage`.

```
context.drawImage(twitterBird, 20, 120, 70, 70);
```

Используем метод `drawImage`.

Вот наш объект `Image`.

Кроме того, указываем значения координат местоположения `x`, `y`, ширины и высоты.

- 2 Об изображениях вам нужно знать еще одну вещь: они не всегда загружаются мгновенно, поэтому вам будет необходимо удостовериться в том, что изображение полностью загрузилось, прежде чем вы приступите к его рисованию. Как мы ждем, пока что-то загрузится, перед тем как предпринять действие? Мы реализуем для этого обработчик `onload`:

```
twitterBird.onload = function() {
    context.drawImage(twitterBird, 20, 120, 70, 70);
};
```

Здесь мы говорим: «Когда изображение загрузится, выполнить данную функцию».

Рисуем изображение в `canvas`, используя метод `drawImage` объекта `context`.



Упражнение

Посмотрите, сможете ли вы собрать функцию `drawBird` из всех тех кусочков, которые нам предоставила Джуди. Функция `drawBird` принимает `canvas` и `context` и рисует изображение птицы в `canvas`. Исходите из того, что с помощью данной функции мы должны разместить "twitterBird.png" в месте с координатами $x = 20$, $y = 120$, при этом ширина и высота будут равны 70 пикселям. Мы уже написали за вас объявление метода и первую строку. Решение данного упражнения вы найдете в конце главы.

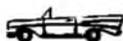
```
function drawBird(canvas, context) {
    var twitterBird = new Image();
```

Свой код
напишите
здесь. →

Убедитесь, что вы добавили вызов функции `drawBird` в функцию `previewHandler`.

```
}
```

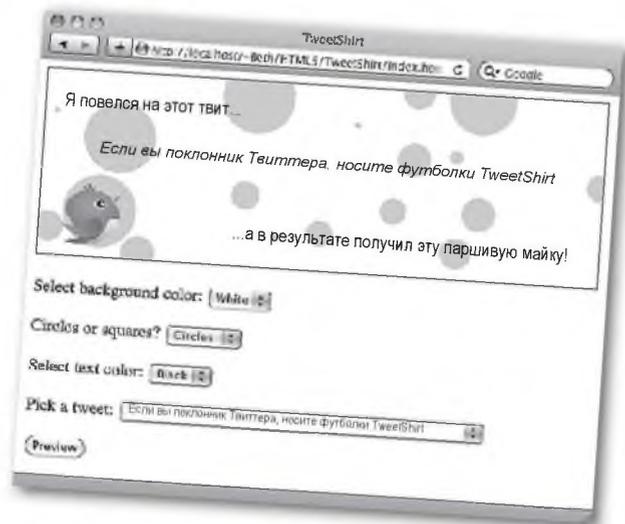
Еще один тест-драйв



Дважды проверьте свой код и проведите еще один тест-драйв! Здорово, теперь наше приложение действительно выглядит отшлифованным.

Сделайте несколько попыток; попробуйте выбрать `Circles` (Круги) или `Squares` (Квадраты) в меню `Circles or squares?` (Круги или квадраты?). Вы заметите, что мы использовали изображение в формате PNG с прозрачным фоном, чтобы круги и квадраты были видны, если они окажутся позади изображения птицы.

Здорово, мы значительно преуспели в разработке классного приложения. Но, как уже отмечалось ранее, мы рассчитываем на вас в реализации всего необходимо для ведения торговли через Интернет и выполнения заказов, связанных с футболками, и т. п. →



Часть Задаваемые Вопросы

В: Мы раньше не сталкивались с объектом `Image`. Вы использовали его при добавлении изображения в `canvas`. А что он собой представляет? Почему его нельзя было создать с помощью `document.createElement("img")`?

О: Оба упомянутых вами метода позволяют создавать объекты `Image`. JavaScript-конструктор `Image` предоставляет более прямой путь создания изображений из JavaScript и дает нам больший контроль над процессом загрузки (например, позволяет использовать обработчик для получения уведомления, когда загрузка изображения завершается). Наша цель заключается в создании изображения, а также в том, чтобы убедиться, что оно уже загрузилось, прежде чем мы приступим к его рисованию в `canvas`. Объект `Image` позволяет сделать это оптимальным путем.

В: `canvas` — это, конечно, здорово... однако он доставляет некоторые трудности по сравнению с HTML. Что-то сложнее базовых фигур, по-видимому, действительно будет сложно нарисовать.

О: Вы, несомненно, пишете код графики, когда программируете `canvas`. В отличие от браузера, который вместо вас заботится о массе деталей вроде добавления элементов на веб-страницу (благодаря чему вам не нужно самим все рисовать), вам придется говорить `canvas`, где все должно размещаться. Однако `canvas` обеспечивает широкие возможности создания графики почти что любого типа (в настоящий момент — 2D). Мы пребываем на начальном этапе существования `canvas`; вероятно, библиотеки JavaScript-кода смогут облегчить создание графики в `canvas` в будущем.

В: В случае с очень длинными твитами я заметил, что часть такого твита, выходящая за край `canvas`, просто исчезает из виду. Как это исправить?

О: Сначала нужно выяснить количество символов, содержащихся в твите, и если оно будет превышать определенное число, то разбить данный твит на несколько строк и рисовать каждую из них отдельно в `canvas`. Мы включили соответствующую функцию `splitIntoLines` в код, доступный по адресу wickedlysmart.com.

В: Я также заметил, что в некоторых твитах имеются HTML-сущности вроде `"` и `&`. Что все это значит?

О: API-интерфейс Twitter, используемый нами для извлечения твитов в виде JSON-данных, преобразует символы, которые пользователи печатают в своих твитах, в HTML-сущности. Вообще-то

это хорошая штука, поскольку любые специальные символы или даже кавычки, которые могли бы помешать нам получать твиты из JSON, представляются в виде сущностей. При отображении твитов с использованием HTML эти сущности отображаются в браузере как символы, которые вы и ожидаете увидеть, подобно тому как сущности, добавляемые вами в собственную страницу, также будут корректно отображаться в браузере. Однако в `canvas` они выглядят не так хорошо. К сожалению, в настоящее время в API-интерфейсе `Canvas` нет функции для преобразования данных сущностей обратно в соответствующие символы, поэтому вам придется делать это самостоятельно.

В: А можно ли сделать в `canvas` что-нибудь необычное, например снабдить текст или фигуры тенью?

О: Да! С помощью `canvas` вы можете делать массу необычных вещей, и эффект отбрасывания тени относится к их числу. Как вы и могли ожидать, создание тени осуществляется путем присваивания значений соответствующим свойствам `context`. Например, чтобы задать степень размытия, необходимо присвоить требуемое значение свойству `context.shadowBlur`. Местоположение тени задается с помощью свойств `context.shadowOffsetX` и `context.shadowOffsetY`, а чтобы установить цвет, нужно присвоить значение свойству `context.shadowColor`. К прочим возможностям `canvas` относится рисование градиентов, вращение фигур и закругление углов прямоугольников.

В: Что еще интересного можно делать в случае с `canvas`?

О: Много чего! В следующих главах мы расскажем еще о паре способов использования `canvas`. Кроме того, вы определенно захотите более подробно исследовать API-интерфейс `Canvas`, для чего можете зайти по адресу <http://dev.w3.org/html5/2dcontext/>.

В: А будет ли все создаваемое в `canvas` работать и на моем мобильном устройстве или мне придется переписывать его для мобильных пользователей?

О: Если на вашем мобильном устройстве установлен современный браузер (на всех устройствах вроде Android, iPhone и iPad как раз имеются такие браузеры), то все будет отлично работать (масштабирование страницы может быть отключено, однако функциональность сохранится). Замечательная особенность `canvas` заключается в том, что, поскольку вы рисуете «сырые» пиксели, все создаваемое вами будет везде отображаться одинаково (то есть во всех браузерах, которые поддерживают `canvas`).



Мне кажется, было бы здорово иметь возможность сохранять созданный дизайн футболки вместе с определенным положением всех этих квадратов на ней. Существует ли какая-либо методика, позволяющая сохранить нарисованное в canvas?

Для этого придется еще потрудиться.

На самом деле canvas призван быть простой поверхностью для рисования. Когда вы создаете фигуру, canvas видит ее как группу пикселей. Он ничего не знает об особенностях того, что вы рисуете, и не отслеживает никаких фигур, а просто создает пиксели, которые вы говорите ему создать (если вам знакомы такие термины из области графики, как «растровое рисование» и «векторное рисование», то вы поймете, что canvas осуществляет *растровое рисование*).

Если вам захочется обходиться с прямоугольниками в своем canvas как с набором объектов, который можно будет сохранить (или, возможно, вы захотите перемещать его или мапипулировать им), то вам потребуется сохранять информацию о фигурах и контурах по ходу рисования их в canvas. Вы сможете сохранить эти данные в JavaScript-объектах. Например, если вы захотите отследить произвольные круги, нарисованные вами в canvas при работе над TweetShirt, вам потребуется сохранить координаты местоположения x , y , значения радиуса и цвета каждого конкретного круга, чтобы вы смогли воссоздать его позже.

Похоже, что это и есть плап действий, который вам пужеп... ;)

Поздравляю, ребята, вам удалось это сделать! Созданное вами приложение работает даже на моем iPad, так что оно идеально подойдет для клиентов, находящихся в дороге. Я впечатлена. У нас намечается вечеринка по поводу запуска стартапа TweetShirt, поэтому присоединяйтесь к нам.



Основательница TweetShirt.com рада видеть, что созданное нами веб-приложение работает на ее iPad и iPhone! Если она рада, то и мы рады.

КЛЮЧЕВЫЕ МОМЕНТЫ



- `canvas` — это элемент, который вы размещаете в своей странице с целью создания области для рисования.
- `canvas` не будет иметь стиля по умолчанию или содержимого, пока вы не снабдите его им (таким образом, вы не увидите `canvas` на веб-странице до тех пор, пока не нарисуете что-нибудь в нем или не добавите рамку для него с помощью CSS).
- У вас на странице может иметься более одного `canvas`. Вам, конечно же, потребуется присвоить каждому из них уникальный идентификатор, чтобы к ним можно было обращаться с использованием JavaScript.
- Для задания размеров элемента `canvas` следует использовать его атрибуты `width` и `height`.
- Все, что вы захотите поместить в `canvas`, будет рисоваться с применением JavaScript.
- Для рисования в `canvas` вам сначала нужно будет создать контекст. В настоящее время вашим единственным выбором является контекст `2d`, однако в будущем возможно появление и других типов контекста.
- Контекст необходим для рисования в `canvas` потому, что он обеспечивает специфический тип интерфейса (например, `2d` и `3d`). Вы сможете выбирать нужный тип интерфейса для рисования в `canvas`.
- Обращаться к `canvas` следует с использованием свойств и методов объекта `context`.
- Чтобы нарисовать в `canvas` прямоугольник, нужно воспользоваться методом `context.fillRect`. Он создает прямоугольник с заливкой цветом.
- Чтобы нарисовать контур прямоугольника, вместо `fillRect` используйте метод `strokeRect`.
- Используйте `fillStyle` и `strokeStyle` для изменения задаваемого по умолчанию цвета заливки и обводки, которым является черный.
- Вы можете задавать цвета, используя тот же самый формат, что и в CSS (например, `"black"`, `"#000000"`, `"rgb(0, 0, 0)"`). Не забывайте заключать значение `fillStyle` в кавычки.
- Метода `fillCircle` не существует. Чтобы нарисовать круг в `canvas`, вам потребуется начертить дугу.
- Для рисования произвольных фигур или дуг сначала необходимо начертить контур.
- Контур — это невидимая линия или фигура, которую вы чертите для определения линии или области в `canvas`. Контур нельзя будет увидеть до тех пор, пока вы не обведете его или не примените к нему заливку.
- Для создания треугольника необходимо начать контур с помощью `beginPath`, а затем использовать `moveTo` и `lineTo` для рисования контура. Для соединения двух точек контура используйте `closePath`.
- Чтобы нарисовать круг, начертите 360-градусную дугу. Начальный угол будет 0° , а конечный — 360° .
- Углы задаются в `canvas` с использованием радианов, а не градусов, поэтому нужно преобразовать градусы в радианы, чтобы задать начальный и конечный углы.
- $360 \text{ градусов} = 2\pi \text{ радианов}$.
- Чтобы нарисовать текст в `canvas`, используйте метод `fillText`.
- При рисовании текста в `canvas` вам будет нужно указывать позицию, стиль и пр., используя свойства объекта `context`.
- Когда вы задаете значение для свойства объекта `context`, оно применяется на протяжении всего процесса рисования, пока вы не присвоите этому свойству другое значение. Например, изменение значения свойства `fillStyle` повлияет на цвет фигур и текста, создаваемых вами после присваивания этому свойству нового значения.
- Добавление изображений в `canvas` осуществляется с помощью метода `drawImage`.
- Чтобы добавить изображение в `canvas`, вам сначала потребуется создать объект `Image` и убедиться, что соответствующее изображение полностью загрузилось.
- Рисование в `canvas` сродни растровому рисованию в программах для работы с графикой.

ВЕБВИЛЛЬСКИЙ КУРЬЕР



Сенсационная новость: у <canvas>
и <video> все-таки много общего!

Вебвилль — здесь ты узнаешь обо всем первым

После эксклюзивного интервью мы можем сообщить, что элементы <canvas> и <video> не просто делят друг с другом одни и те же веб-страницы... Они, например, смешивают свое содержимое.

Трой Армстронг

ШТАТНЫЙ АВТОР «ВЕБВИЛЛЬСКОГО КУРЬЕРА»

<video> говорит: «Это правда, мы наладили тесные взаимоотношения. Видите ли, я довольно простой парень, который умеет воспроизводить видео, причем делает это очень хорошо. Однако это почти все, чем я занимаюсь. Но благодаря <canvas> все изменилось. Я облачаюсь в пользовательские элементы управления, я фильтрую свое видеосодержимое, я одновременно показываю несколько видеозображений».

За комментариями мы обратились к <canvas>. Является ли он тем, кто стоит за тегом <video>? От <canvas> мы услышали следующее: «Что ж, <video> сам по себе очень хорошо справляется, ну там, знаете, с декодированием всех этих видеофайлов, сжатых определенными кодеками, с поддержкой частоты кадров в секунду и тому подобным. Это большая работа, с которой мне никогда не удалось бы справиться. Однако с моей помощью он может избежать своего обычного, осмелюсь даже сказать, «скучного» способа воспроизведения видео. Я даю ему средства для исследования всех креативных возможностей по объединению видео с веб-приложениями».

Что ж, кто бы мог такое предположить? Полагаю, что впереди нас ждут интересные вещи, которые станут результатом партнерства <canvas> и <video>!

Можно ожидать, что последствия этого откровения мы будем наблюдать в главе, посвященной элементу <video>, когда его многообещающее партнерство с <canvas> будет подвергнуто пристальному вниманию общественности.

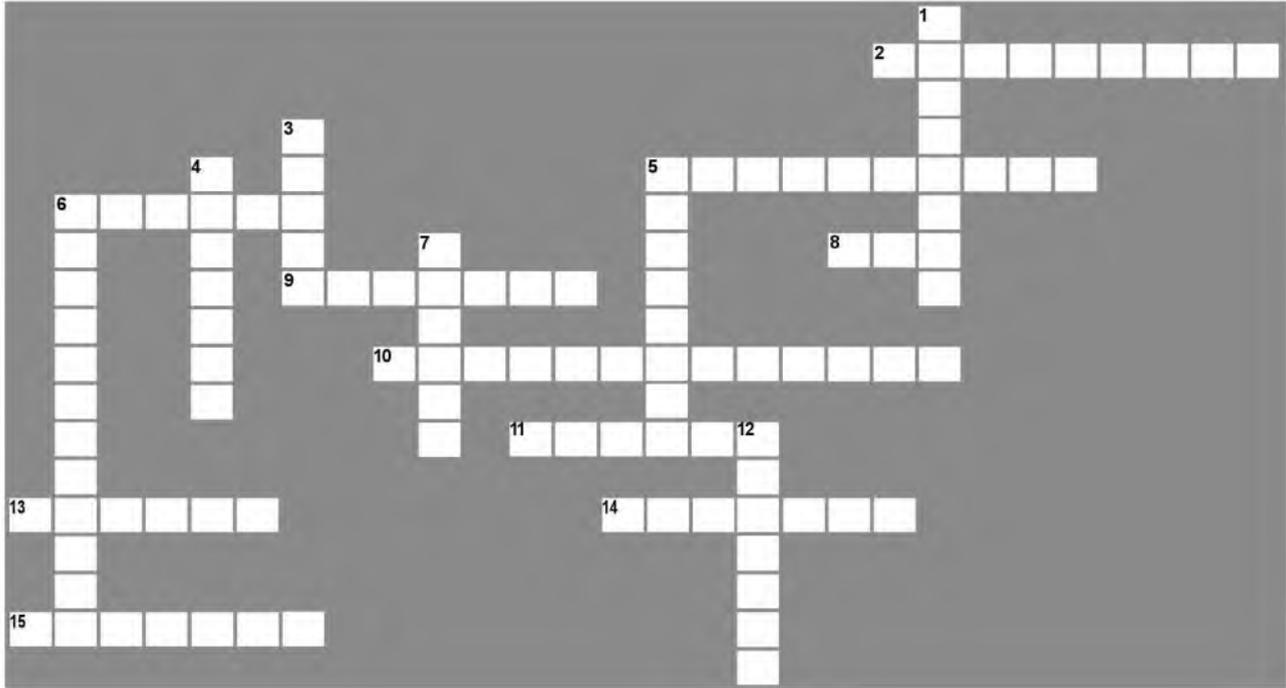


Местная жительница Хайди Масгров была потрясена, когда узнала правду об этих двух элементах.



HTML5-крсслворд

Мы с нетерпением ждем следующей главы, где сможем подробнее поговорить о сепсационной повости, касающейся `<canvas>` и `<video>`. А вы тем временем закрепите свои новые знания о `canvas`, разгадав небольшой кроссворд (возможно, за чашкой чая).



По горизонтали

2. Свойство, для которого мы задавали значение с целью заливки фигуры цветом.
5. Несуществующий метод, который Джим пытался использовать для рисования кругов.
6. Сообщить о завершении загрузки чего-либо можно с помощью обработчика _____.
8. Для рисования кругов следует применять метод _____.
9. Мы осуществляем ее для того, чтобы сделать контур фигуры видимым.
10. Хотите узнать, какой параметр был выбран? Тогда вам может потребоваться данное свойство.
11. Невидимая линия, которую вы чертите с целью нарисовать фигуру.
13. Мы выравнивали текст ...а в результате получил эту паршивую майку! по _____ краю.
14. В круге 360 _____.
15. Все в `canvas` является _____.

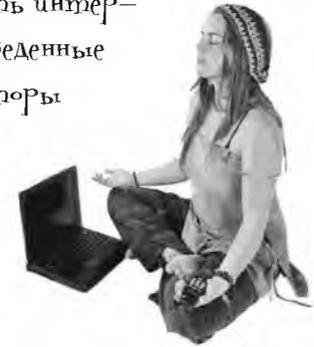
По вертикали

1. Данный метод объекта `context` создает прямоугольник.
3. `<canvas>` и _____ хорошо сочетаются друг с другом.
4. Объект, включающий методы и свойства для рисования в `canvas`.
5. Используйте данный метод для рисования текста в `canvas`.
6. Наилучшее место для размещения хорошего твита.
7. Чтобы переместить карандаш при черчении контура в точку с координатами 100, 100, используйте _____(100, 100).
12. Мы думаем в градусах, а `canvas` — в _____.

СТАНЬ браузером. Решение

Представьте, что вы используете этот интерфейс для выбора значений в случае со своей футболкой.

Теперь, когда у вас есть интерфейс, выполните приведенные здесь JavaScript-операторы и напишите значение для каждого элемента интерфейса.



```
var selectObj = document.getElementById("backgroundColor");
var index = selectObj.selectedIndex;
var bgColor = selectObj[index].value;
```

white

```
var selectObj = document.getElementById("shape");
var index = selectObj.selectedIndex;
var shape = selectObj[index].value;
```

circles

```
var selectObj = document.getElementById("foregroundColor");
var index = selectObj.selectedIndex;
var fgColor = selectObj[index].value;
```

black

Обратите внимание, что в случае с каждым значением меню мы извлекаем элемент select, в котором содержится соответствующий параметр, находим выбранный параметр с помощью свойства selectedIndex и извлекаем значение этого параметра.

Не забывайте, что значение параметра может отличаться от текста, который вы видите в элементах управления; в нашем случае это как раз касается первых букв текста.

Вот значения, которые мы выбрали в интерфейсе TweetShirt для генерирования ответов, приводившихся выше.

При необходимости еще раз взгляните на HTML, чтобы увидеть значения параметров.

Select background color:

Circles or squares?

Select text color:

Pick a tweet:



Развлечения с магнитами. Решение

Воспользуйтесь своими псевдомагическими способностями программиста для размещения в нужной последовательности приведенных ниже табличек. Вам нужно написать псевдокод для функции `drawSquare`. Данная функция принимает `canvas` и `context` и рисует в `canvas` один квадрат произвольного размера. Вот наше решение этого упражнения.

```
function drawSquare ( canvas , context ) {
```

Одну из табличек мы уже разместили за вас в нужном месте.

- вычислить произвольную ширину для квадрата
- вычислить произвольную позицию x для квадрата внутри canvas
- вычислить произвольную позицию y для квадрата внутри canvas
- задать для `fillStyle` значение "lightblue"
- нарисовать квадрат, имеющий позицию x, y и ширину w

Разместите здесь таблички с псевдокодом в нужной последовательности!

```
}
```

Возьми в руку карандаш



Решение

Чтобы при каждом нажатии кнопки Preview (Предварительный просмотр) мы наблюдали в `canvas` только новые квадраты, потребуется залить фон `canvas` цветом, выбранным пользователем в меню "backgroundColor". Сначала реализуем функцию для заливки `canvas` выбранным цветом. Ниже приведен код, в котором вам необходимо устранить пробелы. Вот наше решение этого упражнения.

```
function fillBackgroundColor(canvas, context) {
    var selectObj = document.getElementById("backgroundColor");
    var index = selectObj.selectedIndex;
    var bgColor = selectObj.options[index].value;
    context.fillStyle = bgColor;
    context.fillRect(0, 0, canvas.width, canvas.height);
}
```

Для заливки фона цветом мы рисуем прямоугольник, который заполняет цветом полностью весь `canvas`.

СТАНЬ браузером. Решение



Интерпретируйте приведенный ниже вызов Метода `arc` и напишите все соответствующие значения, которые будут характеризовать круг, а также изобразите дугу, создаваемую в результате этого вызова.

```
context.arc(100, 100, 75, degreesToRadians(270), 0, true);
```

И чертим против часовой стрелки

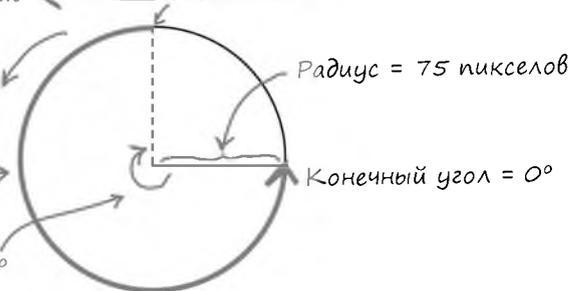
Начинаем здесь

Радиус = 75 пикселей

Дуга

Конечный угол = 0°

Начальный угол = 270°



Итак, у нас есть контур! И что теперь?

Теперь вы будете использовать этот контур для рисования линий и заливки своей фигуры цветом, конечно же! Создайте простую HTML5-страницу с элементом `canvas` и наберите весь приводившийся чуть ранее код. Затем проведите тестирование.

```
context.beginPath();
context.moveTo(100, 150);
context.lineTo(250, 75);
context.lineTo(125, 30);
context.closePath();
```

← Вот код, приводившийся ранее

```
context.lineWidth = 5;
context.stroke();
context.fillStyle = "red";
context.fill();
```

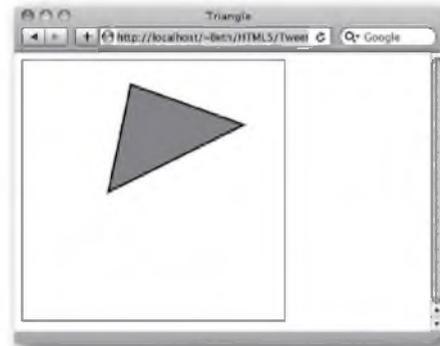
Задать толщину линии для рисования поверх контура.

Нарисовать линию поверх контура.

Задать красный цвет для заливки треугольника.

Применить к треугольнику заливку красным цветом.

Загрузив веб-страницу `Triangle`, мы увидим вот это (для рисования мы создали `canvas` размером `300 × 300` пикселей).





Перерыв. Решение

Пора применить на практике ваши новые знания о дугах и контурах для создания улыбающейся рожицы. Устраните пробелы в приведенном ниже коде, который позволит нарисовать улыбающееся лицо. Мы предусмотрели для вас подсказки относительно того, где на диаграмме должны располагаться глаза, нос и рот.

Вот наше решение этого упражнения:

```
function drawSmileyFace() {
    var canvas = document.getElementById("smiley");
    var context = canvas.getContext("2d");

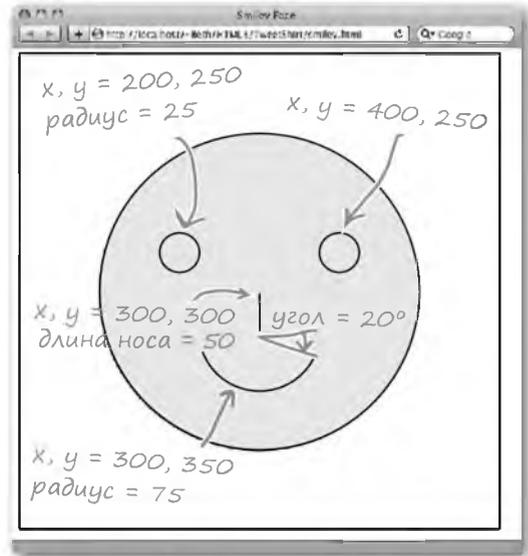
    context.beginPath();
    context.arc(300, 300, 200, 0, degreesToRadians(360), true);
    context.fillStyle = "#ffffcc";
    context.fill();
    context.stroke();

    context.beginPath();
    context.arc(200, 250, 25, 0, degreesToRadians(360), true);
    context.stroke();

    context.beginPath();
    context.arc(400, 250, 25, 0, degreesToRadians(360), true);
    context.stroke();

    context.beginPath();
    context.moveTo(300, 300);
    context.lineTo(300, 350);
    context.stroke();

    context.beginPath();
    context.arc(300, 350, 75, degreesToRadians(20), degreesToRadians(160), false);
    context.stroke();
}
```



Круг лица. Здесь мы уже устранили за вас один из пробелов. Обратите внимание, мы применили к кругу заливку желтым цветом.

Левый глаз. Центр круга имеет координаты $x = 200$, $y = 250$, радиус 25, начальный угол 0, конечный — $\text{Math.PI} * 2$ радиана (360 градусов). Обводим контур, чтобы получить очертания круга (но без заливки).

Правый глаз. Такой же, как и левый, но с координатой $x = 400$. Мы используем направление против часовой стрелки (значение true) (при рисовании полного круга направление неважно).

Это нос. Мы используем `moveTo(300, 300)` для перемещения пера в точку $x = 300$, $y = 300$, чтобы начать создание линии. Затем мы применяем `lineTo(300, 350)`, поскольку нос будет иметь длину 50 пикселей. После этого обводим контур.

Чтобы у нас получилось более правдоподобное улыбающееся лицо, начинаем и заканчиваем чертить край рта на 20° ниже оси X. Это означает, что начальный угол будет равен 20° , а конечный составит 160° .

Направление — против часовой стрелки (значение false), поскольку нам нужен улыбающийся рот (не забывайте, что начальная точка находится справа от центра рта).



Развлечения с Магнитами

Пришло время вашего первого эксперимента с текстом в canvas. Ниже приведен начатый нами код для метода `drawText`, который мы будем вызывать, чтобы нарисовать весь текст в canvas для предварительного просмотра. Посмотрите, сможете ли вы завершить данный код, чтобы нарисовать Я повелся на этот твит... и...а в результате получил эту паршивую майку! в canvas, а рисование реального твита пользователя мы оставим на потом. Вот наше решение этого упражнения.

```
function drawText(canvas, context) {
    var selectObj = document.getElementById(" foregroundColor ");
    var index = selectObj.selectedIndex;
    var fgColor = selectObj[index].value;
    context. fillStyle = fgColor;
    context. font = "bold lem sans-serif";
    context. textAlign = "left";
    context. fillText (" "Я повелся на этот твит..." 20, 40);
```

Подсказка: это позиция x, y для текста «Я повелся на этот твит...».

На данный момент мы размещаем комментарии там, где будет располагаться код для рисования текста твита.

```
// Извлечь выбранный твит из меню выбора твитов
// Нарисовать твит
```

Подсказка: мы будем использовать курсивный шрифт с засечками для твита, однако нам необходимо, чтобы этот текст имел полужирный шрифт Helvetica.

```
context.font = " bold lem sans-serif ";
context. textAlign = " right ";
context. fillText ("...а в результате получил эту паршивую майку!",
    canvas.width-20 , canvas.height-40 );
```

Подсказка: мы хотим поместить текст в правый нижний угол.

Мы хотим нарисовать данный текст в 20 пикселах от правой стороны canvas и в 40 пикселей от основания canvas, чтобы он уравновешивал верхнюю строку текста.

Лишние магнитные таблички

fillCircle

fillRect

left



Упражнение
Решение

Посмотрите, сможете ли вы собрать функцию `drawBird` из всех тех кусочков, которые нам предоставила Джуди. Функция `drawBird` принимает `canvas` и `context` и рисует изображение птицы в `canvas`. Исходите из того, что с помощью данной функции мы должны разместить "twitterBird.png" вместе с координатами $x = 20, y = 120$, при этом ширина и высота будут равны 70 пикселям. Мы уже написали за вас объявление метода и первую строку. Вот наше решение этого упражнения.

Свой код напишите здесь →

```
function drawBird(canvas, context) {
    var twitterBird = new Image();
    twitterBird.src = "twitterBird.png";
    twitterBird.onload = function() {
        context.drawImage(twitterBird, 20, 120, 70, 70);
    };
}
```

Не забудьте добавить вызов `drawBird` в свою функцию `previewHandler!`



HTML5-кроссворд. Решение



Пасхальное яйцо в TweetShirt

Итак, вы сгенерировали отличный предварительный просмотр TweetShirt. А что теперь? Если вы действительно захотите превратить свой дизайн в изображение на футболке, то сможете сделать это! Каким образом? Вот небольшое дополнение, которое вам нужно внести в код, — пасхальное яйцо в TweetShirt. Оно превратит ваш дизайн в изображение, полностью готовое к выгрузке на сайт, позволяющий нанечатать его на настоящей футболке (в Интернете таких сайтов существует целое множество).

Как это сделать? Элементарно! Мы можем воспользоваться методом `toDataURL` объекта `canvas`. Взгляните на пример:



Мы создали новую функцию `makeImage`, чтобы добавить данную возможность.

```
function makeImage() {
    var canvas = document.getElementById("tshirtCanvas");
    canvas.onclick = function () {
        window.location = canvas.toDataURL("image/png");
    };
}
```

Извлекаем объект `canvas`...

...и добавляем обработчик событий, чтобы щелчок на элементе `canvas` приводил к генерированию изображения.

Мы задаем генерируемое изображение в качестве значения `window.location` в случае с браузером, благодаря чему на странице в браузере вы увидите именно это изображение.

Мы просим `canvas` создать изображение в формате PNG из пикселей, нарисованных в `canvas`.

Следует отметить, что PNG является единственным форматом, который должен поддерживаться браузерами, поэтому именно его мы и рекомендуем вам использовать.

Теперь просто добавьте вызов `makeImage` в функцию `window.onload`, и ваш `canvas` отныне сможет генерировать изображения, когда вы будете щелкать на нем. Проведите тестирование. И дайте нам знать, если создадите футболку с изображением!

```
window.onload = function() {
    var button = document.getElementById("previewButton");
    button.onclick = previewHandler;
    makeImage();
}
```

Обеспечьте вызов `makeImage` для добавления обработчика событий `onclick` для `canvas`, и ваше пасхальное яйцо будет готово.



Некоторые браузеры не позволят извлекать изображение из `canvas`, если вы выполняете данный код из `file://`.

Выполняйте данный код, используя `localhost://` или онлайн-сервер, если хотите, чтобы он работал во всех браузерах..



Элемент *video*...

...и наш особый гость — элемент *canvas*



Нам больше не нужны плагины. Элемент `video` отныне является полноценным членом HTML-семейства: просто вставьте его в свою страницу — и вы обеспечите прямую поддержку воспроизведения видео на большинстве устройств. Однако `video` — нечто *значительно большее*, чем просто элемент: это API-интерфейс JavaScript, позволяющий управлять воспроизведением, создавать пользовательские видеоинтерфейсы и интегрировать видео с остальными HTML-элементами совершенно новыми способами. Говоря об *интеграции*... как уже отмечалось ранее, между `video` и `canvas` *существует связь*, — вы увидите, что объединение этих элементов открывает новые возможности по *обработке видео* в режиме реального времени. В этой главе мы научимся внедрять элемент `video` в веб-страницу, а затем поговорим об использовании соответствующего API-интерфейса JavaScript. Вы будете поражены тем, что можно сделать с помощью небольшого количества разметки, JavaScript и элементов `video` и `canvas`.

Знакомство с Webville TV

Webville TV — это содержимое, которого вы ждали, например: «Пункт назначения — планета Земля» (Destination Earth), «Нападение 50-футовой женщины» (The Attack of the 50' Woman), «Нечто» (The Thing), «Капля» (The Blob), и не будет лишним включить сюда образовательные фильмы 1950-х годов. Однако это всего лишь содержимое, а что касается технологий, то ожидаете ли вы здесь чего-то меньшего, чем HTML5-видео?

Это, конечно же, лишь наше видение проблемы, и если мы хотим трансформировать его в нечто реальное, то нам потребуется создать Webville TV. На нескольких следующих страницах мы будем заниматься созданием Webville TV «с нуля», используя HTML5-разметку, элемент video и немного JavaScript.

*Webville TV, на 100 %
созданное с применением
технологии HTML5.*



*Скоро в вашем
браузере!*



Разберемся с HTML-разметкой...

Это ведь уже глава 8, поэтому не будем медлить! Давайте сразу приступим к написанию HTML-разметки:

```

<!doctype html>
<html lang="en">
<head>
  <title>Webville TV</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="webvilletv.css">
</head>
<body>
<div id="tv">
  <div id="tvConsole">
    <div id="highlight">
      
    </div>
    <div id="videoDiv">
      <video controls autoplay src="video/preroll.mp4" width="480" height="360"
        poster="images/prerollposter.jpg" id="video">
    </video>
    </div>
  </div>
</div>
</body>
</html>

```

↙ Вполне стандартный HTML5.

↙ Не забудьте CSS-файл для обеспечения красивого внешнего вида.

↙ Небольшое изображение, чтобы все было похоже на настоящий телевизор.

↙ А вот наш элемент <video> для воспроизведения видео. Более пристально на него мы взглянем через несколько мгновений...

→ Если у вас возникли проблемы, проверьте страницу!

Включите этот «телевизор» и протестируйте его...



Здесь нужно позаботиться о некоторых вещах: во-первых, наберите весь приведенный чуть выше код и сохраните его в файле webvilletv.html; во-вторых, загрузите на свой компьютер соответствующий CSS-файл, скачайте видеофайлы и разместите их в каталог video. Сделав это, загрузите страницу, после чего откиньтесь на спинку стула и смотрите.

↙ Все необходимое загрузите, посетив страницу <http://wickedlysmart.com/hfhtml5>

↘ Вот что мы видим. Если вы наведете указатель мыши на экран, то появится набор элементов управления, предназначенных для того, чтобы воспроизводить видео, ставить на паузу, регулировать звук, осуществлять поиск в видео.



Я не вижу никакого видео. Я трижды проверил код и расположил видео-файлы в нужном каталоге. Есть идеи?

Да, возможно, дело в формате видео.

Несмотря на то что разработчики браузеров согласны с тем, что элемент `<video>` и API-интерфейс Video схожи в HTML5, не все из них сходятся во мнении насчет *фактического формата* самих видео-файлов. Например, если вы используете браузер Safari, то предпочтительным для вас будет формат кодирования H.264, а если Chrome — то WebM и т. д.

В коде, написанием которого мы только что занимались, в качестве формата кодирования предполагался H.264, работающий в Safari, Mobile Safari, Internet Explorer версии 9 и выше. Если вы используете другой браузер, загляните в свой каталог `video`, где найдете видеофайлы трех отличающихся типов с тремя разными расширениями: MP4, OGV и WEBM (что они означают, рассмотрим позже).

В случае Safari вы уже должны использовать MP4 (который содержит H.264).

При использовании Google Chrome следует применять формат WEBM, для чего нужно заменить значение атрибута `src` следующим:

```
src="video/preroll.webm"
```

Если же вы используете Firefox или Opera, то замените значение атрибута `src` на такое:

```
src="video/preroll.ogv"
```

А если вы работаете с Internet Explorer версии 8 или ниже, вам не повезло, — ностойте-ка, это же глава 8! Как вы до сих пор можете пользоваться браузером Internet Explorer 8 (или ниже)? Обновите его! Но если вам необходимо узнать, как обеспечить резервное содержимое для пользователей, у которых установлен Internet Explorer 8, потерпите, поскольку мы еще дойдем до этого.

На момент чтения вами этой книги данные форматы могут уже более широко поддерживаться всеми браузерами. Таким образом, если видео у вас воспроизводится, все отлично. Постоянно заглядывайте в Интернет в поисках самой свежей информации по этой развернутой теме. Мы к ней вскоре вернемся.



Попробуйте сделать так для начала, а чуть позже мы вернемся к этому.

Как работает элемент video?

Итак, вы все настроили, и у вас на странице воспроизводится видео. Однако прежде, чем мы двинемся дальше, давайте взглянем на элемент video, который использован в нашей разметке:

```
<video controls
  autoplay
  src="video/preroll.mp4"
  width="480" height="360"
  poster="images/prerollposter.jpg"
  id="video">
</video>
```

Если атрибут controls присутствует, проигрыватель предоставит пользователю элементы управления для контроля над воспроизведением видео и аудио.

Благодаря атрибуту autoplay воспроизведение видео будет запускаться по завершении загрузки страницы.

Исходное местоположение видео

Ширина и высота видео на странице

Картинка, которая отображается, когда видео не воспроизводится.

Идентификатор для элемента video, чтобы мы могли обращаться к нему позже из JavaScript.

Еще один полезный совет из HTML5-руководства города Вебвилль.



Правила видеозэтикета: свойство autoplay

Несмотря на то что autoplay может оказаться лучшим выбором в случае с такими сайтами, как YouTube и Vimeo (или даже Webville TV), дважды подумайте, прежде чем задавать его в своем теге <video>. Пользователь зачастую желает сам решать, должно ли воспроизводиться видео при загрузке вашей страницы.

Пристальный взгляд на атрибуты элемента video...



Давайте внимательнее взглянем на наиболее важные атрибуты элемента video:

controls
Атрибут controls является логическим. Либо он есть, либо его нет. Если он присутствует, то браузер добавит свои встроенные элементы управления в область отображения видео. Здесь возможны вариации в зависимости от браузера, поэтому проверяйте, как они будут выглядеть в каждом браузере. Вот как они выглядят в Safari.

src определяет, какой видео-файл будет здесь воспроизводиться.

src
Атрибут src подобен src элемента — это URL-адрес, который сообщает элементу video, где искать файл-источник. В данном случае таким файлом является video/preroll.mp4 (если вы загрузили на свой компьютер код, используемый в этой главе, то сможете найти данный видеофайл, а также два других в каталоге video).

preload
Логический атрибут preload обычно используется для тщательного контроля над загрузкой видео в целях оптимизации. Браузер, по большей части, решает, какой объем видео загружать, исходя, например, из того, был ли задан атрибут autoplay, а также ориентируясь на пропускную способность канала пользователя. Вы можете изменить данное поведение, присвоив preload значение none (видео не будет загружаться до тех пор, пока пользователь не нажмет кнопку воспроизведения), либо metadata (метаданные видео будут загружаться, но без видеосодержимого), либо auto, которое позволит браузеру самостоятельно принимать решение.

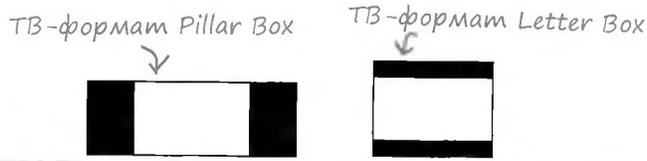
autoplay
Логический атрибут autoplay дает браузеру команду начинать воспроизведение видео, как только у него будет достаточно данных. В случае с нашими демонстрационными видеофайлами вы, вероятно, увидите, что их воспроизведение запускается почти сразу.

Видеопроигрыватель

poster
Браузер обычно отображает один кадр видео в качестве постерного изображения для представления этого видео. Если вы удалите атрибут autoplay, то будете наблюдать данное изображение, пока не нажмете кнопку воспроизведения. Браузер сам решает, какой кадр показывать; зачастую он просто показывает первый кадр видео... который нередко оказывается полностью черным. Если вы хотите, чтобы в данном случае выводилось определенное изображение, вам потребуется создать его, а затем задать, используя атрибут poster.

width, height
Атрибуты width и height определяют ширину и высоту области отображения видео (также известной как область просмотра). Если вы зададите значение для poster, то постерное изображение будет подвергаться масштабированию в соответствии с указанными вами шириной и высотой. Видео также будет масштабироваться, но сохранит соотношение ширины и высоты (например, 4:3 или 16:9), поэтому при наличии дополнительного пространства по бокам либо сверху и снизу видео станет выводиться в ТВ-формате Letter Box либо Pillar Box с целью подгонки под размеры области отображения видео. Вы должны стараться обеспечивать соответствие родным размерам видео, если требуется наилучшая производительность (браузеру не придется заниматься масштабированием в режиме реального времени).

loop
loop, который представляет собой еще один логический атрибут, обеспечивает автоматический повтор воспроизведения видео после того, как его проигрывание завершается.



Элементы управления в каждом браузере выглядят по-разному; хотя при использовании решений вроде Flash в их внешнем виде, по крайней мере, наблюдалась согласованность.



Да, элементы управления отображением HTML-видео в каждом из браузеров выглядят по-разному.

Внешний вид ваших элементов управления определяют те, кто реализует браузеры. Элементы управления зачастую выглядят неодинаково в разных браузерах и операционных системах. В некоторых случаях, например на планшетном компьютере, они должны выглядеть и вести себя по-другому, поскольку само устройство работает иначе (и хорошо, что об этом за вас уже позаботились). Вместе с тем мы понимаем, что в случае, скажем, настольных браузеров было бы неплохо иметь согласованно выглядящие элементы управления, однако все это не является формальной частью спецификации HTML5. В некоторых ситуациях подход, работающий в одной операционной системе, может расходиться с основными принципами интерфейса пользователя другой ОС. Просто знайте, что элементы управления могут различаться, а если вами будет двигаться глубокая мотивация, то можете реализовать пользовательские элементы управления для своих приложений.

↶ Этим мы займемся позже...

Что необходимо знать о форматах видео

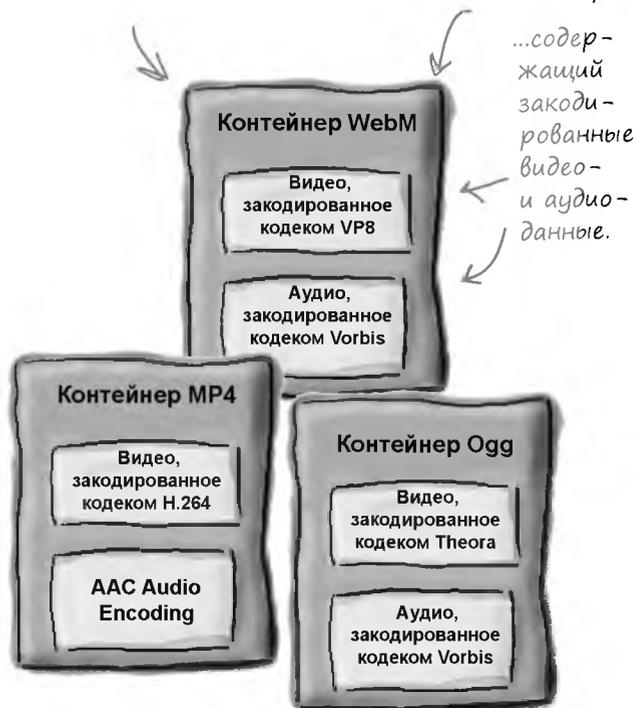
Нам бы хотелось, чтобы все было четко и ясно, как в случае с элементом `video` и его атрибутами, но, как оказалось, с форматами видео в Интернете небольшой беспорядок. Что же такое формат видео? Можно рассмотреть его следующим образом: видеофайл содержит две части — видео и аудио, при этом каждая из них закодирована (чтобы уменьшить размер и обеспечить возможность более эффективного воспроизведения файла) с использованием определенного кодека. Кодек в большинстве случаев может оказываться тем, с чем будут согласны далеко не все, — одни разработчики браузеров отдают предпочтение кодеку H.264, вторым очень нравится VP8, а третьи любят альтернативные решения с открытым исходным кодом, например Theora. Все это *еще больше* усложняется тем, что файл, содержащий закодированные с помощью определенного кодека видео- и аудиоданные (он называется *контейнером*), имеет собственный формат и имя. Таким образом, у нас получается настоящая мешанина из технических терминов.

У нас мог бы быть большой и счастливый мир, если бы все разработчики браузеров договорились между собой о едином формате для использования в Интернете, но, увы, этого не случится, и на то есть ряд технических, политических и философских причин. Однако вместо того чтобы открывать здесь дискуссию, мы просто постараемся дать вам достаточный объем знаний по этой теме, чтобы вы смогли принимать собственные решения о том, как обеспечить поддержку для своей целевой аудитории.

Давайте взглянем на наиболее популярные кодеки. В настоящее время существуют три соперника, пытающихся нравиться миром (Интернета)...

Три разных формата видео, используемых в основных браузерах.

Это контейнер...



У вас может быть другое мнение на этот счет, поскольку предпочитаемые форматы имеют тенденцию меняться со временем.

Каждый формат включает тип контейнера (например, WebM, MP4, Ogg) и используемых видео- и аудиокодеков (например, VP8, Vorbis).

Спецификация HTML5 допускает применение любого формата видео. Все зависит от реализации браузера, которая и определяет, какие форматы действительно поддерживаются.

Претенденты

Если вы собираетесь ноставлять содержимое для широкого круга нользователей, то вам придется предусмотреть наличие видеофайлов более чем одного формата. С другой стороны, если вы нацеливаетесь, например, только на нользователей Apple iPad, возможно, вам удастся обойтись лишь одним форматом видео. В настоящее время существуют три основных претендента.

Контейнер MP4 с видео H.264 и аудио AAC

H.264 лицензирован группой MPEG-LA.

Существует более одного типа H.264, каждый из которых известен как профиль.

MP4/H.264 поддерживается браузером Safari и Internet Explorer версии 9 и выше. Также имеется поддержка со стороны некоторых версий браузера Chrome.

Контейнер WebM с видео VP8 и аудио Vorbis

WebM был создан компанией Google для использования в сочетании с видео, закодированным с помощью VP8.

WebM/VP8 поддерживается браузерами Firefox, Chrome и Opera.

Видеофайлы в контейнерном формате WebM имеют расширение WEBM.

Контейнер Ogg с видео Theora и аудио Vorbis

Theora — это кодек с открытым исходным кодом.

Видео, закодированное с его помощью, обычно содержится в Ogg-файле, который имеет расширение OGV.

Ogg/Theora поддерживается браузерами Firefox, Chrome и Opera.

H.264 — любимец индустрии, но не правящий чемпион...

Theora. Альтернативное решение с открытым исходным кодом.

VP8 — претендент, за спиной которого стоит Google, при этом поддерживается браузерами других разработчиков и набирает силу...



Как жонглировать всеми этими форматами...

Итак, в мире форматов видео присутствует некоторый беспорядок, но что нам делать? В зависимости от вашей целевой аудитории вы можете решить обеспечить наличие видео только в одном формате либо в нескольких. В любом случае вы сможете использовать один элемент `<source>` (не путать с *атрибутом src*) на каждый формат внутри элемента `<video>`, чтобы обеспечить набор видеофайлов, каждый из которых будет иметь свой собственный формат, и дать возможность браузеру выбирать видеофайл именно в том формате, который он поддерживает. Например:

Обратите внимание, что мы удаляем атрибут src из тега <video>...

...и добавляем три тега <source>, каждый из которых имеет собственный атрибут src, указывающий путь к видеофайлу в отличающемся формате.

```
<video poster="video/prerollposter.jpg" controls
width="480" height="360">
  <source src="video/preroll.mp4">
  <source src="video/preroll.webm">
  <source src="video/preroll.ogv">
  <p>Sorry, your browser doesn't support the video element</p>
</video>
```

Сообщение, которое выведет браузер, если он не поддерживает элемент video.

Браузер начинает сверху и двигается вниз, пока не отыщет файл в формате, который он сможет воспроизвести.

В случае с каждым <source> браузер загружает метаданные видеофайла, чтобы проверить, сможет ли он его воспроизвести (этот процесс бывает длительным, однако его можно облегчить для браузера... см. следующую страницу).

КЛЮЧЕВЫЕ МОМЕНТЫ



- **Контейнер** — это формат файлов, используемый для «упаковки» видео-, аудио- и метаданных. Среди распространенных контейнерных форматов можно назвать MP4, WebM, Ogg и Flash Video.
- **Кодек** — это программный инструмент, используемый для кодирования и декодирования видео и аудио в определенном формате. К числу популярных веб-кодеков относятся H.264, VP8, Theora, AAC и Vorbis.
- Браузер сам решает, какое видео он сможет декодировать. При этом среди разработчиков браузеров нет согласия относительно единого формата видео, поэтому если вы хотите обеспечить поддержку для каждого пользователя, вам придется позаботиться о наличии видеофайлов в нескольких форматах.



Как обеспечить еще большую конкретизацию для своих форматов видео

Сообщив браузеру местонахождение своих файлов-источников, вы даете ему набор различных вариантов, из которых он сможет выбрать подходящий. Однако веб-обозревателю придется провести кое-какое «расследование», прежде чем у него получится точно определить, сможет ли он воспроизвести соответствующий файл. Вы можете помочь своему браузеру, сообщив ему дополнительную информацию о типе MIME и (опционально) о кодеках ваших видеофайлов:

Файл, путь к которому вы указываете в `src`, на самом деле является контейнером, содержащим фактическое видео (и аудио, а также метаданные).

Параметр `codecs` определяет, какие кодеки были использованы для кодирования видео и аудио с целью создания закодированного видеофайла

Видеокодек

Аудиокодек

```
<source src="video/preroll.ogv" type='video/ogg; codecs="theora, vorbis"'>
```

`type` является опциональным атрибутом, который выступает в роли подсказки для браузера, помогающей ему понять, сможет ли он воспроизвести данный тип файла.

Это тип MIME видеофайла. Он определяет контейнерный формат.

Обратите внимание на двойные кавычки вокруг значения параметра `codecs`. Они подразумевают, что значение атрибута `type` нам следует заключить в одинарные кавычки.

Мы можем обновить наши элементы `<source>` с целью включения в них информации о типе для каждого из трех типов видеофайлов, которые у нас имеются, как показано далее:

```
<video id="video" poster="video/prerollposter.jpg" controls width="480" height="360">
  <source src="video/preroll.mp4" type='video/mp4; codecs="avc1.42E01E, mp4a.40.2"'>
  <source src="video/preroll.webm" type='video/webm; codecs="vp8, vorbis"'>
  <source src="video/preroll.ogv" type='video/ogg; codecs="theora, vorbis"'>
  <p>Sorry, your browser doesn't support the video element</p>
</video>
```

Если вы не будете знать значений параметров `codecs`, можете отбросить их и указать только тип MIME. Это обеспечит чуть меньшую эффективность, однако, по большей части, все будет в порядке.

Значения параметра `codecs` в случае с MP4 будут более замысловатыми, чем в случае с двумя другими контейнерными форматами, поскольку кодек H.264 поддерживает разнообразные профили с настройками кодирования видеофайлов для разных пользователей (например, у одних из них может иметься канал с высокой пропускной способностью, а у других — с низкой). Таким образом, чтобы задать здесь правильные значения, вам необходимо знать подробности о том, как видео было закодировано.

Когда вы будете разбираться с кодированием видеофайлов, вам придется узнать подробности о разнообразных вариантах параметров `type` для использования в вашем элементе `<source>`. Дополнительную информацию о данных параметрах вы сможете найти по адресу http://wiki.whatwg.org/wiki/Video_type_parameters.

Часть
Задаваемые
Вопросы

В: Есть ли хоть какая-нибудь надежда, что в ближайшие несколько лет мы придем к единому контейнерному формату или типу кодека? Разве на это не указывает наличие у нас стандартов?

О: Вряд ли в ближайшее время появится формат, который станет господствующим над всеми остальными. Как мы уже отмечали, данная тема пересекается с целой массой сложностей, начиная с компаний, желающих контролировать свою судьбу в сфере видео, и заканчивая комплексными проблемами с интеллектуальной собственностью. Комитет по стандартам HTML5 осознал это и решил не определять формат видео в спецификации HTML5. Таким образом, несмотря на то что HTML5, в принципе, поддерживает (или, по крайней мере, агностически воспринимает) все эти форматы, в действительности разработчикам браузеров решать, что они будут, а что не будут поддерживать.

Следите за этой темой, если видео имеет для вас большое значение; за ней, несомненно, будет интересно наблюдать ближайшие несколько лет, пока специалисты будут со всем этим разбираться. И, как всегда, принимайте во внимание потребности своей целевой аудитории и обязательно делайте все возможное, чтобы обеспечить для них соответствующую поддержку.

В: С чего мне начать, если я захочу заняться кодированием собственного видео?

О: Существует масса программ для захвата и кодирования видео. А уж какую из них вам выбрать, зависит от типа видео, захват которого вы хотите производить, и того, как вы собираетесь использовать конечный результат. На тему кодирования видео написаны книги, так что будьте готовы войти в мир новых акронимов и технологий. Вы можете начать с несложных программ вроде iMovie или Adobe Premiere Elements, которые позволяют кодировать

видео для последующего размещения в Интернете. Если вы собираетесь заняться серьезной обработкой видео, используйте программы Final Cut Pro или Adobe Premiere, в которых имеются встроенные производственные инструменты. Наконец, если вы хотите осуществлять доставку своего видео посредством сети Content Delivery Network (CDN), то знайте, что многие CDN-компании также предлагают услуги по кодированию. Таким образом, у вас имеется широкий выбор разнообразных вариантов.

В: Могу ли я воспроизводить свое видео в полноэкранном режиме? Удивительно, что в соответствующем API-интерфейсе нет свойства для этого.

О: Данная функциональность еще не была стандартизирована, однако в Интернете можно найти способы воспроизводить полноэкранное видео с помощью некоторых браузеров. Часть браузеров предусматривают наличие элемента управления для проигрывания видео во весь экран (например, на планшетных компьютерах), который наделяет данной возможностью элемент `video`. Однако после того, как вы найдете способ воспроизводить видео в полноэкранном режиме, список того, что с ним можно будет сделать помимо простого проигрывания, может оказаться ограниченным по соображениям безопасности (как это бывает с видеоплагинами).

В: А как насчет звука в моем видео? Можно ли использовать API-интерфейс для управления уровнем громкости?

О: Конечно можно. Вы можете присваивать свойству `volume` значение с плавающей точкой в промежутке от 0.0 (звук отключен) до 1.0 (максимальная громкость звука). Просто используйте объект `video` для задания значения для этого свойства в любое время:

```
video.volume = 0.9;
```

ДЕЛО: HTML5

**ВАША СЛЕДУЮЩАЯ МИССИЯ:
РАЗВЕДКА ПОДДЕРЖКИ ВИДЕО**

**СОВЕРШЕННО
СЕКРЕТНО**

ВАМ ПОРУЧАЕТСЯ ОПРЕДЕЛИТЬ [REDACTED] ТЕКУЩИЙ УРОВЕНЬ ПОДДЕРЖКИ ВИДЕО В КАЖДОМ ИЗ ПРИВЕДЕННЫХ НИЖЕ БРАУЗЕРОВ (ПРИМЕЧАНИЕ: НУЖНУЮ ИНФОРМАЦИЮ ВЫ СМОЖЕТЕ ОТЫСКАТЬ ПО АДРЕСАМ [HTTP://EN.WIKIPEDIA.ORG/WIKI/HTML5_VIDEO](http://en.wikipedia.org/wiki/HTML5_VIDEO), [REDACTED] [HTTP://CANIUSE.COM/#SEARCH=VIDEO](http://caniuse.com/#search=video)). ПРИНИМАЙТЕ ВО ВНИМАНИЕ НОВЕЙШИЕ ВЕРСИИ БРАУЗЕРОВ. ПО КАЖДОМУ БРАУЗЕРУ, ПРИВЕДЕННОМУ В СТОЛБЦЕ «ВИДЕО/БРАУЗЕР», ОТМЕЬТЕ ВИДЕОПАРАМЕТРЫ, КОТОРЫЕ ОН ПОДДЕРЖИВАЕТ. ПО ВОЗВРАЩЕНИИ ПРЕДСТАВЬТЕ ОТЧЕТ ДЛЯ ПОЛУЧЕНИЯ НОВОГО ЗАДАНИЯ!

Устройства под управлением операционных систем iOS и Android (среди прочих)

Видео \ Браузер	Safari	Chrome	Firefox	Mobile Webkit	Opera	Internet Explorer 9 и выше	Internet Explorer 8	Internet Explorer 7 или ниже
H.264								
WebM								
Ogg Theora								

Мне кажется, что формат Flash Video по-прежнему актуален, и я хочу позаботиться о резервном варианте на случай, если браузеры моих пользователей не будут поддерживать HTML5-видео.



Нет проблем.

Существует ряд методик для переключения на другой проигрыватель видео, если тот, который вы в настоящее время читаете (будь то проигрыватель видео HTML5, или Flash, или другой), не поддерживается.

Ниже вы найдете пример того, как можно вставить свое Flash-видео в качестве резервного содержимого, заменяющего HTML5-видео в ситуациях, когда браузер не знает, как воспроизвести HTML5-видео. Ясно, что данная область быстро меняется, поэтому заглядывайте в Интернет (в котором размещается более свежая информация, чем приводимая в книгах), чтобы использовать новейшие и наилучшие методики. Вы также сможете найти способы сделать резервным не Flash-видео, а HTML5-видео, если предпочтете отдавать приоритет Flash-видео.

```
<video poster="video.jpg" controls>  
  <source src="video.mp4">  
  <source src="video.webm">  
  <source src="video.ogv">  
  <object>...</object>  
</video>
```

↑ Вставьте свой элемент `<object>` в элемент `<video>`. Если браузер не узнает элемент `<video>`, то будет использоваться элемент `<object>`.

Я слышал, там будут API-интерфейсы?

Как вы видите, используя разметку и элемент `<video>`, можно много чего сделать. Однако элемент `<video>` также предоставляет нам богатый API-интерфейс, который можно использовать для реализации всевозможных интересных нововведений и взаимодействий, связанных с видео. Вот краткая сводка некоторых методов, свойств и событий элемента `<video>`, которые могут вас заинтересовать (их исчерпывающий перечень вы найдете в спецификации):



Вызывайте эти методы

- `play` ← воспроизводит ваше видео
- `pause` ← ставит ваше видео на паузу
- `load` ← загружает ваше видео
- `canPlayType` ← помогает программно определить, какие типы видео вы сможете воспроизвести



Используйте эти свойства

<code>videoWidth</code>	<code>loop</code>
<code>videoHeight</code>	<code>muted</code>
<code>currentTime</code>	<code>paused</code>
<code>duration</code>	<code>readyState</code>
<code>ended</code>	<code>seeking</code>
<code>error</code>	<code>volume</code>

Это свойства объекта элемента `<video>`. Одним из них вы можете присваивать значения (например, `loop` и `muted`), другие же предназначены только для чтения (например, `currentTime` и `error`).



Перехватывайте эти события

<code>play</code>	<code>abort</code>
<code>pause</code>	<code>waiting</code>
<code>progress</code>	<code>loadeddata</code>
<code>error</code>	<code>loadedmetadata</code>
<code>timeupdate</code>	<code>volumechange</code>
<code>ended</code>	

Это события, которые вы сможете обрабатывать, если захотите, добавив обработчики событий, вызов которых будет происходить при наступлении события, прослушивание которого вы ведете.

Немного «программирования» содержимого Webville TV

До настоящего момента у нас на Webville TV демонстрировалось только одно видео. Нам бы хотелось запрограммировать расписание, на котором будет воспроизводиться плейлист видеофайлов. Допустим, в случае с Webville TV нам требуется сделать следующее:



- 1 Показать аудитории небольшое предварительное шоу, ну там, знаете, рекламу «кока-колы» и попкорна, правила поведения для зрителей и т. д.



- 2 Показать наш первый ролик под названием «Популярны ли вы?» (Are you Popular?). Поверьте, он вам понравится.



- 2 А затем показать гвоздь нашей программы — полноцветный фильм «Пункт назначения — планета Земля» (Destination Earth). Созданный Американским институтом нефти, какой же посыл он в себе несет? Посмотрите — и узнаете.

Возьми в руку карандаш



У вас может возникнуть соблазн взглянуть на спецификации разметки `<video>`, чтобы узнать, как определить плейлист. Что ж, для этого вам потребуется код, поскольку элемент `<video>` позволяет определять только одно видео. Если бы вы находились на необитаемом острове и вам требовалось реализовать плейлист исключительно с использованием браузера, элемента `<video>`, свойства `src`, методов `load` и `play`, а также свойства `ended`, то как бы вы сделали это (у вас есть возможность использовать любые типы данных JavaScript, какие пожелаете)?

Подсказка: событие `ended` наступает, когда видео достигает конца и его воспроизведение завершается. Как и в случае с любым другим событием, у вас может иметься обработчик, вызов которого будет происходить при наступлении данного события.

Не подсматривайте решение этого упражнения!

Возьми в руку карандаш



Решение

При загрузке страницы мы генерируем массив `playlist`, запускаем воспроизведение первого видео и задаем обработчик событий, иницируемых, когда воспроизведение видео завершается.

У вас может возникнуть соблазн взглянуть на спецификации разметки `<video>`, чтобы узнать, как определить плейлист. Что ж, для этого вам потребуется код, поскольку элемент `<video>` позволяет определять только одно видео. Если бы вы находились на необитаемом острове и вам требовалось реализовать плейлист исключительно с использованием браузера, элемента `<video>`, свойства `src`, методов `load` и `play`, а также свойства `ended`, то как бы вы сделали это (у вас есть возможность использовать любые типы данных JavaScript, какие пожелаете)? Вот наше решение этого задания:

Псевдокод плейлиста

Создать массив `playlist` с видео

Извлечь `video` из объектной модели документа (DOM).

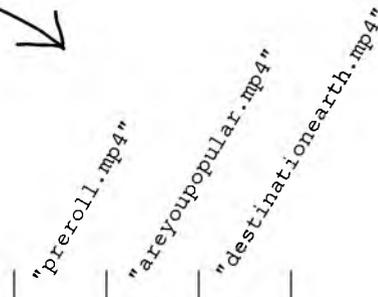
Задать обработчик событий в отношении `video` для обработки события "ended".

Создать переменную `position = 0`.

Задать в качестве источника видео позицию 0 в `playlist`.

Воспроизвести видео.

Мы сохраним плейлист в виде массива. Каждый элемент в нем будет представлять собой видео для воспроизведения.



Массив `playlist`



Каждый раз, когда воспроизведение видео завершается, наступает событие `ended`...

Вот наш обработчик, который вступит в дело по завершении воспроизведения видео.

Псевдокод обработчика событий `ended`

...что приводит к вызову обработчика событий `ended`.

Увеличить значение `position` на 1.

Задать в качестве источника видео следующую позицию в `playlist`

Воспроизвести следующее видео

Дойдя до конца плейлиста, мы можем либо остановиться, либо снова вернуться к первому видео.

Реализация плейлиста Webville TV

Теперь мы воспользуемся JavaScript и API-интерфейсом Video для реализации плейлиста Webville TV. Начнем с добавления ссылки на JavaScript-файл в `webvilletv.html`; просто добавьте данную строку в элемент `<head>`:

```
<script src="webvilletv.js"></script>
```

И удалите эту строку из имеющегося у вас элемента `<video>`:

```
<video controls autoplay src="video/preroll.mp4" width="480" height="360"
  poster="images/prerollposter.jpg" id="video">
</video>
```

← Удаляем атрибуты `autoplay` и `src` из тега `<video>`.

Также удалите все элементы `<source>`, с которыми вы, возможно, экспериментировали.

А теперь создайте новый файл с именем `webvilletv.js`. Кроме того, давайте определим ряд глобальных переменных и функцию, которая будет вызываться после полной загрузки страницы:

```
var position = 0;
var playlist;
var video;
```

← Сначала определим переменную, чтобы следить за тем, какое видео мы воспроизводим; мы присвоим ей имя `position`.

← Еще нам потребуется переменная для размещения массива `playlist` с видео.

← А также переменная для размещения ссылки на элемент `video`.

```
window.onload = function() {
  playlist = ["video/preroll.mp4",
             "video/areyoupopular.mp4",
             "video/destinationearth.mp4"];
  video = document.getElementById("video");
  video.addEventListener("ended", nextVideo, false);
  video.src = playlist[position];
  video.load();
  video.play();
}
```

← Мы создадим массив `playlist` с тремя видео.

← Извлекаем элемент `video`.

← И добавляем обработчик событий `ended` для `video`. Да, это выглядит иначе, чем мы привыкли (потерпите немного — мы поговорим об этом на следующей странице).

← Теперь мы задаем для `src` значение в виде первого видео.

← И загружаем это видео, после чего воспроизводим его!

Так что там с кодом того обработчика событий?

Раньше мы всегда просто присваивали функцию обработчика, вызываемую при наступлении соответствующего события, свойству (например, `onload` или `onclick`), как показано далее:

```
video.onended = nextVideo;
```

Однако на этот раз мы собираемся поступить немного по-другому. Почему? А потому, что на момент написания книги поддержка всевозможных свойств событий в объекте `video` была немного неравномерной. Этот недостаток даст нам возможность продемонстрировать еще один способ регистрации, касающийся событий: `addEventListener`, который является общим методом, поддерживаемым многими объектами для регистрации в случае с различными событиями. Вот как он работает:

Вы можете использовать метод `addEventListener` для добавления обработчика событий. *Функция, которую мы будем вызывать при наступлении соответствующего события*

```
video.addEventListener("ended", nextVideo, false);
```

Объект, на котором мы прослушиваем определенное событие

Событие, прослушивание которого мы ведем. Обратите внимание на то, что мы не ставим `on` перед именем события, как в случае с обработчиками, при задании которых используем свойства (например, `onload`).

Третий параметр контролирует продвинутые методы получения событий, если задано значение `true`. Всегда задавайте здесь значение `false` (если только вы не пишете продвинутый код).

Помимо того факта, что применение метода `addEventListener` является немного более замысловатым, чем простое добавление обработчика путем присваивания функции свойству, он работает во многом так же. Итак, вернемся к нашему коду!

Как написать обработчик «конца видео»

Нам осталось лишь написать обработчик для события `ended`, наступающего при завершении воспроизведения видео. Вызов данного обработчика будет происходить каждый раз, когда видеопроигрыватель будет достигать конца текущего видеофайла. Вот как мы напишем функцию `nextVideo` (добавьте ее в файл `webvilletv.js`):

```
function nextVideo() {
    position++;
    if (position >= playlist.length) {
        position = 0;
    }
    video.src = playlist[position];
    video.load();
    video.play();
}
```

Сначала увеличиваем позицию в массиве `playlist`.

По достижении конца плейлиста мы просто возвращаемся в его начало, для чего задаем для `position` значение 0.

Теперь мы задаем в качестве источника следующее видео, которое будет воспроизводить проигрыватель.

Наконец, загружаем и запускаем воспроизведение следующего видео.

Данный обработчик не будет вызываться, если пользователь поставит видео на паузу либо если видео воспроизводится в циклическом режиме (что можно обеспечить, задав соответствующее значение свойства `loop`).

Еще один тест-драйв...

Вы можете поверить, что мы готовы к тест-драйву? Все, что мы сделали, — это воспользовались соответствующим API-интерфейсом, чтобы обеспечить видео для воспроизведения, после чего позаботились о наличии слушателя событий, готового к обработке ситуации, возникающей по завершении проигрывания видео (что он и делает путем запуска воспроизведения следующего видео в плейлисте). Убедитесь, что вы внесли необходимые изменения в свой HTML-файл, нанекачайте новый JavaScript-код и проведите тест-драйв.



Вот что мы видим, при этом вы можете воспользоваться перемоткой, чтобы увидеть, как одно видео сменяет другое, а не смотреть все шоу целиком.

Все работает! Но как мы решим, какой формат видео воспроизводить, когда будем использовать код для загрузки источника видео?

Хороший вопрос.

Когда мы использовали множественные теги `<source>`, мы могли рассчитывать, что браузер пройдет по одному или более форматам видео и решит, сможет ли он воспроизвести какой-либо из них. Теперь при использовании кода мы просто даем элементу `video` единственный вариант выбора. Так как же нам проверить и узнать, какие форматы видео поддерживает браузер, чтобы убедиться в том, что мы обеспечили наиболее подходящий из них?

Для этого мы можем воспользоваться методом `canPlayType` объекта `video`. Он принимает формат видео и возвращает строку, которая дает понять, насколько браузер уверен в том, что он сможет воспроизвести данный тип видео. Существуют три степени уверенности: *вероятно, может быть, уверенности нет*. Давайте более пристально взглянем на `canPlayType`, а затем доработаем код нашего плейлиста, чтобы задействовать в нем данный метод.

Вы почесываете затылок, произнося: «Вероятно? Может быть? А почему этот метод не возвращает `true` или `false`»? С нами было то же самое! Через несколько мгновений вы узнаете, что под всем этим понимается...



Как работает метод canPlayType

Объект `video` предусматривает наличие метода `canPlayType`, который может определять, насколько высока вероятность того, что вам удастся воспроизвести тот или иной формат видео. Метод `canPlayType` принимает то же самое описание формата, которое вы использовали в случае с тегом `<source>`, и возвращает одно из трех значений: пустую строку, "maybe" или "probably". Вот как осуществляется вызов `canPlayType`:

Если мы передадим только краткое описание формата, то в качестве результата сможем получить лишь "" или "maybe".

```
video.canPlayType("video/ogg")
```

```
video.canPlayType('video/ogg; codecs="theora, vorbis"')
```

Однако если мы передадим определенный тип с кодеком, то сможем получить в ответ "", "maybe" либо "probably".

Будет возвращена пустая строка, если браузер поймет, что не сможет воспроизвести видео.

Будет возвращена строка "maybe", если браузер посчитает, что ему, возможно, удастся воспроизвести видео.

Будет возвращена строка "probably", если браузер будет уверен в том, что сможет воспроизвести видео.

Обратите внимание, что браузер будет уверен больше, чем "maybe", только в том случае, если вы наряду с типом MIME укажете значение параметра `codecs`. Также обращаем ваше внимание на то, что возвращаемого значения "I'm absolutely sure" не существует. Даже если браузер знает, что сможет воспроизвести *тип* видео, но-прежнему не будет никаких гарантий, что он сможет воспроизвести *фактическое* видео: например, битрейт видео может оказаться слишком высоким и браузеру не удастся декодировать его.

Битрейт — это количество битов, которое браузеру необходимо успевать обрабатывать за единицу времени, чтобы декодировать видео и корректно отобразить его.

Использование canPlayType

Мы собираемся использовать `canPlayType` для определения того, какой формат видео будет применяться в случае с видео Webville TV. Вы уже знаете, что у нас имеются три версии каждого файла: MP4, WebM и Ogg, и в зависимости от используемого вами браузера одни из них будут работать, а другие нет. Создадим новую функцию, которая будет возвращать расширение файла (".mp4", ".webm" или ".ogv"), подходящее для вашего браузера. Мы станем указывать только типы MIME ("video/mp4", "video/webm" и "video/ogg") без значений `codecs`, поэтому возможными возвращаемыми значениями у нас будут "maybe" или пустая строка:

```
function getFormatExtension() {
  if (video.canPlayType("video/mp4") != "") {
    return ".mp4";
  } else if (video.canPlayType("video/webm") != "") {
    return ".webm";
  } else if (video.canPlayType("video/ogg") != "") {
    return ".ogv";
  }
}
```

При большинстве вариантов использования, если вы не будете знать значений `codecs`, вполне достаточно будет уверенности "maybe".

Мы знаем, что сможем получить на-зад только значение "maybe" либо пустую строку, поэтому сможем удостовериться лишь в том, что в случае с нашим соответствующим типом не окажется возвращена пустая строка.

Мы будем проверять каждый из типов и возвращать соответствующее расширение файла, если браузер скажет: «Я может быть, и смогу его воспроизвести».

Интеграция функции `getFormatExtension`

Теперь нам необходимо внести ряд изменений в функции `window.onload` и `nextVideo` с целью задействовать `getFormatExtension`. Сначала мы удалим расширения файлов из файловых имен в плейлисте (несколько вместо этого мы будем применять `getFormatExtension` для их выяснения), а затем вызовем `getFormatExtension` там, где мы задали свойство `video.src`:

```
window.onload = function() {  
    playlist = ["video/preroll",  
               "video/areyoupopular",  
               "video/destinationearth"];  
    video = document.getElementById("video");  
    video.addEventListener("ended", nextVideo, false);  
    video.src = playlist[position] + getFormatExtension();  
    video.load();  
    video.play();  
}
```

Удалите расширения файлов.
Теперь мы будем выяснять их программным путем.

И конкатенируйте результат `getFormatExtension` с файловым именем источника нового видео.

И проделайте то же самое в `nextVideo`:

```
function nextVideo() {  
    position++;  
    if (position >= playlist.length) {  
        position = 0;  
    }  
    video.src = playlist[position] + getFormatExtension();  
    video.load();  
    video.play();  
}
```

Здесь мы делаем то же самое — конкатенируем результат `getFormatExtension` с файловым именем источника видео.

И проведение тест-драйва...

Добавьте функцию `canPlayType` и внесите продемонстрированные чуть выше изменения, после чего нерезагрузите свой файл `webvilletv.html`. Ну как, работает? Теперь ваш код способен выяснять наиболее подходящий формат. Если вы захотите узнать, какое видео выбрал браузер, не забудьте добавить код `alert` в функции `window.onload` и `nextVideo`; внесите его в нижнюю часть каждой функции после `video.play()`:

```
alert("Playing " + video.currentSrc);
```

Какой файл воспроизвел ваш браузер?



Часть Задаваемые Вопросы

В: Если я программно задаю источник своего видео и `capPlayType` возвращает «`таубе`», но воспроизведение все же не происходит, как это можно исправить?

О: Исправить это можно двумя путями. Первый состоит в том, чтобы перехватывать соответствующую ошибку и предоставлять объекту `video` другой источник (о перехвате ошибок мы поговорим в конце главы). Второй заключается в использовании объектной модели документа (DOM) для написания сразу нескольких тегов

`<source>` в объекте `video` (как если бы вы набрали их в своей разметке). Благодаря этому у вашего объекта `video` будет несколько вариантов на выбор, и вам не придется заниматься написанием более сложного кода для обработки ошибок. Мы не станем делать этого в данной главе, однако знайте о таком способе дать вашему объекту `video` несколько вариантов на выбор, причем посредством кода, а не разметки.



**Будьте
осторожны!**

Вам может потребоваться установить **Quicktime** для воспроизведения видеофайлов с расширением **MP4** в браузере **Safari**.

Программа **Quicktime** зачастую устанавливается по умолчанию, но если она все же не была инсталлирована, вы можете загрузить ее отсюда: <http://www.apple.com/quicktime/download/>.



**Будьте
осторожны!**

Браузер Google Chrome предусматривает дополнительные ограничения с целью обеспечения безопасности.

Данные ограничения не позволят вам совершать некоторые операции `video+canvas`, если вы загрузите веб-страницу как файл (то есть ваш URL-адрес будет выглядеть как `file://`, а не `http://`), подобно тому, как мы будем поступать в оставшейся части главы. Если вы попытаетесь так сделать, то приложение не будет работать и вы не получите никаких уведомлений о причине этого. В данной главе мы рекомендуем вам использовать либо другой браузер, либо свой собственный сервер, запуская примеры с `http://localhost`.



**Будьте
осторожны!**

Убедитесь, что с вашего сервера загружаются видеофайлы, имеющие корректный тип **MIME**.

Независимо от того, используете вы свой собственный локальный сервер или выполняете приложение, задействуя видео с онлайн-сервера, нужно удостовериться в том, что с сервера поступают видеофайлы с корректным типом **MIME**. В противном случае они не смогут работать должным образом.

Если вы работаете в операционной системе **Mac OS** или **Linux**, то, скорее всего, используете **Apache**. Вы можете модифицировать файл `httpd.conf` (если у вас имеется корневой доступ) либо создать файл `.htaccess` в каталоге, где располагаются ваши видеофайлы, и добавить в него следующие строки:

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

Этот код сообщит серверу, как подходить к загрузке файлов с данными расширениями.

Вы можете установить **Apache** в операционной системе **Windows** и произвести аналогичные манипуляции. В случае с серверами **IIS** рекомендуем заглянуть в онлайн-документацию **Microsoft** и найти там «**Конфигурирование типов MIME в IIS**» (*Configuring MIME types in IIS*).

Я неустанно твержу, что здесь речь идет не только о JavaScript... Необходимо видеть общую картину. Создание веб-приложений подразумевает использование разметки, CSS, JavaScript и его API-интерфейсов.



В какой-то момент нам уже придется относиться к вам как к настоящим разработчикам.

В этой книге мы (надеемся) помогаем вам делать каждый шаг — мы находимся рядом, чтобы подхватить вас, нрежде чем вы унадете, и убедиться в том, что в вашем коде были расставлены все точки над *i*. Однако быть настоящим разработчиком номимо всего остального означает нонимать код, нанисанный другими людьми, уметь увидеть главное за массой второстепенных деталей и разбираться в запутанности того, как все объединяется в одно целое.

В оставшейся части главы мы нредоставим вам возможность сделать все это. Далее нриведен нример, который наиболее близок к реальному веб-нриложению из всего того, что нам доводилось видеть до настоящего момента, нри этом он включает массу составных частей, активное иснользование API-интерфейсов и большое количество кода, который обеснечивает обработку множества реальных деталей. Тенерь мы не можем разбирать каждую часть, разъясняя вам каждый нюанс, как обычно это делали (иначе книга разрослась бы до 1200 страниц); мы и не хотим этого делать, носкольку вам необходимо нприобретать навыки объединения всех кусочков «мозаики» *без нас*.

Не беспокойтесь, носкольку мы ннрежнему будем рядом и расскажем, что именно делать, однако вы должны учиться воснринимать код, читать его, разбираться в нем, а затем доносить и изменять его для того, чтобы он делал *то, что вам необходимо*. Таким образом, на нпротяжении следующих трех глав мы хотим, чтобы вы ногрузились в нприводимые нримеры, изучили их и «вбили» код себе в голову. Да... вы к этому готовы!

Нам требуется ваша помощь!

Свежая новость... мы получили контракт на создание программного обеспечения компании **Starring You Video** для их новых видеокабинок. Что это такое? Это новейшие кабинки с поддержкой HTML5 для записи видеосообщений: клиент заходит в уединенную кабинку и записывает свое видеосообщение. Затем он сможет разнообразить свое видео, используя настоящие киноэффекты; в его распоряжении будет сепия-фильтр Old-time western (Сепия), черно-белый фильтр Film noir (Очень темный) и даже фильтр не от мира сего Sci-fi (Инвертированное видео). После этого клиент сможет отправить свое сообщение другу. Мы решили действовать и взяли на себя обязательство создать видеоинтерфейс и систему обработки видеоэффектов.

Однако есть одна проблема. Видеокабинки не будут доступны в течение следующих шести недель, а когда они придут, код уже должен быть готов. Таким образом, мы за это время собираемся обзавестись демоблоком с частичной функциональностью и несколькими тестовыми видеофайлами и написать весь код, используя их. Когда мы закончим, ребята из Starring You Video смогут просто применить наш код к только что снятому реальному видео. И, конечно же, не забывайте, что сделать все это нам необходимо с использованием HTML5.

Итак, мы надеемся, что вы с нами, поскольку мы подписали контракт!

Заходите, снимайте видео, стилизуйте его и отправляйте своим друзьям!



Заходите в видеокabinку, и давайте осмотримся...

Ниже приведен наш демоблок с интерфейсом нользователя. Там имеется видеозэкран, на котором нользователи смогут просматривать свое видео. У них будет возможность применить фильтр (например, Old-time western или Sci-fi) и посмотреть, как он работает, после чего они смогут отправить видео друзьям. Возможность заниси видео у нас пока отсутствует, поэтому мы предусмотрели наличие тестовых видеофайлов, с которыми можно будет работать. Наша первая задача — обеспечить работоспособность кнопок, а затем мы займемся написанием видеофильтров. Прежде чем мы приступим, взгляните на интерфейс:

Интерфейс демоблока. В центре располагается окно проигрывателя для просмотра видео.



Применяйте свой любимый эффект: Old-time western, Film noir или Sci-fi.

Элементы управления, чтобы воспроизводить, ставить на паузу и проигрывать видео в зацикленном режиме, а также отключать звук.

Выбирайте тестовое видео. Наш демоблок предлагает на выбор два таких видео.

Распаковка демоблока



На следующий день наш демоблок был доставлен самолетом, и настало время распаковать его. Похоже, что мы получили функциональный блок с уже написанной простой HTML-разметкой и JavaScript-кодом. Давайте сначала взглянем на HTML (videobooth.html). Впереди вас ждут несколько страниц «заводского» кода, просмотром которого мы займемся, а затем перейдем к настоящему коду.

```

<!doctype html>
<html lang="en">
<head>
  <title>Starring YOU Video Booth</title>
  <meta charset="utf-8">
  <link rel="stylesheet" href="videobooth.css">
  <script src="videobooth.js"></script>
</head>
<body>
<div id="booth">
  <div id="console">
    <div id="videoDiv">
      <video id="video" width="720" height="480"></video>
    </div>
    <div id="dashboard">
      <div id="effects">
        <a class="effect" id="normal"></a>
        <a class="effect" id="western"></a>
        <a class="effect" id="noir"></a>
        <a class="effect" id="scifi"></a>
      </div>
      <div id="controls">
        <a class="control" id="play"></a>
        <a class="control" id="pause"></a>
        <a class="control" id="loop"></a>
        <a class="control" id="mute"></a>
      </div>
      <div id="videoSelection">
        <a class="videoSelection" id="video1"></a>
        <a class="videoSelection" id="video2"></a>
      </div>
    </div>
  </div>
</div>
</body>
</html>

```

HTML5, конечно же

Кроме того, вся стилизация была сделана за нас! Вот соответствующий CSS-файл.

А вот JavaScript-файл, который нам потребуется для размещения большей части написанного кода. Чуть позже мы взглянем на него более подробно, и, похоже, что они уже написали здесь код для контроля над кнопками в интерфейсе.

Вот главный интерфейс, где у нас имеется сама консоль, которая разделена на область отображения видео и панель управления с тремя наборами кнопок, сгруппированных в "effects", "controls" и "videoSelection".

Они уже установили видеопроигрыватель... это хорошо, поскольку он нам понадобится.

Вот все эффекты

Это всего лишь HTML-якоря. О привязке к ним мы поговорим чуть позже...

И элементы управления проигрывателем

И два демовидео для тестирования



Рассмотрение остальной части «заводского» кода

Теперь взглянем на весь JavaScript-код, который мы получили с «завода», включая код для конфигурирования кнопок (мы только что его рассматривали в соответствующем HTML-файле) и код для каждого обработчика кнопки (который на текущий момент заботится о том, чтобы назад «отжимались» именно те нажатые кнопки, что и пужно). Мы рассмотрим весь этот код прежде, чем начнем добавлять свой собственный код.

А сейчас переходим к JavaScript...

Давайте откроем JavaScript-файл (videobooth.js). Похоже, что все кнопки в интерфейсе работают, просто они пока не делают ничего интересного. Мы знаем, как они сконфигурированы, поскольку эти кнопки будут вызывать код, который нам необходимо написать (например, для воспроизведения видео или для просмотра видео с применением фильтра, обеспечивающего тот или иной эффект).

Чуть ниже вы найдете функцию, вызов которой происходит по завершении загрузки страницы. В случае с каждым набором кнопок ("effects", "controls" и "videoSelection") код проходится по кнопкам и присваивает обработчики событий click якорным ссылкам. Давайте взглянем на это:

```

window.onload = function() {
  var controlLinks = document.querySelectorAll("a.control");
  for (var i = 0; i < controlLinks.length; i++) {
    controlLinks[i].onclick = handleControl;
  }

  var effectLinks = document.querySelectorAll("a.effect");
  for (var i = 0; i < effectLinks.length; i++) {
    effectLinks[i].onclick = setEffect;
  }

  var videoLinks = document.querySelectorAll("a.videoSelection");
  for (var i = 0; i < videoLinks.length; i++) {
    videoLinks[i].onclick = setVideo;
  }

  pushUnpushButtons("video1", []);
  pushUnpushButtons("normal", []);
}
    
```

Функция, которая будет вызываться после полной загрузки страницы

Каждый оператор совершает цикл по элементам одной из группы кнопок.

В случае с обработчиком onclick для каждой кнопки в элементах управления проигрывателем задается обработчик handleControl.

А в случае с обработчиком для effects задается setEffect.

И наконец, в случае с обработчиком для videoSelection задается setVideo.

Сделав всю подготовительную работу, мы используем вспомогательную функцию, чтобы визуально «отжать» назад кнопки "video1" и "normal" (без применения фильтров) в интерфейсе.

С методом `document.querySelectorAll` вы ранее не сталкивались; он аналогичен `document.getElementsByTagName`, за исключением того, что вы осуществляете выборку элементов, соответствующих селектору CSS. Данный метод возвращает массив объектов элементов, соответствующих аргументу в виде указанного селектора CSS.

```
var elementArray = document.querySelectorAll("selector");
```



Взгляд на обработчики кнопок

Итак, JavaScript-код заботится о конфигурировании всех кнопок, чтобы при щелчке пользователя на них произошел вызов соответствующего обработчика. Давайте взглянем на фактические обработчики, начав с обработчика для кнопок проигрывателя (Play (Воспроизведение), Pause (Пауза), Loop (Воспроизведение в зацикленном режиме) и Mute (Отключить звук)), и посмотрим, что они делают:

Первое, что мы делаем в данном обработчике, это выясняем, кто совершил вызов обработчика, для чего извлекаем идентификатор элемента, который это сделал.

```
function handleControl(e) {
    var id = e.target.getAttribute("id");

    if (id == "play") {
        pushUnpushButtons("play", ["pause"]);
    } else if (id == "pause") {
        pushUnpushButtons("pause", ["play"]);
    } else if (id == "loop") {
        if (isButtonPushed("loop")) {
            pushUnpushButtons("", ["loop"]);
        } else {
            pushUnpushButtons("loop", []);
        }
    } else if (id == "mute") {
        if (isButtonPushed("mute")) {
            pushUnpushButtons("", ["mute"]);
        } else {
            pushUnpushButtons("mute", []);
        }
    }
}
```

Вывяснив идентификатор, мы узнаем, какой элемент это был: "play", "pause", "loop" или "mute".

В зависимости от того, какая это окажется кнопка, мы изменим интерфейс, чтобы он отражал нажатую кнопку. Например, если была нажата кнопка Pause (Пауза), — в этом случае кнопка Play (Воспроизведение).

Мы используем вспомогательную функцию, чтобы позаботиться о состояниях кнопок на экране, с именем pushUnpushButtons, которая принимает значение id нажатой кнопки наряду с массивом значений id ненажатых кнопок и обновляет интерфейс с целью отражения в нем данного состояния.

Разные кнопки обладают разной семантикой. Например, Play (Воспроизведение) и Pause (Пауза) подобны настоящим радиокнопкам, в то время как Mute (Отключить звук) и Loop (Воспроизведение в зацикленном режиме) сродни кнопкам-переключателям.

Весь этот показанный код является косметическим и лишь изменяет внешний вид кнопок с нажатого состояния на ненажатое. У нас пока отсутствует код для выполнения чего-то реального, например для воспроизведения видео. Его нам и предстоит написать.

Все это, конечно, здорово, однако где будет располагаться *наш код*? Давайте подумаем: когда пользователь нажмет кнопку, например Play (Воспроизведение), нам не только нужно будет обеспечить обновление интерфейса (для чего у нас уже имеется код), но и предусмотреть выполнение дополнительного кода, который действительно *что-то* делает, например запускает воспроизведение видео. Двинемся дальше и взглянем на два других обработчика (для задания видеоэффектов и тестового видео), и вам станет вполне ясно (если уже не стало), где именно будет располагаться наш код...

Обработчику setEffect и setVideo



Обработчик `setEffect` обрабатывает ваш выбор эффекта: например, применяемые эффекты могут вообще отсутствовать (при выборе "normal") либо быть "western", "noir" или "scifi". Аналогичным образом, обработчик `setVideo` обрабатывает ваш выбор тестового видео под номером 1 или 2:

```
function setEffect(e) {
  var id = e.target.getAttribute("id");

  if (id == "normal") {
    pushUnpushButtons("normal", ["western", "noir", "scifi"]);
  } else if (id == "western") {
    pushUnpushButtons("western", ["normal", "noir", "scifi"]);
  } else if (id == "noir") {
    pushUnpushButtons("noir", ["normal", "western", "scifi"]);
  } else if (id == "scifi") {
    pushUnpushButtons("scifi", ["normal", "western", "noir"]);
  }
}
```

Он работает так же, как обработчик `handleControl`: мы извлекаем идентификатор осуществившего вызов элемента (кнопки, которую нажал пользователь), а затем соответствующим образом обновляем интерфейс.

А здесь будет располагаться наш код.

В каждом случае мы добавим код для обработки реализации соответствующего фильтра, обеспечивающего спецэффекты.

То же самое справедливо и для `setVideo`; мы выясняем, какая кнопка была нажата, и обновляем интерфейс.

```
function setVideo(e) {
  var id = e.target.getAttribute("id");
  if (id == "video1") {
    pushUnpushButtons("video1", ["video2"]);
  } else if (id == "video2") {
    pushUnpushButtons("video2", ["video1"]);
  }
}
```

Мы также добавим сюда код для реализации переключения между двумя тестовыми видео.

А вот и вспомогательные функции

И не забывайте, что если вы не хотите набирать приводимый в книге код, можете целиком загрузить его, посетив адрес <http://wickedlysmart.com/hfhtml5>.



И для полноты (либо если вы совершаете 11-часовой перелет на Фиджи, не имея при этом доступа в Интернет, и у вас возникло желание заняться набором всего этого кода):

pushUnpushButtons заботится о состояниях кнопок. Он принимает аргументы в виде значения *id* кнопки, которая будет выглядеть нажатой, а также значения *id* одной или более кнопок в массиве, которые будут выглядеть ненажатыми.

Сначала мы убеждаемся в том, что значение *id* кнопки для нажатия не является пустым.

```
function pushUnpushButtons(idToPush, idArrayToUnpush) {
    if (idToPush != "") {
        var anchor = document.getElementById(idToPush);
        var theClass = anchor.getAttribute("class");
        if (!theClass.indexOf("selected") >= 0) {
            theClass = theClass + " selected";
            anchor.setAttribute("class", theClass);
            var newImage = "url(images/" + idToPush + "pressed.png)";
            anchor.style.backgroundImage = newImage;
        }
    }

    for (var i = 0; i < idArrayToUnpush.length; i++) {
        anchor = document.getElementById(idArrayToUnpush[i]);
        theClass = anchor.getAttribute("class");
        if (theClass.indexOf("selected") >= 0) {
            theClass = theClass.replace("selected", "");
            anchor.setAttribute("class", theClass);
            anchor.style.backgroundImage = "";
        }
    }

    function isButtonPushed(id) {
        var anchor = document.getElementById(id);
        var theClass = anchor.getAttribute("class");
        return (theClass.indexOf("selected") >= 0);
    }
}
```

Извлекаем элемент *anchor* с использованием данного значения *id*...
 ...и извлекаем атрибут *class*.
 Мы «нажимаем» кнопку путем добавления класса "selected" в *anchor* и...
 ...обновляем фоновое изображение элемента *anchor*, чтобы оно перекрыло данную кнопку изображением «нажатой кнопки». Таким образом, в случае с "pause" будет использоваться изображение "pausepressed.png".
 Чтобы кнопки выглядели ненажатыми, мы совершаем цикл по массиву значений *id* кнопок, которые должны приобрести такой вид, извлекая каждый *anchor*...
 ...Убеждаемся, что кнопка действительно нажата (если это так, то у нее будет иметься класс "selected")...
 ...Удаляем "selected" из *class*...
 ...а также удаляем фоновое изображение, чтобы мы смогли увидеть ненажатую кнопку.
isButtonPushed просто проверяет, нажата ли кнопка. Он принимает *id* элемента *anchor*...
 ...извлекает *anchor*...
 ...извлекает *class* данного *anchor*...
 ...и возвращает *true*, если у *anchor* имеется класс "selected".

Запах новой демомашины... пришло время тест-драйва!

Мы нанисали не слишком много кода, однако мы читаем имеющийся код и вникаем в него, и это хорошо. Итак, загрузите файл `videobooth.html` в своем браузере и воспользуйтесь кнонками. Хорошенько протестируйте их. Также добавьте несколько `alert` в функции обработчиков. Прочувствуйте, как все это работает. Когда вы вернетесь, мы приступим к нанисанию кода, который заставит кнонки работать но-настоящему.



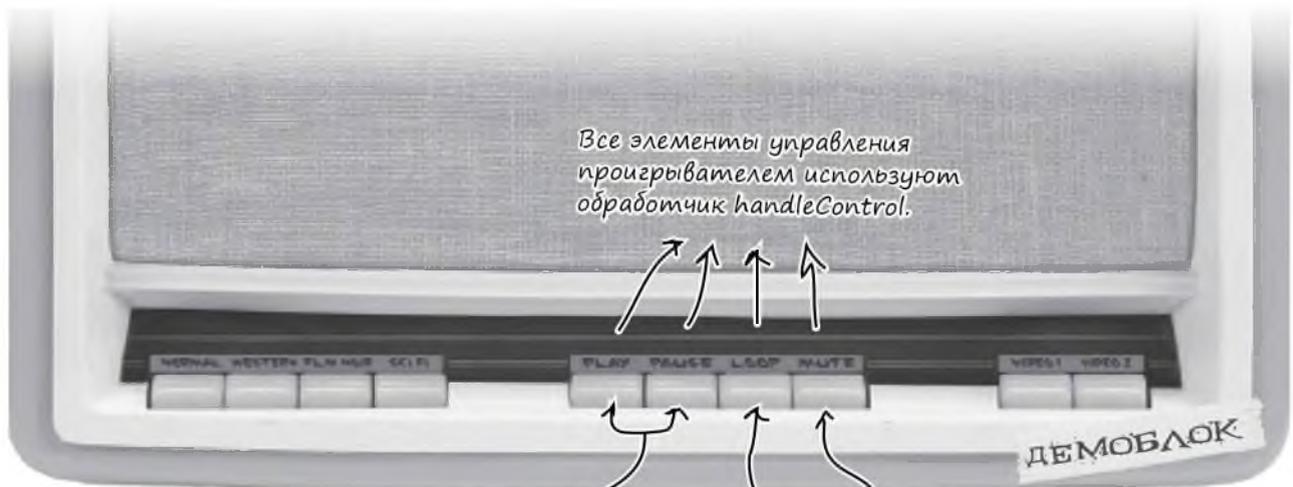
Протестируйте все эти кнопки, при этом обратите внимание на то, что одни из них подобны радиокнопкам, а другие сродни кнопкам-переключателям.

Возьми в руку карандаш



Решение к данному заданию вы найдете, пролистав вперед две страницы...

Пометьте, какие из приведенных ниже кнопок подобны переключателям (то есть являются независимыми), а какие — радиокнопкам, когда нажатие одной кнопки будет приводить к тому, что остальные перестанут быть нажатыми). Также напишите рядом с каждой кнопкой имя соответствующего обработчика. Мы уже сделали это за вас для двух пар кнопок:



Все элементы управления проигрывателем используют обработчик `handleControl`.

`Play` (Воспроизведение) и `Pause` (Пауза) подобны радиокнопкам и не могут быть выбраны одновременно.

`Loop` (Воспроизведение в зацикленном режиме) и `Mute` (Отключить звук) подобны кнопкам-переключателям и могут использоваться независимо от других кнопок.

Кажется, я что-то пропустила... Как вы проводили извлечение из элементов `<div>` с использованием якорных тегов `<a>`, чтобы в интерфейсе были кнопки?

Все благодаря мощи CSS.

Жаль, что эта книга не называется «Изучаем программирование на HTML5 с использованием JavaScript и CSS» (Head First HTML5 Programming with JavaScript & CSS), но тогда она содержала бы 1400 страниц, не так ли? Нас, конечно же, можно было бы уговорить написать продвинутую книгу по CSS...

А если серьезно, то мощь разметки направлена на структуру, а CSS — на представление (если это новая для вас тема, то загляните в книгу «Изучаем HTML, XHTML и CSS»). То, что мы делаем, не является таким уж сложным; опишем все вкратце для тех, кому любопытно.

Мы задаем картинку с консолью видеокабинки (без кнопок) в качестве фонового изображения для элемента `<div>` с `id` в виде `console`.



Элемент `<video>` располагается в `<div>`, который позиционируется относительно консоли. Кроме того, мы производим абсолютное позиционирование элемента `<video>`, чтобы он оказался в центре консоли.



Мы позиционируем `<div>` с `id` в виде `dashboard` относительно консоли, а затем позиционируем элементы `<div>` для каждой группы кнопок относительно панели управления.

Каждый элемент `<div>` группы кнопок получает фоновое изображение для всех ненажатых кнопок.

Каждый якорь `"button"` позиционируется в `<div>` для определенной группы и получает ширину и высоту, чтобы совпадать по размерам с кнопкой, которой он соответствует. Когда пользователь щелкнет `"button"`, мы снабдим данный якорь фоновым изображением нажатой кнопки, которое перекроет собой ненажатую кнопку.

Если вам захочется подробно исследовать CSS, загляните в файл `videobooth.css`.

Возьми в руку карандаш

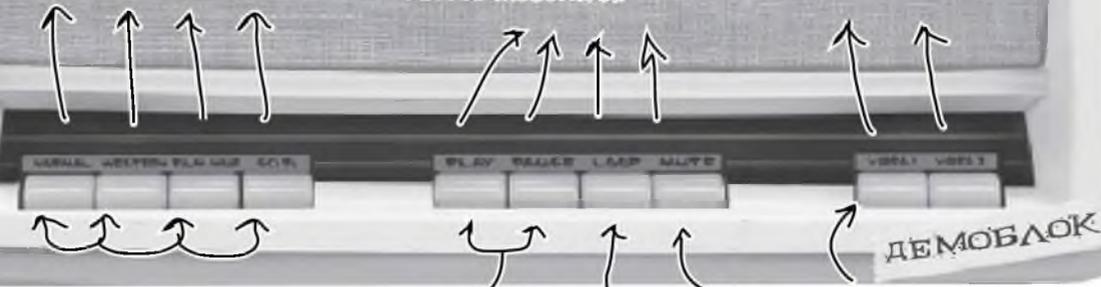


Пометьте, какие из приведенных ниже кнопок подобны переключателям (то есть являются независимыми), а какие — радиокнопкам (когда нажатие одной кнопки будет приводить к тому, что остальные перестанут быть нажатыми). Также напишите рядом с каждой кнопкой имя соответствующего обработчика. Вот наше решение этого упражнения.

Все кнопки для применения эффектов используют обработчик `setEffect`.

Все элементы управления проигрывателем используют обработчик `handleControl`.

Все кнопки для выбора видео используют обработчик `setVideo`.



Все эти кнопки подобны радиокнопкам; мы даем пользователю возможность применять к видео только один эффект за раз.

Play (Воспроизведение) и Pause (Пауза) подобны радиокнопкам и не могут быть выбраны одновременно.

Loop (Воспроизведение в зацикленном режиме) и Mute (Отключить звук) подобны кнопкам-переключателям и могут использоваться независимо от других кнопок.

Кнопки для выбора видео также подобны радиокнопкам, то есть за раз пользователь сможет выбрать только одно видео для просмотра.

Подготовка демовидео...

Прежде чем мы реализуем элементы управления `button`, нам потребуется видео для их тестирования, и, как можно судить по кнопкам, сотрудники `Starring You Video` прислали нам два демовидео. Создадим объект для размещения двух видео, а затем добавим код для нашего обработчика `onload`, чтобы задать источник объекта `video` (точно так же, как мы делали в случае с `Webville TV`).

Мы создадим этот объект для размещения двух демовидео. Вскоре мы вернемся к этому и объясним вам все подробнее.

```
var videos = {video1: "video/demovideo1", video2: "video/demovideo2"};
```

```
window.onload = function() {
```

```
    var video = document.getElementById("video");
    video.src = videos.video1 + getFormatExtension();
    video.load();
```

Извлекаем элемент `video` и даем в качестве его источника первое видео в массиве, имеющее расширение, которое можно будет воспроизвести.

```
    var controlLinks = document.querySelectorAll("a.control");
    for (var i = 0; i < controlLinks.length; i++) {
        controlLinks[i].onclick = handleControl;
    }

    var effectLinks = document.querySelectorAll("a.effect");
    for (var i = 0; i < effectLinks.length; i++) {
        effectLinks[i].onclick = setEffect;
    }

    var videoLinks = document.querySelectorAll("a.videoSelection");
    for (var i = 0; i < videoLinks.length; i++) {
        videoLinks[i].onclick = setVideo;
    }

    pushUnpushButtons("video1", []);
    pushUnpushButtons("normal", []);
}
```

Затем мы загружаем видео, благодаря чему оно будет готово к проигрыванию, если пользователь щелкнет на кнопке воспроизведения.

ВНИМАТЕЛЬНО ПРОЧИТАЙТЕ!

Чтобы не допустить небрежности, не забывайте, что в `Webville TV` есть функция `getFormatExtension`, но не этот код! Поэтому откройте файл `webvilletv.js`, скопируйте и вставьте данную функцию в код своего приложения `Video Booth`. И еще один небольшой аспект: в коде `Video Booth` у нас нет глобального объекта `video`, поэтому добавьте данную строку в верхнюю часть своей функции `getFormatExtension`, чтобы исправить это:

```
var video = document.getElementById("video");
```

Добавьте данную строку в верхнюю часть функции `getFormatExtension`.



Сейчас мы собираемся реализовать все эти кнопки.

Реализация элементов управления видео

Пришло время заняться кнопками! Важно отметить, что в данном проекте мы собираемся реализовать *собственные* элементы управления видео. То есть вместо того чтобы использовать встроенные элементы управления видео, мы сами будем контролировать данное взаимодействие. Таким образом, когда пользователям потребуется воспроизвести, поставить на паузу видео, отключить в нем звук или даже проиграть его в заикленном режиме, они будут применять наши пользовательские элементы управления `button`, а не встроенные. Это также означает, что мы будем делать все это программно с помощью соответствующего API-интерфейса. Сейчас мы не собираемся идти до конца, поскольку это означало бы реализацию, например, кнопок `Next` (Далее) и `Previous` (Назад), смысла в которых в данном приложении нет, но мы могли бы это сделать при необходимости. На простом примере реализации небольшой панели управления вы сможете понять идею и развить ее дальше, если пожелаете.

Итак, приступим. Как насчет того, чтобы начать с кнопки `Play` (Воспроизведение), а затем двинуться от нее вправо, к кнопке `Pause` (Пауза), после нее — к `Loop` (Воспроизведение в заикленном режиме), а затем — к `Mute` (Отключить звук)? Найдите обработчик `handleControl` и добавьте туда следующий код:

```
function handleControl(e) {
  var id = e.target.getAttribute("id");
  var video = document.getElementById("video");

  if (id == "play") {
    pushUnpushButtons("play", ["pause"]);
    if (video.ended) {
      video.load();
    }
    video.play();
  } else if (id == "pause") {
    pushUnpushButtons("pause", ["play"]);
  } else if (id == "loop") {
    if (isButtonPushed("loop")) {
      pushUnpushButtons("", ["loop"]);
    } else {
      pushUnpushButtons("loop", []);
    }
  } else if (id == "mute") {
    if (isButtonPushed("mute")) {
      pushUnpushButtons("", ["mute"]);
    } else {
      pushUnpushButtons("mute", []);
    }
  }
}
```

← Нам необходима ссылка на объект `video`.

← Это должно быть довольно просто для вас. Если пользователь щелкнет на кнопке `Play` (Воспроизведение), то произойдет вызов метода `play` в отношении объекта `video`.

← Но хотим предупредить, что здесь имеется один сложный момент, который может оказаться хлопотным: если бы мы воспроизвели видео и позволили ему проиграться до конца, то для того, чтобы опять запустить его, нам сначала пришлось бы снова его загрузить. Мы проверяем, что видео проигралось до конца (а не было просто поставлено на паузу), поскольку в данном случае нам потребуется лишь снова загрузить его. Если оно было поставлено на паузу, мы сможем воспроизвести его, не прибегая к загрузке.

Реализация остальных элементов управления видео

Реализуем остальные элементы управления — они настолько просты, что чуть ли не сами могут написать свой код:

```
function handleControl(e) {
    var id = e.target.getAttribute("id");
    var video = document.getElementById("video");

    if (id == "play") {
        pushUnpushButtons("play", ["pause"]);
        video.load();
        video.play();
    } else if (id == "pause") {
        pushUnpushButtons("pause", ["play"]);
        video.pause();
    } else if (id == "loop") {
        if (isButtonPushed("loop")) {
            pushUnpushButtons("", ["loop"]);
        } else {
            pushUnpushButtons("loop", []);
        }
        video.loop = !video.loop;
    } else if (id == "mute") {
        if (isButtonPushed("mute")) {
            pushUnpushButtons("", ["mute"]);
        } else {
            pushUnpushButtons("mute", []);
        }
        video.muted = !video.muted;
    }
}
```

Если пользователь поставит видео на паузу, то будет задействован метод `pause` объекта `video`.

Для воспроизведения видео в за-
цикленном режиме у нас в объекте `video` имеется логическое свойство `loop`. Мы просто присваиваем ему соответствующее значение...

...для чего, чтобы было еще
интереснее, используем логический оператор `!` («не»), который просто меняет за нас логическое значение.

В случае с `mute` все происходит аналогичным образом: мы просто меняем текущее значение свойства `mute`, когда пользователь нажимает кнопку `Mute` (Отключить звук).

Еще один тест-драйв!



Убедитесь, что вы внесли в код все приведенные выше изменения. Загрузите `videobooth.html` в браузере и протестируйте элементы управления `button`. Вы должны видеть, как запускается воспроизведение видео, иметь возможность ставить его на паузу, отключать в нем звук и даже проигрывать его в зацикленном режиме. Вы, конечно же, пока не сможете выбрать другое демовидео или добавить эффект, однако вскоре мы дойдем и до этого!



Дорабатываем один нюанс...

Нам необходимо доработать один небольшой нюанс, чтобы кнопки действительно работали так, как надо. Вот вариант использования: допустим, вы воспроизводите видео и при этом не нажали кнопку Loop (Воспроизведение в цикленном режиме), в результате чего проигрывание видео осуществляется до конца и затем завершается. В силу нашего подхода к реализации кнопка Play (Воспроизведение) будет оставаться в нажатом состоянии. А не будет ли лучше, если она вернется в исходное положение и будет готова к новому нажатию?

Используя события, мы можем легко обеспечить данное поведение. Начнем с добавления слушателя событий ended. Внесите данный код в нижнюю часть обработчика onload:

```
video.addEventListener("ended", endedHandler, false);
```

Теперь напишем обработчик, вызов которого будет происходить каждый раз, когда воспроизведение видео будет завершаться при достижении конца видеофайла:

```
function endedHandler() {  
    pushUnpushButtons("", ["play"]);  
}
```

← Нам нужно лишь сделать так, чтобы кнопка Play (Воспроизведение) снова приобрела ненажатое состояние. И все!

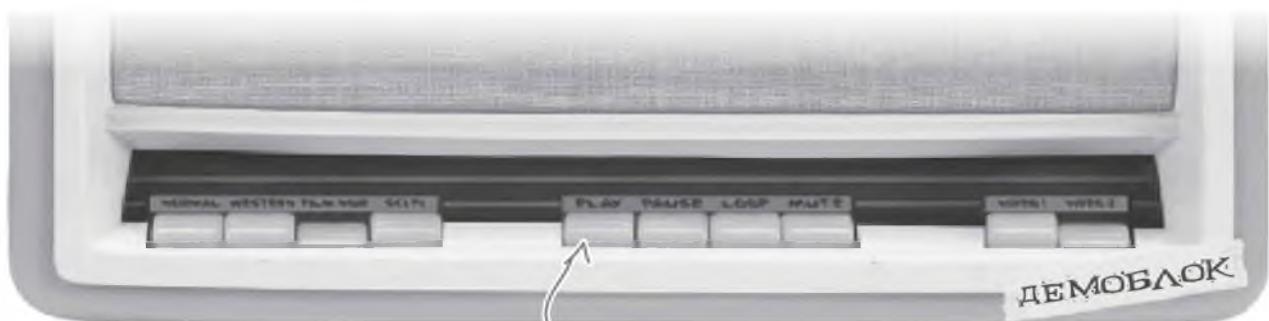
← Мы задаем обработчик для события "ended", который будет вызываться по завершении воспроизведения видео (но НЕ когда видео будет ставиться на паузу!).



Наша кнопка Play (Воспроизведение) пока не является на 100% такой, какой она должна быть.

И еще один тест-драйв...

Итак, внесите изменения, сохраните код и перезагрузите страницу. Запустите видео, дайте ему проиграться до конца и не нажимайте кнопку Loop (Воспроизведение в цикленном режиме). В результате вы должны увидеть, как в конце кнопка Play (Воспроизведение) сама вернется в исходное положение.



Кнопка Play (Воспроизведение) вернется в исходное положение сразу же по завершении проигрывания видео.

Переключение между тестовыми video

Мы уже добавили объект для размещения наших двух тестовых видео и даже располагаем двумя кнопками для выбора между ними. Каждой кнопке присвоен обработчик `setVideo`. Давайте взглянем на код, который позволит переключаться между этими видео:

Вот наш объект с двумя видео, который мы приводим снова в качестве напоминания, чтобы вы смогли понять, как мы собираемся его использовать...



```
var videos = {video1: "video/demovideo1", video2: "video/demovideo2"};
```

```
function setVideo(e) {
```

← А вот снова наш обработчик.

```
var id = e.target.getAttribute("id");
```

```
var video = document.getElementById("video");
```

← Опять-таки, нам требуется ссылка на объект `video`.

```
if (id == "video1") {
```

```
    pushUnpushButtons("video1", ["video2"]);
```

```
} else if (id == "video2") {
```

```
    pushUnpushButtons("video2", ["video1"]);
```

```
}
```

← Затем мы обновляем кнопки тем же путем, что и раньше, никаких изменений здесь нет.

```
video.src = videos[id] + getFormatExtension();
```

```
video.load();
```

```
video.play();
```

← Далее мы используем `id` кнопки (атрибут `id` элемента `anchor`) для извлечения соответствующего имени видеофайла и прибавляем расширение, поддерживаемое нашим браузером. Обратите внимание, что мы применяем скобочную нотацию в случае с объектом `videos`, используя строку `id` в качестве имени свойства.

```
pushUnpushButtons("play", ["pause"]);
```

Мы удостоверяемся, что кнопка `Play` (Воспроизведение) нажата, поскольку запустим проигрывание видео, когда пользователь щелкнет на кнопке для выбора нового видео.

Имея корректный путь к видео и его файловое имя, мы загружаем и воспроизводим данный видеофайл.



Внесите изменения и проведите тест-драйв!

Внесите данные изменения в вашу функцию `setVideo`, а затем перезагрузите страницу. После этого у вас должна появиться возможность легко переключаться между источниками видео.





СЕКРЕТЫ HTML5

Интервью недели:
Признания элемента video

Head First: Добро пожаловать, Video. Сразу перейду к вопросу, который всех интересует, и заключаться он будет в отношениях между Вами и элементом canvas.

Элемент Video: Что Вы имеете в виду?

Head First: Вы якобы кутите ночи нанролет в вихре удовольствий, завтракаете вместе но утрам. Нужно ли еще что-то говорить?

Video: Здесь следует отметить, что у нас с Canvas сложились прекрасные взаимоотношения. Canvas отображает свое содержимое в весьма привлекательной в визуальном плане манере, а я являюсь «рабочей видеолошадкой». Я разбираюсь с кодеками и доставляю видеосодержимое в браузер.

Head First: Что ж, это не совсем тот ответ, на который я рассчитывал, но он меня устраивает. Ладно, элемент canvas прекрасно справляется с отображением 2D-графики, а Вы — с отображением видео. И что? В чем здесь истинная связь?

Video: Как и в любых взаимоотношениях, когда вы объединяете две вещи и имеете на выходе нечто большее, чем сумма двух частей, то вы получаете нечто особенное.

Head First: Что ж, а не могли бы Вы выразиться более конкретно?

Video: Это простая концепция. Если вы хотите делать что-то еще помимо обычного воспроизведения видео (например, осуществлять обработку своего видео, либо задействовать пользовательские оверлеи, либо одновременно выводить несколько видеоизображений), то вам потребуется использовать canvas.

Head First: Все это звучит здорово, однако видео требует больших вычислительных ресурсов для обработки. Я имею в виду, что при этом проходит огромный поток данных. Как JavaScript, язык сценариев, сможет сделать здесь что-либо реальное? Написание JavaScript-кода — не то же самое, что программирование на нативном языке.

Video: О, для вас это будет сюрпризом... Вы смотрели последние тесты производительности JavaScript? Он уже является быстрым, и его скорость с каждым днем повышается. Самые блестящие специалисты по виртуальным машинам в индустрии успешно работают над этой проблемой.

Head First: Да, но что с видео? Действительно ли с ним все обстоит так?

Video: Действительно.

Head First: Не могли бы Вы привести примеры того, что можно сделать с помощью JavaScript, canvas и video?

Video: Конечно. Вы можете обрабатывать видео в режиме реального времени, инспектировать характеристики видео, извлекать данные из видеокадров, модифицировать видеоданные путем, например... вращения, масштабирования видео или даже изменения пикселей.

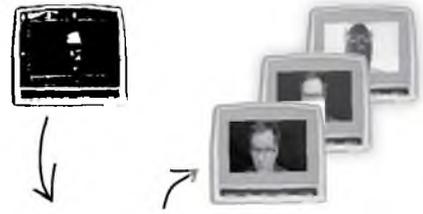
Head First: Не могли бы Вы нам поведать, как это можно сделать в коде?

Video: Я еще вернусь к Вам, чтобы обсудить данный вопрос, мне тут просто ностунил звонок от Canvas... Нужно бежать...

Время спецэффектов

Не нора ли нам добавить видеоэффекты? Мы хотим применять эффекты к нашему оригинальному видео, такие как Film noir, Old-time western и даже Sci-fi. Но если вы взглянете на API-интерфейс Video, то не найдете там каких-либо методов для применения эффектов либо способов добавить их напрямую. Так как же мы будем добавлять эти эффекты?

Немного подумайте над тем, как мы могли бы добавить эффекты в наше видео. Не беспокойтесь, если вы пока еще не знаете, как обрабатывать видео, просто обдумайте высокоуровневый дизайн.



Мы хотим применять к нашему оригинальному видео такие эффекты, как Film noir, Old-time western и Sci-fi.

Starring You Video
Заметки по разработке...

Используйте эти заметки по разработке, чтобы сделать набросок, пометки или написать псевдокод для любого кода, который относится к вашим видеоэффектам. Считайте это своего рода разминкой для мозга...

Как вы сможете добраться до пикселей, образующих каждый кадр видео?

Добравшись до пикселей, как вы будете обрабатывать их с целью применения определенного эффекта?

Допустим, вам необходимо написать функцию для реализации каждого эффекта, — как она будет выглядеть?

Как вы сможете отобразить видео, когда обрабатываете все его пиксели с целью применения определенного эффекта?

↑
Свои идеи напишите здесь.



План реализации спецэффектов

Мы пока не знаем точно, как реализовать эффекты, однако высокоуровневый план действий будет выглядеть следующим образом:

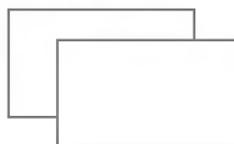
- 1 Мы знаем, что нам все еще нужно подключить соответствующие кнопки, которые управляют эффектами. Поэтому мы займемся этим в первую очередь.
- 2 Мы немного изучим особенности видеообработки и познакомимся с методикой на основе применения временного буфера для добавления наших эффектов.
- 3 Мы реализуем временный буфер, который даст нам возможность увидеть совместно в действии `video` и `canvas`.
- 4 Мы реализуем по одной функции для каждого эффекта: `western`, `noir` и `scifi`.
- 5 И наконец, мы соединим все воедино и проведем тестирование!



Кнопки все еще ждут, когда мы их подключим.



Временный буфер, который интересно выглядит...



Мы реализуем временный буфер, используя `canvas` (хотите — верьте, хотите — нет)!

```
function noir(pos, r, g, b, data) {  
  ...  
}
```



Измененные пиксели мы будем отображать, как вы уже догадались, в `canvas`.



Теперь вы знаете, что мы собираемся реализовать функцию, которая будет обрабатывать каждый эффект. Возьмем, к примеру, эффект `Film noir`. Как вы будете выбирать тот или иной цветной пиксел из видео и делать его черным либо белым? Подсказка: каждый пиксел включает три компонента: `r` (`red` — красный), `g` (`green` — зеленый) и `b` (`blue` — синий). Если бы у нас был доступ к этим составным частям, то чтобы мы могли сделать?



Пора обеспечить работоспособность кнопок для применения эффектов

Сначала легкая часть: мы подключим соответствующие кнопки и сделаем так, чтобы они работали. Начнем с создания глобальной переменной `effectFunction`. Она будет содержать функцию, которая сможет извлекать данные из видео и применять к ним фильтр. То есть в зависимости от желаемого эффекта переменная `effectFunction` будет содержать функцию, которая знает, как обрабатывать видеоданные и делать их черно-белыми либо применять к ним сению или инвертирование. Итак, добавьте эту переменную в верхнюю часть своего файла:

```
var effectFunction = null;
```

Мы станем присваивать этой функции соответствующее значение каждый раз, когда пользователь будет щелкать на кнопке для применения эффекта. Сейчас мы воспользуемся такими именами функций, как `western`, `noir` и `scifi`, а сами эти функции напишем немного позже.

Вот снова наш обработчик `setEffect`. Как вы помните, его вызов происходит всякий раз, когда пользователь щелкает на кнопке для применения эффекта.

```
function setEffect(e) {
    var id = e.target.getAttribute("id");

    if (id == "normal") {
        pushUnpushButtons("normal", ["western", "noir", "scifi"]);
        effectFunction = null;
    } else if (id == "western") {
        pushUnpushButtons("western", ["normal", "noir", "scifi"]);
        effectFunction = western;
    } else if (id == "noir") {
        pushUnpushButtons("noir", ["normal", "western", "scifi"]);
        effectFunction = noir;
    } else if (id == "scifi") {
        pushUnpushButtons("scifi", ["normal", "western", "noir"]);
        effectFunction = scifi;
    }
}
```

При каждом нажатии кнопки пользователем мы будем присваивать переменной `effectFunction` значение в виде соответствующей функции (все эти функции нам еще предстоит написать).

Если пользователь решит не применять эффекты, то есть выберет "normal", мы присвоим значение `null`.

В иных случаях мы будем присваивать `effectFunction` значение в виде имени соответствующей функции, которая займется применением определенного эффекта.

Нам по-прежнему нужно написать все эти функции для применения эффектов. Давайте посмотрим, как осуществляется обработка видео, чтобы мы смогли применить к нему эффекты!

Итак, мы можем переходить к изучению временного буфера, после чего вернемся и посмотрим, как данные функции вносятся в общую картину и как осуществляется их написание!

Как происходит обработка видео

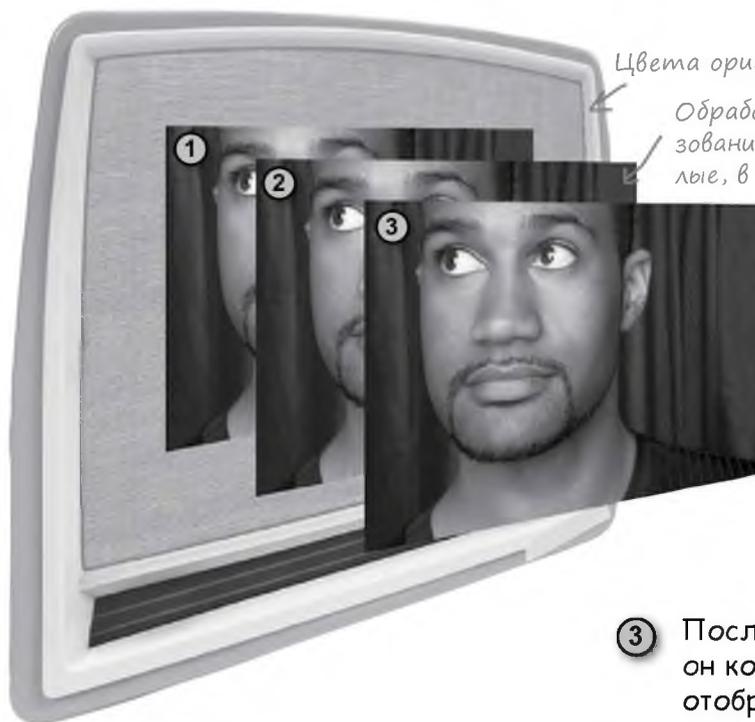
Ранее мы обеспечили способ присваивать функцию глобальной переменной `effectsFunction` в результате нажатия пользователем кнопок для применения эффектов в интерфейсе. Сейчас возьмите эти сведения и ненадолго отложите их у себя в подсознании, поскольку нам предстоит заняться тем, как мы на самом деле собираемся брать и обрабатывать видео в режиме реального времени с целью добавления эффектов. Для этого нам потребуется «дотянуться руками» до никселей видео, изменить их для достижения желаемого нами эффекта, а затем как-то вернуть на экран.

Предусматривает ли API-интерфейс Video какой-нибудь способ обработки видео до того, как оно будет отображено? Нет. Однако он дает нам способ добраться до никселей, поэтому остается лишь выяснить, как их обработать и отобразить. Постойте, никсели? Отображение? Помните главу 7? Элемент `canvas`! Все верно, ранее мы кое-что упоминали об «особых взаимоотношениях», установившихся между элементами `video` и `canvas`. Давайте взглянем на один из способов, посредством которых элементы `video` и `canvas` могут работать сообща.

Подробности сенсационной новости, наконец, раскрыты! ↗

- 1 Видеопроигрыватель декодирует и воспроизводит видео «за кулисами».

- 2 Видео копируется кадр за кадром в (скрытый) буферный `canvas` и подвергается обработке.



Цвета оригинального видео

Обработка цветов видео с преобразованием их, например, в черно-белые, в буферном `canvas`.

✓ Затем мы копируем обработанный кадр из буферного `canvas` в `canvas` отображения.

← Если говорить кратко, то мы берем каждый кадр видео и преобразуем из цветного в черно-белый, после чего отображаем его.

- 3 После того как кадр обработан, он копируется в другой `canvas` отображения для просмотра.

Как обрабатывать видео с использованием временного буфера

У вас может возникнуть вопрос: почему мы используем два canvas — для обработки и отображения видео. Почему бы просто не найти способ обрабатывать видео при его декодировании?

Методика, которую мы здесь используем, является проверенной и позволяет минимизировать визуальные неполадки во время интенсивной обработки видео и изображений: она известна как *использование временного буфера*. Путем обработки кадров видео в буфере с последующим копированием их всех сразу в canvas отображения мы минимизируем визуальные проблемы.

Пошагово разберемся, как будет работать наша реализация временного буфера.

- 1 Браузер декодирует видео в серию кадров. Каждый кадр — это прямоугольник с пикселями, который представляет собой моментальный снимок видео в определенный момент времени.



Один кадр видео



- 2 По мере декодирования каждого из кадров мы копируем его в canvas, который выступает в роли временного буфера.



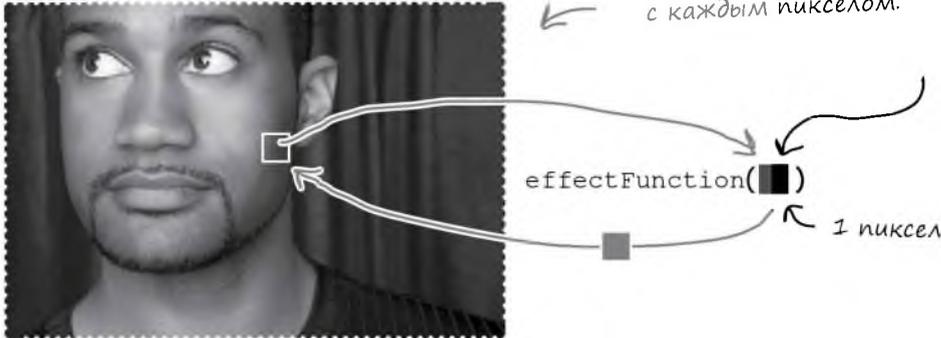
Целиком копируем кадр в canvas.



Временный буфер

- ③ Мы совершаем итерацию по временному буферу, пиксел за пикселом, передавая на обработку каждый пиксел нашей функции `effectFunction`.

После извлечения пикселных данных из `canvas` мы обращаемся к ним по одному пикселу за раз и обрабатываем их путем манипулирования значениями RGB в случае с каждым пикселом.



- ④ После того как все пикселы во временном буфере будут обработаны, мы скопируем их из временно-буферного `canvas` в `canvas` отображения.



- ⑤ Затем мы повторяем данный процесс для каждого кадра по мере того, как он декодируется объектом `video`.

Реализация временного буфера в canvas

Как вы уже знаете, чтобы реализовать временной буфер в canvas, нам необходимо два canvas: первый, в котором мы будем производить свои вычисления, и второй, где будут отображаться результаты. Создание этих двух canvas мы начнем в HTML-файле videobooth.html. Откройте данный файл, найдите там `<div> с id в виде "videoDiv"` и добавьте два элемента canvas под `<video>`:

```
<div id="videoDiv">
  <video id="video" width="720" height="480"></video>
  <canvas id="buffer" width="720" height="480"></canvas>
  <canvas id="display" width="720" height="480"></canvas>
</div>
```

Мы добавляем два элемента canvas, один из которых будет играть роль буфера, а другой станет использоваться для отображения.

Обратите внимание, что они будут точно такого же размера, как элемент video.

Позиционирование элементов video и canvas

У вас мог возникнуть вопрос насчет позиционирования данных элементов; мы собираемся расположить один из них поверх другого. Таким образом, внизу будет располагаться элемент video, поверх него — буферный canvas (bufferCanvas), поверх которого в свою очередь будет находиться canvas отображения (displayCanvas). Мы воспользуемся CSS, чтобы сделать это; хотя мы не слишком много говорим о CSS в данной книге, если вы откроете videobooth.css, то увидите, как осуществляется позиционирование этих трех элементов:

```
div#videoDiv {
  position: relative;
  width: 720px;
  height: 480px;
  top: 180px;
  left: 190px;
}
video {
  background-color: black;
}
div#videoDiv canvas {
  position: absolute;
  top: 0px;
  left: 0px;
}
```

Элемент `<div>` с id в виде `videoDiv` позиционируется относительно элемента, в котором он располагается (`<div>` с id в виде `console`), в 180 пикселях от верха и в 190 пикселях от левой границы, в результате чего он оказывается в центре элемента `<div>` с id в виде `console`. Для `width` и `height` мы задаем значения, аналогичные значениям `width` и `height` элемента `<video>` и двух элементов `<canvas>`.

`<video>` является первым элементом в `<div>` с id в виде `videoDiv`, поэтому он автоматически позиционируется вверху слева в `<div>`. Мы задаем `black` в качестве значения цвета фона, из-за чего при выводе видео в ТВ-формате Letter Box или Pillar Box полосы по его краям будут черного цвета.

К двум элементам `<canvas>` в `<div>` с id в виде `videoDiv` применяется абсолютное позиционирование относительно `videoDiv` (их родительского элемента), поэтому, позиционируя элементы `<canvas>` в 0 пикселях от верха и 0 пикселях от левого края, мы располагаем их в том же месте, что и элементы `<video>` и `videoDiv`.

Написание кода для обработки видео

У нас имеется элемент `video`, а также буфер, который является `canvas`, и еще один `canvas`, в котором будут отображаться итоговые видеокadres. Кроме того, эти элементы располагаются один поверх другого, в силу чего мы сможем увидеть только верхний `canvas` отображения, в котором будет содержаться видео с примененным эффектом. Для обработки видео мы воспользуемся событием `play` элемента `video`, которое инициируется при запуске воспроизведения видео. Добавьте данный код в конец обработчика `onload`:

```
video.addEventListener("play", processFrame, false);
```

← Когда видео начнет воспроизводиться, произойдет вызов функции `processFrame`.

В функции `processFrame` мы будем осуществлять обработку пикселей видео и перенос их в `canvas` для отображения. Начнем мы с того, что убедимся в наличии доступа ко всем нашим объектам DOM:

```
function processFrame() {
```

```
    var video = document.getElementById("video");
    if (video.paused || video.ended) {
        return;
    }
```

← Сначала мы извлекаем объект `video`...

← ...и проверяем, продолжается ли все еще воспроизведение видео. Если нет, то здесь нам больше нечего делать и мы просто выполняем `return`.

```
    var bufferCanvas = document.getElementById("buffer");
    var displayCanvas = document.getElementById("display");
    var buffer = bufferCanvas.getContext("2d");
    var display = displayCanvas.getContext("2d");
```

← Извлекаем ссылку на оба элемента `canvas` и ссылку на их `context`, которые нам потребуются.

```
}
```

Как создать буфер

Для создания буфера нам потребуется взять текущий видеокادر и скопировать его в буферный `canvas`. Как только он окажется в `canvas`, мы сможем обработать данные в этом кадре. Таким образом, чтобы создать буфер, мы делаем следующее (добавьте данный код в нижнюю часть `processFrame`):

Он принимает изображение и рисует его в `canvas` в позиции `x, y` с заданными шириной и высотой.

Помните метод `drawImage` объекта `context` из главы 7?

На этот раз мы извлекаем изображение из видео. Мы указываем `video` как источник, и `drawImage` получает один кадр соответствующего видео в качестве данных изображения.

```
buffer.drawImage(video, 0, 0, bufferCanvas.width, bufferCanvas.height);
var frame = buffer.getImageData(0, 0, bufferCanvas.width, bufferCanvas.height);
```

↑ Затем мы извлекаем данные изображения из `context` элемента `canvas` и сохраняем их в переменной `frame`, чтобы была возможность их обработать.

↑ Здесь мы просто говорим, что нам требуются все данные изображения в `canvas`.

Как обрабатывать буфер

Итак, мы получили кадр видеоданных, поэтому давайте обработаем его! Для обработки кадра мы будем совершать цикл по каждому пикселу в `frame.data` и извлекать значения цветов RGB, содержащиеся в каждом пикселе. Вообще-то пиксел имеет четыре значения — RGB и Alpha (непрозрачность), однако мы не станем использовать Alpha. Получив значения RGB, мы вызовем `effectFunction` (это функция, о которой мы говорили на с. 426, где просили вас отложить сведения о ней у себя в подсознании!) с использованием RGB-информации и кадра.

Добавьте данный код в нижнюю часть функции `processFrame`:

```
buffer.drawImage(video, 0, 0, bufferCanvas.width, displayCanvas.height);
var frame = buffer.getImageData(0, 0, bufferCanvas.width, displayCanvas.height);
```

```
var length = frame.data.length / 4;
```

← Сначала мы выясняем длину `frame.data`. Следует отметить, что данные располагаются в свойстве `frame` — `frame.data`, а `length` является свойством `frame.data`. На самом деле длина в четыре раза превышает размер `canvas`, поскольку каждый пиксел имеет четыре значения (RGBA).

```
for (var i = 0; i < length; i++) {
  var r = frame.data[i * 4 + 0];
  var g = frame.data[i * 4 + 1];
  var b = frame.data[i * 4 + 2];
  if (effectFunction) {
    effectFunction(i, r, g, b, frame.data);
  }
}
```

← Теперь мы совершаем цикл по данным и извлекаем значения RGB в случае с каждым пикселом. Каждый пиксел занимает четыре места в массиве, поэтому мы извлекаем `r` из первой позиции, `g` из второй и `b` из третьей.

← Затем мы вызываем `effectFunction` (если только значением соответствующей переменной не окажется `null`, что бывает, когда пользователь нажимает кнопку "normal"), передавая ей позицию пиксела, значения RGB и массив `frame.data`. Функция `effectFunction` обновит массив `frame.data`, используя новые значения пикселей, обработанные в соответствии с тем, какая функция для применения фильтра была присвоена `effectFunction`.

```
display.putImageData(frame, 0, 0);
```

На данном этапе обработка `frame.data` уже проведена, поэтому мы используем метод `putImageData` объекта `context` для размещения данных в `canvas` отображения. Данный метод принимает данные в `frame` и записывает их в `canvas` в указанную позицию `x, y`.

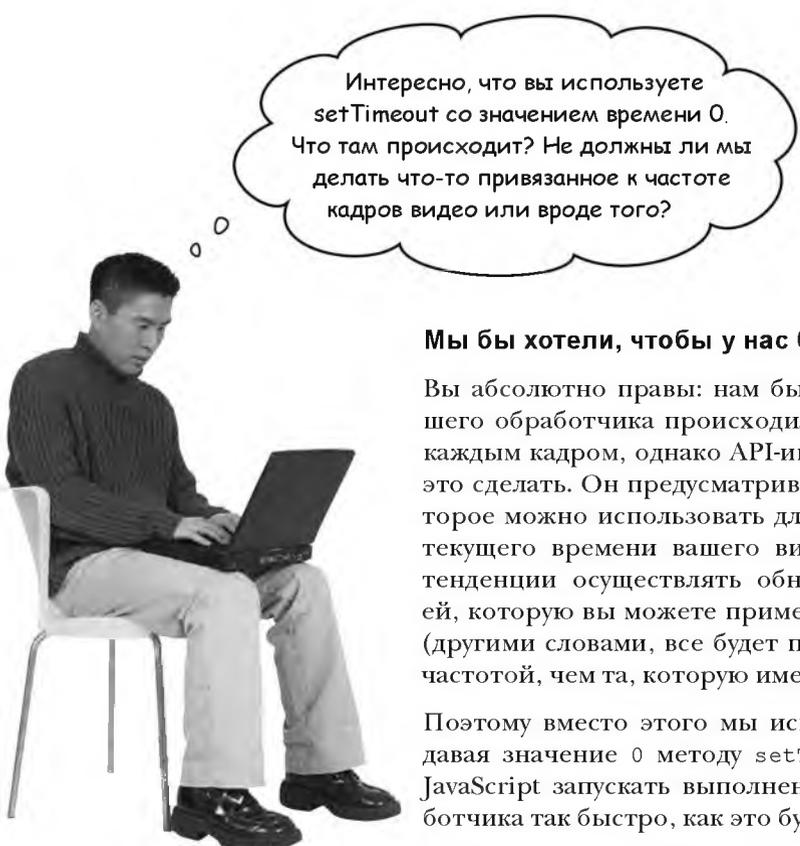
Мы обработали один кадр, что дальше?

Да, мы только что обработали один-единственный кадр и хотим продолжить обработку всех кадров, пока будет идти процесс воспроизведения видео. Мы можем использовать метод `setTimeout` и передать ему значение 0 миллисекунд, чтобы дать указание JavaScript снова вынудить `processFrame` так быстро, как это будет возможно. Вообще-то JavaScript не станет вынуждать функцию через 0 миллисекунд, однако обеспечит для нас самое быстрое следующее временное окно. Для этого вам нужно просто добавить данный код в нижнюю часть функции `processFrame`:

← `setTimeout` аналогичен `setInterval`, за исключением того, что выполняется только один раз через заданное время в миллисекундах.

```
setTimeout(processFrame, 0);
```

← Этим мы даем указание JavaScript снова выполнить `processFrame` так быстро, как это будет возможно!



Интересно, что вы используете `setTimeout` со значением времени 0. Что там происходит? Не должны ли мы делать что-то привязанное к частоте кадров видео или вроде того?

Мы бы хотели, чтобы у нас была такая возможность.

Вы абсолютно правы: нам бы хотелось, чтобы вызов нашего обработчика происходил но одному разу в случае с каждым кадром, однако API-интерфейс `Video` не дает нам это сделать. Он предусматривает событие `timeupdate`, которое можно использовать для обновления отображения текущего времени вашего видео, однако оно не имеет тенденции осуществлять обновление с той детализацией, которую вы можете применить при обработке кадров (другими словами, все будет происходить с более низкой частотой, чем та, которую имеет видео).

Поэтому вместо этого мы используем `setTimeout`. Передавая значение 0 методу `setTimeout`, вы даете указание JavaScript запускать выполнение соответствующего обработчика так быстро, как это будет возможно.

А может ли это происходить быстрее частоты кадров? Не лучше ли вычислить время ожидания близко к требуемому в случае с определенной частотой кадров? Это можно сделать, однако вряд ли обработчик действительно станет выполняться строго с частотой, аналогичной частоте кадров вашего видео, но поэтому 0 является подходящим приближительным значением. Если вы стремитесь увеличить производительность своего приложения, то, конечно же, всегда сможете прибегнуть к профилированию и выяснить оптимальные значения. Но до тех пор, пока у нас не появится более специфический API-интерфейс, дело будет обстоять именно так.

Займемся написанием эффектов

Наконец, у нас есть все необходимое для написания видеоэффектов: мы извлекаем каждый кадр по мере его поступления, обращаемся к `frame.data` пиксел за пикселем и отправляем пиксели нашей функции, используемой для применения определенного фильтра. Давайте взглянем на фильтр `Film noir` (который в нашем варианте просто является причудливым названием черно-белого фильтра):

Передаем функции, используемой для применения фильтра, позицию пиксела...

...значения пикселей красного, зеленого и синего цвета...

...а также ссылку на массив `frame.data` в `canvas`.

>>> является поразрядным оператором, который сдвигает биты в числовом значении с целью его модификации. Более подробно о нем вы сможете узнать из справочника по JavaScript.

```
function noir(pos, r, g, b, data) {
  var brightness = (3*r + 4*g + b) >>> 3;
  if (brightness < 0) brightness = 0;
  data[pos * 4 + 0] = brightness;
  data[pos * 4 + 1] = brightness;
  data[pos * 4 + 2] = brightness;
}
```

Помните, что эта функция будет вызываться по одному разу в случае с каждым пикселем в видеокадре!

Это будет иметь эффект задания для пикселя значения, обеспечивающего оттенки серого, которые соответствуют общей яркости пиксела.

Таким образом, первое, что мы делаем, — это вычисляем значение `brightness` для данного пиксела, взяв за основу все его компоненты (`r`, `b` и `g`).

А затем присваиваем каждому компоненту в изображении `canvas` данное значение `brightness`.

Тест-драйв фильтра `Film noir`



Добавьте приведенную выше функцию в файл `videobooth.js`, а затем перезагрузите свою страницу. Как только видео начнет проигрываться, нажмите кнопку `FILM NOIR`, и вы увидите его в мрачном черно-белом варианте. А теперь снова нажмите `NORMAL`. Ненлохо, не правда ли? И все это на JavaScript, в режиме реального времени!

Если задуматься, получается в некотором роде удивительная вещь.



Возьми в руку карандаш



Данная книга не посвящена обработке видео и эффектам, однако это, несомненно, увлекательная тема. Ниже приведены функции `western` и `scifi`, позволяющие применять соответствующие эффекты. Прочитайте код и напишите справа, как работает каждая из них. Кроме того, мы добавили к ним еще одну функцию — что она делает?

```
function western(pos, r, g, b, data) {
    var brightness = (3*r + 4*g + b) >>> 3;
    data[pos * 4 + 0] = brightness+40;
    data[pos * 4 + 1] = brightness+20;
    data[pos * 4 + 2] = brightness-20;
}

function scifi(pos, r, g, b, data) {
    var offset = pos * 4;
    data[offset] = Math.round(255 - r) ;
    data[offset+1] = Math.round(255 - g) ;
    data[offset+2] = Math.round(255 - b) ;
}

function bwcartoon(pos, r, g, b, outputData) {
    var offset = pos * 4;
    if( outputData[offset] < 120 ) {
        outputData[offset] = 80;
        outputData[++offset] = 80;
        outputData[++offset] = 80;
    } else {
        outputData[offset] = 255;
        outputData[++offset] = 255;
        outputData[++offset] = 255;
    }
    outputData[++offset] = 255;
    ++offset;
}
```

Большой тест-драйв



Вот и все! Мы завершили работу над кодом, и он готов к отправке в компанию **Starring You Video**. Дважды проверьте весь код, который вы нанесли, сохраните его и загрузите `videobooth.html`. Затем развлекайтесь, но играя со своим новым приложением!



В ЛАБОРАТОРИИ

Мы лишь поверхностно затронули тему обработки видео. Уверены, что вы сможете придумать более креативные эффекты, чем те, которые были продемонстрированы. Придумайте несколько таких эффектов, реализуйте и задокументируйте их здесь.

Удалось ли вам изобрести и реализовать что-то по-настоящему классное? Расскажите нам об этом, посетив wickedlysmart.com, и мы продемонстрируем плоды вашего труда другим читателям!



Свои идеи напишите здесь!

Функция *bwcartoon* позволяет делать одну из многих забавных вещей, которые возможны при использовании функций для применения эффектов.



Я знаю, что мы почти подошли к концу главы, однако все же хочу спросить: мы загружали видео из локального файла, а что изменится, если мое видео будет располагаться в Интернете?



Вам лишь потребуется использовать веб-URL.

Вы можете подставить веб-URL вместо любого из источников, которые мы определяли локально. Например:

```
<video src="http://wickedlysmart.com/myvideo.mp4">
```

Имейте в виду, что вероятность возникновения проблем более высока, когда ваше видео доставляется через Интернет (чуть позже мы поговорим о том, как с ними справляться). Кроме того, битрейт вашего видео начинает приобретать намного большее значение, когда вы доставляете его в браузер или на мобильное устройство по Сети. Как и в случае с форматами видео, если вы решите пойти этим путем, общайтесь со специалистами и занимайтесь самообразованием.

Отлично, и еще один вопрос: есть ли разница между тем, что мы делаем, и потоковым видео?



Да, есть, причем большая разница.

Понятие «потоковая передача» используется так же часто, как понятие «ксерокс» или «носовой платок», — это общий термин, означающий доставку видео из Интернета в ваш браузер. «Прогрессивное видео» и «потоковое видео» на самом деле являются техническими терминами. В данной книге мы используем прогрессивное видео, а это означает, что когда мы извлекаем видео (локально либо по Сети), мы извлекаем соответствующий файл с использованием HTTP точно так же, как HTML-файл или изображение, и пытаемся декодировать и воспроизвести данный файл после его извлечения. Потоковое видео доставляется с использованием протокола, который тонко настроен для доставки видео оптимальным путем (возможно, даже с изменением битрейта видео по мере того, как снижается или растет нагрузка на канал).

Потоковое видео, вероятно, способно обеспечить для ваших пользователей более качественное взаимодействие (и это правда) и, возможно, является более эффективным в плане нагрузки на канал подключения ваших пользователей, а также на ваш канал (что тоже правда). В дополнение ко всему этому потоковое видео облегчает защиту содержимого видео, если вам потребуется обеспечить безопасность.



А существует ли стандарт для потокового видео в случае HTML5?

Нет.

В HTML5 нет стандарта для потокового видео. Дело в том, что проблема заключается не в HTML5, а в отсутствии поддерживаемого стандарта для потокового видео; вместе с тем существует масса собственных вариантов. Почему? Тому есть множество причин: это и деньги, которые можно заработать на потоковом видео, и тот факт, что многие люди, занятые в сфере программного обеспечения с открытым исходным кодом, не желают работать над протоколом, который может быть использован для DRM и прочих технологий защиты. Как и с форматами видео, ситуация с потоковым видео непростая.



Так что же делать, если мне потребуется обеспечить передачу потокового видео?

Существует ряд решений.

Технологии потокового видео находят массу разумных применений, и если вы располагаете обширной аудиторией либо содержимым, которое, как вам кажется, нуждается в защите, обратите внимание на HTTP Live Streaming от компании Apple, Smooth Streaming от Microsoft и HTTP Dynamic Streaming от Adobe, которые станут хорошим отправным пунктом.

Есть и хорошие новости: комитет стандартов начинает обращать пристальное внимание на потоковое видео на основе HTTP, поэтому следите за разработками в данной области.

Если бы это был совершенный мир...

Однако он не является таковым: мы сталкиваемся с ненриятными сетевыми проблемами, несовместимыми устройствами и операционными системами и возрастающей вероятностью того, что астероиды могут врезаться в Землю. С последним мы ничего не можем поделать, а что касается первых двух, то знание того, что у вас возникла ошибка, — это уже поделла, поскольку вы тогда, но крайней мере, сможете что-нибудь предпринять.

Объект `video` включает событие `error`, инициирующееся по ряду причин, которые можно отыскать в свойстве `video.error`, точнее говоря, в свойстве `video.error.code`. Давайте взглянем, какие типы ошибок мы можем обнаруживать.



Ошибки

`MEDIA_ERR_ABORTED=1`

Генерируется каждый раз, когда процесс передачи видео по сети отменяется браузером (возможно, по требованию пользователя).

`MEDIA_ERR_NETWORK=2`

Генерируется каждый раз, когда передача видео по сети прерывается вследствие ошибки сети.

`MEDIA_ERR_DECODE=3`

Генерируется каждый раз, когда декодирование видео оказывается неудачным. Это может случаться из-за того, что видео было закодировано с особенностями, которые не поддерживаются браузером, либо видеофайл оказался поврежден.

`MEDIA_ERR_SRC_NOT_SUPPORTED=4`

Генерируется, когда указанный источник видео не поддерживается из-за неправильного URL-адреса либо тип источника не может быть декодирован браузером.

Каждый тип ошибки также обладает ассоциированным номером, который представляет собой код ошибки, сгенерированный событием `error`, о чем мы поговорим через несколько мгновений...

Как использовать события error

Обращение с ошибками является непростым занятием, зависящим от вашего приложения, а также от того, что окажется подходящим для вашего приложения и ваших пользователей. По крайней мере, мы можем помочь вам начать и указать правильное направление. Возьмем приложение Webville TV и наделим его способностью узнавать о том, что оно столкнулось с ошибкой, — и если оно с ней столкнется, то выведет для аудитории сообщение `please stand by` (не меняйте настройки).

Мы хотим получать уведомление, когда имеет место сообщение об ошибке, поэтому нам потребуется добавить слушатель событий `error`. Вот как мы сделаем это (добавьте данный код в обработчик `onload` в `webville.js`):

```
video.addEventListener("error", errorHandler, false);
```

Теперь нам нужно написать функцию `errorHandler`, которая будет проводить проверку на предмет ошибок. Если она обнаружит ошибку, то разместит наше изображение с надписью `please stand by` (не меняйте настройки) в области отображения видео, сделав его постерным:

```
function errorHandler() {
    var video = document.getElementById("video");
    if (video.error) {
        video.poster = "images/technicaldifficulties.jpg";
        alert(video.error.code);
    }
}
```

↖ Когда произойдет ошибка, будет вызвана функция `errorHandler`.

↙ Если происходит вызов обработчика, мы удостоверимся в том, что имела место ошибка, для чего обращаемся к свойству `video.error`, а затем размещаем постерное изображение в области отображения видео.

↙ Вы можете опционально добавить данную строку, чтобы иметь возможность увидеть код ошибки (см. предыдущую страницу, где рассказывается о целочисленных значениях, которые располагаются в свойстве `code`).

Краш-тест!



Существует много причин, по которым воспроизведение видео может оказаться неудачным, а для тестирования данного кода вы специально сделаете так, чтобы это произошло. Вот ряд советов о том, как это можно сделать:

- не пробуйте отключать свою сеть в разные моменты воспроизведения видео;
- введите в проигрывателе неправильный URL-адрес;
- не пробуйте воспроизвести в проигрывателе видео, которое, как вам будет известно заранее, он не сможет декодировать;
- используйте программное обеспечение для снижения пропускной способности своего канала (оно существует, нужно лишь поискать).



Итак, наберите данный код и проведите тестирование. Вы сможете сопоставить целочисленные значения в диалоговых окнах `alert` с реальным кодом, взглянув на коды ошибок на с. 441.

Куда вы сможете отправиться отсюда?

Задумайтесь над всем тем, что вы уже умеете делать с помощью HTML-разметки, элемента video и, конечно же, canvas... не говоря уже о веб-службах, API-интерфейсе Geolocation. Здорово! Мы с вами провели классную обработку видео с использованием canvas, однако вы сможете применить свои знания о canvas и в случае с элементом video. Ниже приведены наши идеи, а вы добавьте к ним свои. И похвалите себя от нашего имени, поскольку вы это заслужили!

Создавайте основанное на времени содержимое, которое отображается только в определенные промежутки времени при воспроизведении видео.

Используйте веб-службы для обеспечения соответствующей контексту информации для своего видео.

Создавайте собственные экранные элементы управления или программные указатели.

Используйте JSONP для добавления интерактивности!

Обеспечивайте режим просмотра «картинка в картинке».

Используйте свои страницы для размещения соответствующей рекламы.

Применяйте все API-интерфейсы для работы с графикой и текстом, которыми вы умеете пользоваться, начиная с Canvas, и накладывайте изображения прямо поверх видео.

КЛЮЧЕВЫЕ МОМЕНТЫ

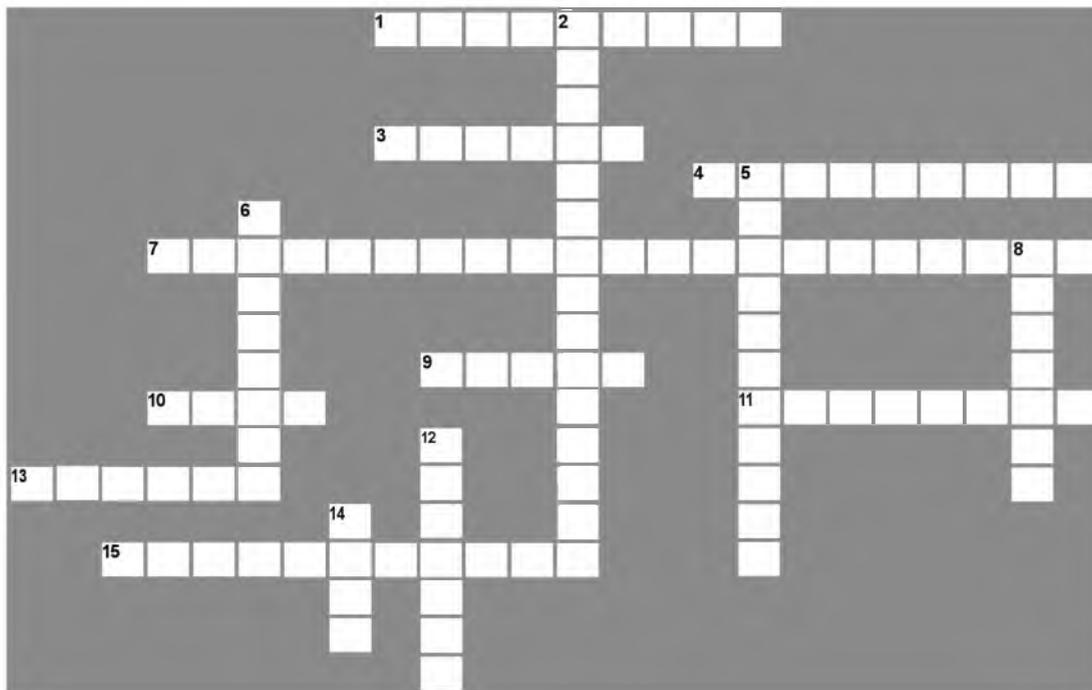


- Вы можете воспроизводить видео путем использования элемента `<video>` в сочетании с несколькими простыми атрибутами.
- Атрибут `autoplay` обеспечивает запуск воспроизведения видео по окончании загрузки страницы, однако его следует использовать, только когда он уместен.
- Благодаря наличию атрибута `controls` браузер предоставит пользователю набор элементов управления воспроизведением видео.
- Внешний вид элементов управления различается в разных браузерах.
- Посредством атрибута `poster` вы можете предусмотреть наличие своего собственного постерного изображения.
- Атрибут `src` содержит URL-адрес видео для воспроизведения.
- Для форматов видео и аудио существует множество «стандартов».
- Широко применяются три формата: WebM, MP4/H.264 и Ogg/Theora.
- Изучайте свою аудиторию, чтобы понять, какие форматы видео вам потребуется обеспечить.
- Используйте тег `<source>` для определения альтернативных форматов видео.
- Используйте полностью определенные типы в своем теге `<source>`, чтобы сэкономить силы и время браузера.
- Вы сможете сохранить поддержку других видео-фреймворков, например Flash, добавив резервный тег `<object>` в элемент `video`.
- Элемент `video` предусматривает богатый набор свойств, методов и событий.
- `video` поддерживает методы и свойства `play`, `pause`, `load`, `loop` и `mute` для прямого контроля над воспроизведением видео.
- Событие `ended` может использоваться для того, чтобы дать знать, когда воспроизведение видео закончится (например, для реализации плейлиста).
- Вы можете программно спрашивать у объекта `video`, сумеет ли он воспроизвести видео в том или ином формате, используя `canPlayType`.
- Метод `canPlayType` возвращает пустую строку (поддержка формата отсутствует), "maybe" (если ему, возможно, удастся воспроизвести видео в определенном формате) и "probably" (если он будет уверен в том, что не сможет воспроизвести видео в определенном формате).
- `canvas` может использоваться в качестве поверхности отображения для `video` с целью реализации пользовательских элементов управления и прочих эффектов в случае с видео.
- Вы можете использовать временный буфер для обработки видео, прежде чем оно будет скопировано в `displayCanvas`.
- Вы можете задать обработчик `setTimeout` для видео-кадров; несмотря на то что он не привязывается напрямую к каждому кадру видео, это наилучший метод, который имеется на данный момент.
- Вы можете использовать URL-адрес как источник видео для воспроизведения видеофайлов, размещаемых в Сети.
- Некоторые браузеры задействуют политику одного источника в случае с видео, так что вам придется загружать свое видео из того же источника, откуда исходит ваша соответствующая страница.
- В случае с видео возникновение ошибок возможно всегда, особенно когда задействуется Сеть.
- Событие `error` может использоваться для уведомления обработчика, когда при извлечении, декодировании или воспроизведении случаются ошибки.
- Элемент `video` полагается на прогрессивно загружаемое видео. В настоящее время нет HTML5-стандарта для потоковой передачи видео, однако комитет стандартов начинает обращать внимание на решения для передачи потокового видео на основе HTTP.
- На текущий момент не существует стандартного способа защиты видео, доставляемого посредством элемента `video`.



HTML5-Кроссворд

Прежде чем вы откинетесь на спинку стула и захотите еще посмотреть Webville TV, разгадайте небольшой кроссворд для закрепления изученного материала.



По горизонтали

1. Для обеспечения набора видео в форматах, из которых можно будет выбрать подходящий, используйте сразу _____ элементов `source`.
3. Используется для отображения обработанного видео.
4. Тип буфера, который мы использовали в случае с `canvas`.
7. Тип доставки, который элемент `video` использует в случае с видео.
9. Когда воспроизведение видео заканчивается, инициируется данное событие.
10. Свойство для воспроизведения видео в зацикленном режиме.
11. Запускает воспроизведение видео так быстро, как это только будет возможно.
13. Аудиокодек с открытым исходным кодом.
15. Я могу воспроизвести данный тип видео, а ты?

По вертикали

2. Мы смотрели _____ фильмы 1950-х годов.
5. Внешний вид элементов управления _____ в разных браузерах.
6. Задействуйте атрибут _____, если вам требуются встроенные средства управления видео.
8. Что следует делать, если астероид собирается врезаться в Землю?
12. Клинту Иствуду понравился бы тип видеозффекта, который обеспечивается нажатием данной кнопки.
14. То, что мы обрабатывали при каждом вызове `setTimeout`.

Возьми в руку карандаш



Решение

Данная книга не посвящена обработке видео и эффектам, однако это, несомненно, увлекательная тема. Ниже приведены функции `western` и `scifi`, позволяющие применять соответствующие эффекты. Просмотрите код и напишите справа, как работает каждая из них. Кроме того, мы добавили к ним еще одну функцию — что она делает? Вот наше решение этого упражнения.

```
function western(pos, r, g, b, data) {
    var brightness = (3*r + 4*g + b) >>> 3;
    data[pos * 4 + 0] = brightness+40;
    data[pos * 4 + 1] = brightness+20;
    data[pos * 4 + 2] = brightness-20;
}
```

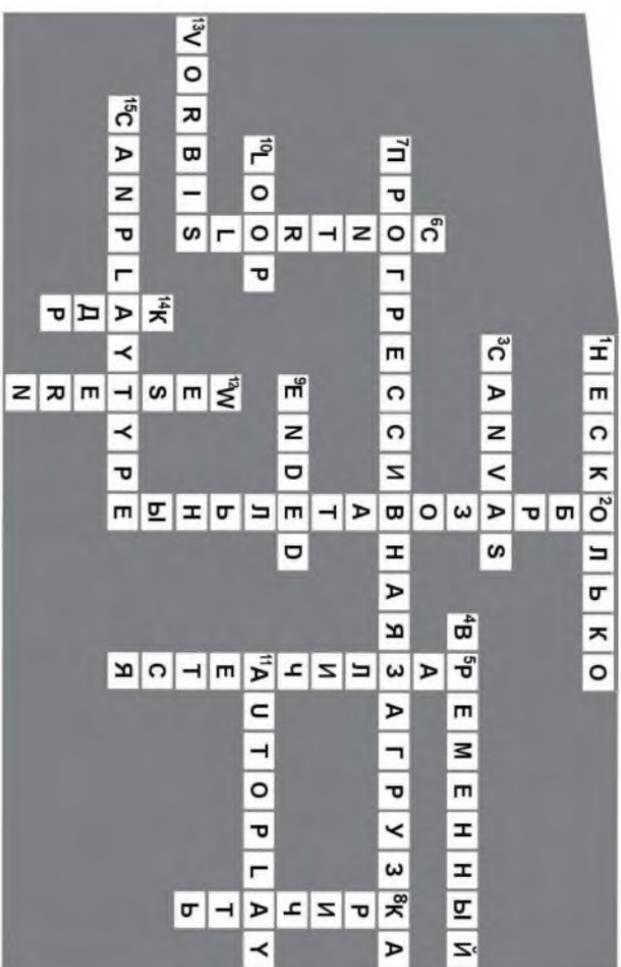
Функция для применения фильтра `western` усиливает компоненты `r` (красный) и `g` (зеленый) пиксела, одновременно ослабляя компонент `b` (синий), чтобы придать видео коричневатый оттенок.

```
function scifi(pos, r, g, b, data) {
    var offset = pos * 4;
    data[offset] = Math.round(255 - r);
    data[offset+1] = Math.round(255 - g);
    data[offset+2] = Math.round(255 - b);
}
```

Функция для применения фильтра `scifi` изменяет в обратную сторону количество компонентов `RGB` каждого пиксела. Таким образом, если пиксел содержал много красного, то теперь его будет мало. Если же пиксел включал мало зеленого, то теперь его будет много.

```
function bwcartoon(pos, r, g, b, outputData) {
    var offset = pos * 4;
    if( outputData[offset] < 120 ) {
        outputData[offset] = 80;
        outputData[++offset] = 80;
        outputData[++offset] = 80;
    } else {
        outputData[offset] = 255;
        outputData[++offset] = 255;
        outputData[++offset] = 255;
    }
    outputData[++offset] = 255;
    ++offset;
}
```

Функция для применения фильтра `bwcartoon` превращает каждый пиксел с компонентом `r` (красный), меньшим 120 (из 255), в черный, а все остальные пикселы — в белые, придавая видео причудливый мультяшный черно-белый вид.



телевидение для нового поколения

РАЗВЕДКА ПОДДЕРЖКИ ВИДЕО РЕШЕНИЕ

Устройства под управлением операционных систем iOS и Android (среды прочих)

СОВЕРШЕННО СЕКРЕТНО

Видео	Браузер								
H.264	Safari	✓			✓		✓	Internet Explorer 9 и выше	
WebM	Chrome	частично	✓					Internet Explorer 8	
Ogg Theora	Firefox		✓					Internet Explorer 7 или ниже	
	Mobile Webkit				✓				
	Opera						✓		



Будьте острожными!

Все это будет быстро меняться! Поэтому заглядывайте в Интернет для получения самой свежей информации о поддержке браузером данных возможностей.



ИМЛ5-КроссВорд. Решение

NO: ВИДЕО

9 сохраняем данные локально

API-интерфейс Web Storage

Я уже устала от этого тесного шкафа и от того, что мне приходится носить один и тот же брючный костюм. Благодаря HTML5 у меня будет достаточно места для одежды, я смогу каждый день надевать новый костюм!



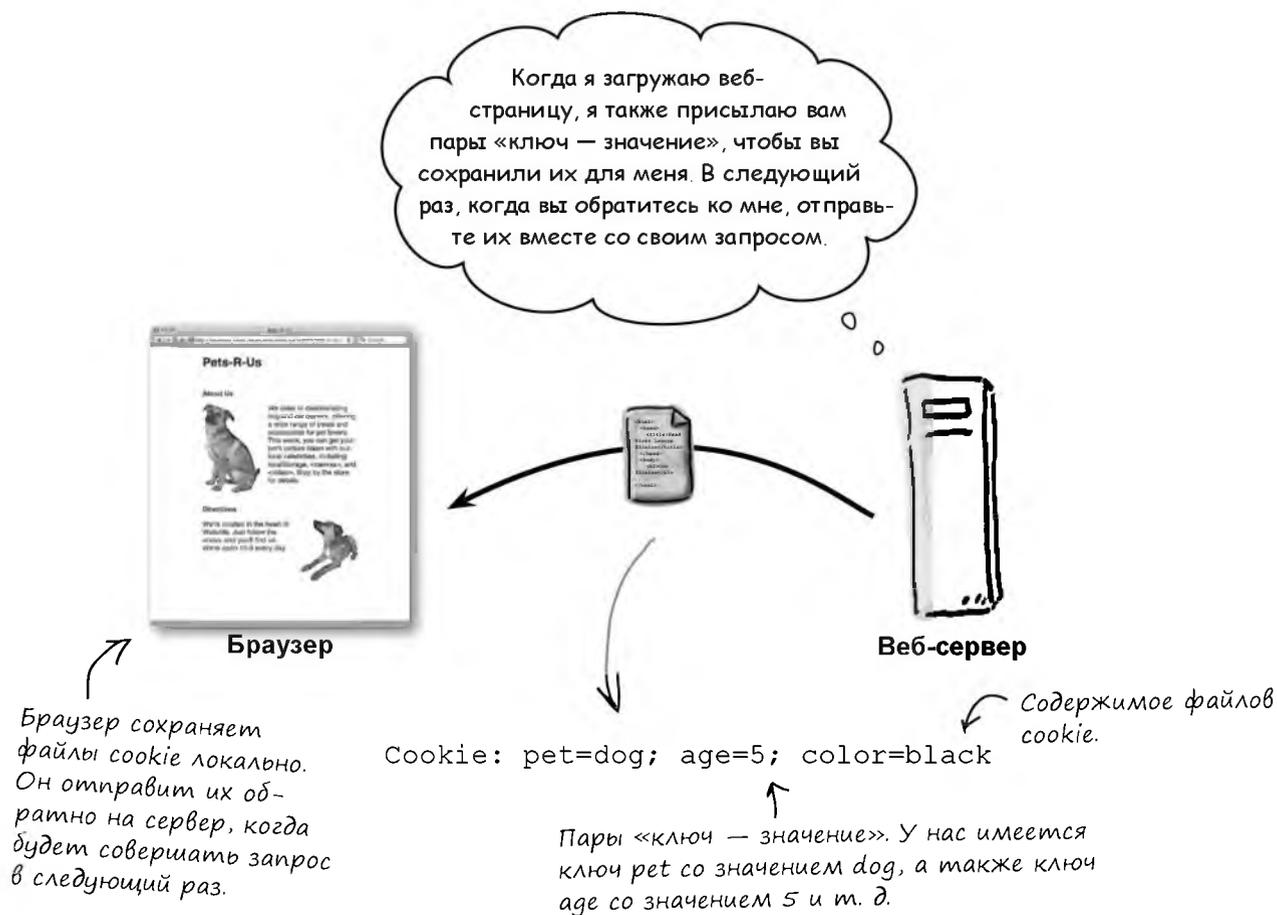
Устали от того, что клиентские данные приходится втискивать в тесные шкафы файлы cookie? В 1990-е годы такой проблемы не было, однако сейчас при использовании веб-приложений запросы значительно возросли. Как бы вы отнеслись к тому, если бы мы сказали, что у вас есть возможность выделять по 5 Мбайт на браузер каждого пользователя? Вы, вероятно, подумали бы, что мы шутим. Однако не стоит быть скептичными, поскольку API-интерфейс HTML5 Web Storage как раз позволят это делать! Из данной главы вы узнаете обо всем, что необходимо для сохранения объектов локально на устройстве пользователя и использования их в работе ваших веб-приложений.

Как работает браузерное хранилище (1995–2010)

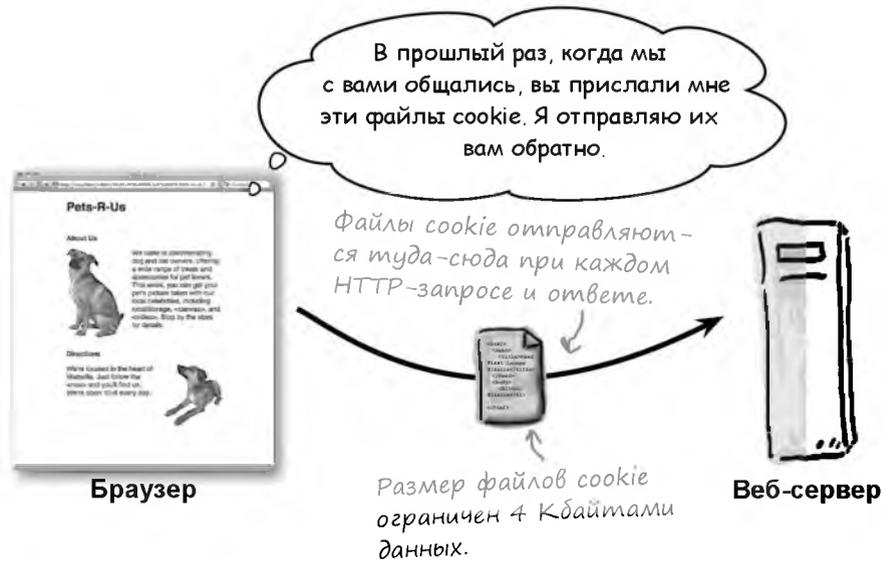
Создаете электронную корзину? Вам требуется сохранять установки пользователей на своем сайте? Или просто нужно прирывать данные, которые должны быть ассоциированы с каждым из пользователей? Именно здесь вступает в дело браузерное хранилище. Это хранилище позволяет сохранять на постоянной основе данные, которые мы сможем использовать в работе своих веб-приложений.

До настоящего времени существовал только один оптимальный вариант для сохранения информации в браузере — файлы cookie. Давайте посмотрим, как они работают.

- 1 Когда ваш браузер извлекает веб-страницу, скажем, из `pets-R-us.com`, сервер может присылать файлы cookie вместе со своим ответом. Файлы cookie содержат одну или более пар «ключ — значение»:



- 1 Когда браузер будет совершать запрос `pets-R-us.com` в следующий раз, он отправит вместе с ним все файлы cookie, которые были присланы ранее.



- 1 Сервер затем сможет использовать эти файлы cookie для персонализации взаимодействия. В данном случае это будет выражаться в продвижении соответствующих элементов, касающихся определенного пользователя, но существует и масса других способов использования файлов cookie.



МОЗГОВОЙ ШТУРМ

Файлы cookie уже давно у нас под рукой, однако вам, возможно, удастся придумать способ усовершенствовать их.

Отметьте флажками те пункты из приведенных ниже, которые, по вашему мнению, делают использование файлов cookie проблематичным.

- Для работы доступно лишь 4 Кбайта, а моему приложению требуется хранилище большего размера.
- Отправка файлов cookie туда-сюда каждый раз очень неэффективна, особенно при использовании мобильных устройств в сочетании с медленным каналом связи.
- Похоже, что они являются удобным способом передачи вирусов и прочего вредоносного программного обеспечения в мой браузер.
- Мне доводилось слышать, что способ, посредством которого пары «ключ — значение» включаются в HTTP-запрос, сложно реализовать в коде.
- А разве мы потенциально не отправляем личные данные туда-сюда каждый раз, когда совершаем запрос?
- Не похоже, что они хорошо сочетаются с нашей разработкой, связанной с клиентской стороной. Мне кажется, они предполагают, что все будет происходить на сервере.

Несмотря на то что в прессе утверждается обраться к файлам cookie сегодня довольно сложно, я думаю, что они и в будущем будут использоваться в качестве удобного способа передачи данных в мой браузер.



Я надеюсь, что HTML5 предусматривает простой API-интерфейс клиентской стороны для сохранения данных, который обеспечивает их хранение на постоянной основе в браузере, предлагает большой объем хранилища и передачу данных на сервер только в том случае, если я этого захочу.

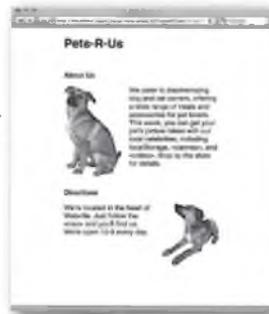
Как работает Web Storage HTML5



HTML5 предусматривает изящный и простой API-интерфейс JavaScript в браузере для сохранения устойчивых пар «ключ — значение». Кроме того, вы не будете ограничены скудными 4 Кбайтами хранилища; в настоящее время все браузеры готовы предоставить вам хранилище размером от 5 до 10 Мбайт, причем в браузере каждого пользователя. HTML5-технология localStorage также создавалась с учетом веб-приложений (и мобильных приложений!); она означает, что ваше приложение сможет сохранять данные в браузере, чтобы снизить объем «общения» с сервером. Давайте посмотрим, как все это работает (а затем попробуем в данный API-интерфейс с головой).

1 Страница может сохранить одну или более пар «ключ — значение» в локальном хранилище (localStorage) браузера.

Используя соответствующий API-интерфейс, мы можем записать пару «ключ — значение» в локальное хранилище.



Браузер

ключ: "pet"
значение "dog"

Пара «ключ — значение»



localStorage

Каждый современный браузер обеспечивает локальное хранилище размером 5 Мбайт (и более!) для каждого домена.

Хранилище обеспечивает сохранение данных на постоянной основе, даже если вы закроете окно браузера или вообще завершите его работу.

1 Позднее можно будет использовать ключ для извлечения соответствующего значения.

А можно мне значение для ключа pet?

Имея ключ, мы сможем также извлечь значение из локального хранилища.



Браузер

Конечно можно, этим значением будет dog.



localStorage

Как и в случае с файлами cookies, ваша страница сможет сохранять и извлекать лишь те элементы, которые были сгенерированы страницей и исходными из того же домена. Подробно об этом поговорим позже.

ПРИМЕЧАНИЕ: сервер будет продолжать загружать ваши страницы, а вы при этом даже сможете отправить на него немного данных, расположенных в вашем локальном хранилище, для проведения вычислений на стороне сервера. Однако с деталями локального хранилища будет разбираться клиент, а не сервер (как это обычно бывает в случае с файлами cookie).

Заметка для себя...

Вам требуется система, чтобы управиться с делами? Сложно улучшить старую добрую систему заметок-папюмипапий (более известную как *клеякие заметки*). Вы знаете, как она работает: вы записываете на листке бумаги пункт «это нужно сделать», приклеиваете его куда-нибудь и после выполнения записанного выбрасываете эту клейкую бумажку в мусорное ведро.

Попробуем создать такую систему с использованием HTML. Нам потребуется способ сохранять все эти клейкие заметки, поэтому нам будет нужен сервер, а также файлы cookie... Постойте-ка, включите задний ход, ведь мы можем сделать все это с помощью API-интерфейса HTML5 Web Storage!

API-интерфейс Web Storage прост и увлекателен, работать с ним — одно удовольствие. Уверяем вас!



Не будем валять дурака, а сразу приступим к использованию локального хранилища. Для этого вам потребуется создать простую HTML-страницу с базовыми элементами: `<head>`, `<body>` и `<script>` (либо вы можете просто воспользоваться файлом `notetoself.html` в примере кода). Следуйте за нами и наберите в своем элементе `<script>` приведенный далее код (ввод кода способствует запоминанию материала):

- 1 На клейкой заметке присутствует лишь текст, который вы написали, не так ли? Поэтому давайте сначала сохраним эту заметку (sticky) для "Pick up dry cleaning" ("Забрать вещи из химчистки"):

API-интерфейс Web Storage будет доступен посредством объекта `localStorage`. Вы увидите, что браузер уже определил его за вас. Применяя его, вы используете основную систему локального хранения.

Метод `setItem` принимает две строки в качестве аргументов, которые играют роль пары «ключ — значение».

Мы начинаем с простого, однако прежде, чем вы осознаете это, у нас уже будет целое работающее приложение `Stickies`.

Вы сможете сохранять только элементы типа string. Напрямую сохранять числа или объекты нельзя (однако мы вскоре найдем способ обойти это ограничение).

```
localStorage.setItem("sticky_0", "Pick up dry cleaning");
```

Чтобы сохранить что-то, мы используем метод `setItem`.

Первый строковый аргумент является ключом, под которым сохраняется элемент. Вы можете присваивать ему любое имя по своему усмотрению, поскольку это строка.

Второй строковый аргумент является значением, которое вы хотите сохранить в локальном хранилище.

- 1 Предыдущее действие было довольно простым. Давайте добавим второй элемент в локальное хранилище:

```
localStorage.setItem("sticky_1", "Cancel cable tv, who needs it now?");
```

Еще один ключ. Как мы уже говорили, у вас есть возможность использовать для ключа любое имя, поскольку он является строкой, однако вы сможете сохранять только одно значение на каждый ключ.

Значение, которое будет соответствовать нашему новому ключу.

- 1 Теперь, когда у нас есть два значения, надежно сохраненные в локальном хранилище браузера, вы можете воспользоваться одним из ключей для извлечения соответствующего значения из localStorage. Например:

Извлекаем значение, ассоциированное с ключом «sticky_0», из локального хранилища...

...и присваиваем его переменной sticky.

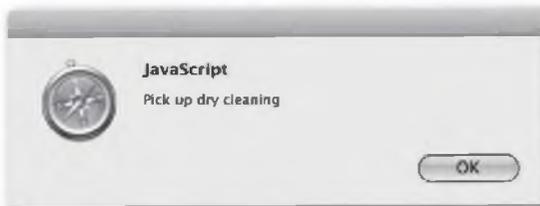
```
var sticky = localStorage.getItem("sticky_0");
```

```
alert(sticky);
```

Чтобы все стало еще интереснее, используем функцию alert для вывода на экран значения sticky.

Время тест-драйва!

Убедитесь, что вы напечатали весь приведенный чуть выше код в своем элементе <script>, и загрузите его в своем браузере. Вот результат нашего тест-драйва:



Это наше диалоговое JavaScript-окно alert со значением sticky_0 в качестве предупреждающего сообщения.

Здорово, что данное значение было сохранено и извлечено из localStorage браузера! Вы можете завершить работу своего браузера, отправиться в отпуск на Фиджи, затем вернуться, а оно по-прежнему будет дожидаться вас там.

Ладно-ладно, мы согласны, что пример мог бы быть более захватывающим... Однако оставайтесь с нами, поскольку скоро так и будет...

Это было здорово,
но не могли бы мы с вами
пройтись по коду подробно?
Я не уверена, что разобралась
в происшедшем на 100 %.



Разумеется. Кратко о том, как обстоят дела: браузер обеспечивает для вас локальное хранилище — область на вашем компьютере, в вашем браузере, — которое страница может использовать для сохранения пар «ключ — значение». Вы создаете несколько пар «ключ — значение», сохраняете их, используя API-интерфейс localStorage, а затем извлекаете одну из них для использования в своем приложении. Несмотря на то что это, может, и не самый захватывающий пример, существует масса интересных вещей, которые вы сможете сделать, используя хранилище в браузере каждого из пользователей (и мы уверены, что вы сможете придумать по крайней мере несколько таких ситуаций).

А теперь подробно разберемся в том, что только что произошло.

- 1 Во-первых, помните, что каждый браузер располагает локальным хранилищем, которое вы можете использовать для сохранения пар «ключ — значение».



Браузер



localStorage

← Каждый современный браузер располагает локальным хранилищем «за кулисами», готовым к использованию вами для сохранения пар «ключ — значение».

- Используя данное локальное хранилище, вы можете взять ключ и значение (которые будут представлять собой строки) и сохранить их там.

```
localStorage.setItem("sticky_0", "Pick up dry cleaning");
```

Используем метод `setItem` для сохранения пары «ключ — значение». Ключом будет `"sticky_0"`, а значением — `"Pick up dry cleaning"` ("Забрать вещи из химчистки").

Пара «ключ — значение», сохраненная посредством вызова метода `setItem`.



localStorage

- Затем мы снова вызвали `setItem` и сохранили вторую пару «ключ — значение», куда на этот раз входит ключ `"sticky_1"` и значение `"Cancel cable tv, who needs it now?"` ("Отказаться от подписки на кабельное ТВ, ведь кому оно теперь нужно?").

```
localStorage.setItem("sticky_1", "Cancel cable tv, who needs it now?");
```

Теперь у нас есть два уникальных значения, сохраненных под двумя уникальными ключами.



localStorage

- И когда мы вызовем метод `getItem`, указав при этом ключ `"sticky_0"`, он вернет значение, имеющееся в соответствующей паре «ключ — значение».

```
localStorage.getItem("sticky_0");
```

возвращает
↓
"Pick up dry cleaning"

`getItem` находит элемент с ключом `"sticky_0"` (при наличии такового) и возвращает его значение.

Следует отметить, что, извлекая элемент, мы не удаляем его из хранилища (он по-прежнему остается там). Мы лишь извлекаем значение, соответствующее определенному ключу.

В: «Web Storage» и локальное хранилище — это одно и то же?

О: Веб-стандарт называется Web Storage, однако большинство людей просто называют его локальным хранилищем (фактически, браузеры как раз и предоставляют данный API-интерфейс посредством объекта `localStorage`). Web Storage — не самое удачное название для стандарта (поскольку элементы сохраняются в вашем браузере, а не в Интернете). Тем не менее мы придерживаемся его. Вы увидите, что мы будем использовать термин «локальное хранилище» чаще, чем название стандарта Web Storage.

В: Насколько широко поддерживается API-интерфейс Web Storage? Можно ли рассчитывать на такую поддержку?

О: Да, можно. На самом деле это один из наиболее поддерживаемых API-интерфейсов, даже в версиях вплоть до Internet Explorer 8, причем на текущий момент — в большинстве мобильных браузеров. Кое-где имеются предостережения, и мы обратим на них ваше внимание. В плане поддержки Web Storage вам, как всегда, следует провести тест, прежде чем пытаться использовать данный API-интерфейс. Вот как можно проверить, поддерживается ли `localStorage`:

```
if (window["localStorage"]) {
    // your localStorage code here...
}
```

Обратите внимание, что при тестировании мы выясняем, имеется ли у глобального объекта `window` свойство `localStorage`. Если оно присутствует, то мы будем знать, что браузер поддерживает `localStorage`.

В: В самом начале главы вы упоминали о 5 Мбайтах хранилища в каждом браузере. Это общий объем, который приходится на все приложения?

О: Нет, по 5 Мбайт приходится на каждый из доменов.

В: Вы сказали, что участие сервера не требуется, а потом начали вести разговор о доменах.

О: Все верно, поскольку управление хранилищем полностью осуществляется на стороне клиента. Домен вступает в дело потому, что 5 Мбайт хранилища выделяются всем страницам, исходящим из одного и того же домена. `Pet-R-Us.com` получает 5 Мбайт, `PetEmporium.com` — еще 5 Мбайт и т. д., на всех ваших компьютерах.

В: А если сравнить данный механизм с Google Gears (или с другой технологией локального хранения данных)?

О: В других технологиях, позволяющих сохранять данные в браузере, нет ничего плохого, однако локальное хранилище HTML5 сейчас является стандартом (Google, Apple, Microsoft и прочие компании считают Web Storage стандартным инструментом для сохранения содержимого локально в браузере).

В: Что произойдет, если я вызову `setItem` в отношении одного и того же ключа несколько раз? Скажем, дважды вызову `setItem` в отношении «sticky_1». В итоге в локальном хранилище у меня окажутся два ключа «sticky_1»?

О: Нет. Ключи являются уникальными в `localStorage`, поэтому `setItem` будет перезаписывать первое значение вторым. Например, если вы выполните этот код:

```
localStorage.setItem("sticky_1", "Get Milk");
localStorage.setItem("sticky_1", "Get Almond Milk");
var sticky = localStorage.getItem("sticky_1");
значением sticky будет "Get Almond Milk".
```

В: Кто сможет сохранять данные в моем локальном хранилище?

О: Управление локальным хранилищем осуществляется в соответствии с источником (вы можете рассматривать источник как свой домен) происхождения данных. Таким образом, к примеру, каждая страница `wickedlysmart.com` сможет «увидеть» элементы, сохраненные другими страницами на этом сайте, однако код с других сайтов, скажем, с `google.com`, не будет иметь доступа к данному хранилищу (он сможет получить доступ только к элементам в своем собственном локальном хранилище).

В: Когда я загружаю страницу со своего компьютера, как мы это делали в упражнениях, что является моим источником?

О: Хороший вопрос. В данном случае вашим источником будет Local Files (локальные файлы), и он отлично подходит для проведения тестирования. Если у вас будет доступ к серверу, где вы также сможете протестировать свои файлы, то тогда источником будет соответствующий домен.

Будьте осторожны! Локальное хранилище не будет функционировать должным образом, если вы используете `file://`.

Это еще один случай, когда некоторые браузеры требуют, чтобы вы загружали свои страницы, используя `localhost://` или онлайн-сервер, а не делали это из файла. Если ваши клейкие заметки не будут работать, прибегните к загрузке с сервера или воспользуйтесь другим браузером.

Итак, я могу сохранять данные в localStorage. А если потребуется сохранить числовое значение? Я полагал, что смогу использовать localStorage для сохранения целочисленных значений итогового количества элементов и цен с плавающей точкой для такого приложения, как электронная корзина, которое хочу написать. Та ли это технология, которая мне требуется?

Это именно та технология, которая вам нужна.

Верно, в ситуации с localStorage вы сможете использовать только строки в качестве ключей и значений. Однако здесь все не настолько ограничено, как может показаться. Допустим, вам нужно сохранить целочисленное значение 5. Вы можете вместо этого сохранить строку «5», а затем преобразовать ее обратно в целое число при извлечении из локального хранилища. Посмотрим, как это можно сделать в случае с целочисленными значениями и значениями с плавающей точкой. Скажем, вам требуется сохранить целочисленное значение с ключом "numItems". Для этого вы написали бы следующее:

```
localStorage.setItem("numItems", 1);
```

Разве мы не говорили, что сохранять целочисленные значения нельзя?

Может показаться, что здесь вы сохраняете целочисленное значение, однако JavaScript знает, что это должна быть строка, поэтому преобразует за вас данное значение в строку. В действительности `setItem` увидит строку "1", а не целое число. Однако JavaScript окажется не так умен, когда вы будете извлекать значение с помощью `getItem`:

```
var numItems = localStorage.getItem("numItems");
```

В данном коде мы присваиваем `numItems` строку "1", а не целочисленное значение, как бы нам хотелось. Чтобы `numItems` было числом, потребуется применить JavaScript-функцию `parseInt` для преобразования строки в целое число:

Обертываем значение в вызов `parseInt`, что преобразует строку в целое число.

```
var numItems = parseInt(localStorage.getItem("numItems"));
```

```
numItems = numItems + 1;
```

```
localStorage.setItem("numItems", numItems);
```

Затем мы опять сохраняем его, при этом JavaScript снова обеспечит преобразование.

Мы можем прибавить 1 к нему, поскольку это число.

Если вы будете сохранять значения с плавающей точкой, то взамен вам потребуется использовать функцию `parseFloat` при извлечении элементов `price` из `localStorage`:

```
localStorage.setItem("price", 9.99);
```

```
var price = parseFloat(localStorage.getItem("price"));
```

Здесь все то же самое: мы сохраняем значение с плавающей точкой, которое преобразуется в строку.

И преобразуем эту строку обратно в значение с плавающей точкой с помощью `parseFloat`.

Были ли локальное хранилище и массив разделены при рождении?

Локальное хранилище имеет и другую сторону, которой вы еще не видели. `localStorage` не только предусматривает методы получателя и установщика (то есть `getItem` и `setItem`), но также позволяет обращаться с объектом `localStorage` как с ассоциативным массивом. Что это означает? Вместо того чтобы использовать метод `setItem`, вы сможете присвоить ключу значение в хранилище, как показано далее:

```
localStorage["sticky_0"] = "Pick up dry cleaning";
```

Здесь ключ выглядит как индекс для массива хранения.

А вот наше значение, располагающееся по правую сторону от оператора присваивания.



Используя данный подход, мы также можем извлечь значение, сохраненное в ключе:

```
var sticky = localStorage["sticky_0"];
```

Присваиваем нашей переменной `sticky`...

...значение ключа `"sticky_0"` в локальном хранилище.

Это работает точно так же, как использование вызова метода `getItem`.

Неплохо, правда? Таким образом, вы можете применять любой синтаксис, поскольку они оба допустимы. А если вы привыкли использовать ассоциативные массивы на JavaScript, то данный синтаксис может показаться вам более компактным и удобочитаемым.

Погодите, это еще не все!

API-интерфейс `localStorage` предусматривает еще два интересных инструмента: свойство `length` и метод `key`. Свойство `length` содержит количество элементов в локальном хранилище. А что делает метод `key`, вы можете увидеть чуть ниже:

Осуществляем итерацию по каждому элементу.

Свойство `length` позволяет узнать, сколько элементов содержится в `localStorage`.

```
for (var i = 0; i < localStorage.length; i++) {  
  var key = localStorage.key(i);  
  var value = localStorage[key];  
  alert(value);  
}
```

Общая картина: мы используем `length` для осуществления итераций по содержимому `localStorage` (как в случае с массивом) и обращаемся к каждому ключу (например, `"sticky_0"`) по ходу процесса. Затем мы сможем использовать данный ключ для извлечения соответствующего значения.

Для каждого элемента в `localStorage` метод `key` дает нам определенный ключ (`"sticky_0"`, `"sticky_1"` и т. д.).

Проведите тест и посмотрите, выводится ли на экране диалоговое окно `alert` относительно каждого из элементов?

Затем с помощью имени ключа мы сможем извлечь соответствующее значение.

Часть
Задаваемые
Вопросы

В: Если я стану осуществлять итерации по localStorage с использованием localStorage.length и localStorage.key, то каким при этом будет порядок элементов? Таким же, каким я записывал их в хранилище?

О: Вообще-то порядок элементов не является определенным. То есть вы сможете увидеть каждую пару «ключ — значение» в хранилище посредством совершения итераций, однако не следует при этом рассчитывать на какой-то определенный порядок в коде. Фактически разные браузеры могут предусматривать различный порядок для одного и того же кода и элементов.

Игра в скорлупки

Готовы испытать удачу (или, скорее, сноровку)? Данная игра позволит проверить, насколько хорошо вы знаете localStorage, однако вам потребуется проявить решительность. Используйте свои знания об извлечении и сохранении пар «ключ — значение» в localStorage, чтобы уследить за горошиной, когда она будет перемещаться от одной скорлупки к другой.

```
function shellGame() {
    localStorage.setItem("shell1", "pea");
    localStorage.setItem("shell2", "empty");
    localStorage.setItem("shell3", "empty");
    localStorage["shell1"] = "empty";
    localStorage["shell2"] = "pea";
    localStorage["shell3"] = "empty";
    var value = localStorage.getItem("shell2");
    localStorage.setItem("shell1", value);
    value = localStorage.getItem("shell3");
    localStorage["shell2"] = value;
    var key = "shell2";
    localStorage[key] = "pea";
    key = "shell1";
    localStorage[key] = "empty";
    key = "shell3";
    localStorage[key] = "empty";

    for (var i = 0; i < localStorage.length; i++) {
        var key = localStorage.key(i);
        var value = localStorage.getItem(key);
        alert(key + ": " + value);
    }
}
```

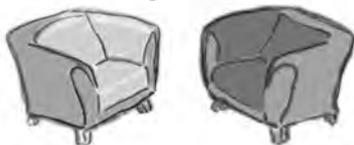
Можете использовать это пространство для отслеживания состояния localStorage.

Под какой скорлупкой находится горошина («pea»)? Свой ответ напишите здесь:

Ключ	Значение
shell1	
shell2	
shell3	

Можете набрать это, чтобы проверить свой ответ и узнать, под какой скорлупкой будет находиться горошина.

Беседа у камина



Участники сегодняшнего разговора: файлы cookie и локальное хранилище

Сегодня мы станем свидетелями разговора текущей распространенной технологии хранения данных в браузере, называемой «файлы cookie», и новым кандидатом на лидерство, имя которого «локальное хранилище».

Файлы cookie:

А вот и паш «золотой мальчик» — локальное хранилище. Мы в этом деле уже более десяти лет, и Вы думаете, что можете вот так просто добиться успеха? Да у Вас еще молоко па губах не обсохло!

Да Вы хоть имеете представление о том, со сколькими страницами мы используемся? А па свою статистику Вы когда-нибудь смотрели?

Эй, мы вездесущи, повсеместны, пас можно встретить везде! Мы не думаем, что в настольном или мобильном сегменте существует браузер (неважно, насколько старой оп версии), где пельзя было бы пас пайти.

Это мы еще посмотрим. А что имеппо Вы можете предложить сверх того, что можем предложить мы? Мы прекрасно обеспечиваем храниение данных.

Мы попятя не имеем, о чем Вы здесь говорите.

Локальное хранилище:

Конечно, па это можно и так посмотреть либо сказать, что я создаю па основе опыта, извлеченного из ваших ошибок.

Подождите несколько лет, и тогда посмотрим. Реальность такова, что я помогаю задействовать целое новое поколение веб-приложений в браузере. Многие из страниц, о которых вы упомянули, являются *всего лишь* страницами.

Я быстро догоняю вас по популярности. Среди всех технологий HTML5 я являюсь одной из наиболее хорошо поддерживаемых.

Что ж, я не уверено, что хочу упоминать об этом па публице, по у вас проблемы с объемом.

Эй, вы сами это пачали, а не я. Вам отлично известно, что вы ограничены 4 Кбайтами хранилища, а в моем случае оно более чем в 1200 раз больше!

Файлы cookie:

Мы легки, проворны, можно даже сказать, гибки.

Да бросьте, мы как открытая книга — простое хранилище, в которое можно поместить все, что вам потребуется.

А разве пары «ключ — значение» являются каким-то большим повествованием?

<Хихикаешь> Ах, да, Вы же сохраняете все в виде строки! Отличная работа! </Хихикаешь>.

Да, да, позовите меня через десять лет, и мы посмотрим, выдержали ли Вы проверку временем.

Вы увидите, как еще будете в слезах звать меня, когда Вам скажут: «Ха-ха, пять мегабайт — и это все, что у Вас есть?».

Локальное хранилище:

Вот это забавно. Вы когда-нибудь общались с разработчиками? Вы какие угодно, но только не гибкие. У вас, если вы так любите статистику, есть данные о количестве часов, которые разработчики потеряли из-за глупых ошибок и заблуждений при использовании файлов cookie?

На самом деле у вас, по сути, отсутствуют какие-либо форматы данных, из-за чего разработчикам приходится заново изобретать схему сохранения информации в файлах cookie.

Нужды в каких-либо больших повествованиях в хранилище данных нет; пары «ключ — значение» прекрасно работают, они просты и подходят для многих вычислительных приложений.

Из строк можно извлечь много пользы, а если вам потребуется что-то более сложное, то для этого есть соответствующие нуты.

О, будьте уверены. Посмотрите правде в глаза: вы с самого начала были обречены. Кто захочет позвать своих детей в вашу честь?

Серьезно о клейких заметках

Итак, вы немного поиграли с Web Storage, а теперь продолжим нашу реализацию. Мы создадим приложение Note to Self для работы с электронными клейкими заметками, чтобы вы смогли увидеть свои заметки и добавить к ним новые. Посмотрим, что мы собираемся сделать, прежде чем приступим к работе.



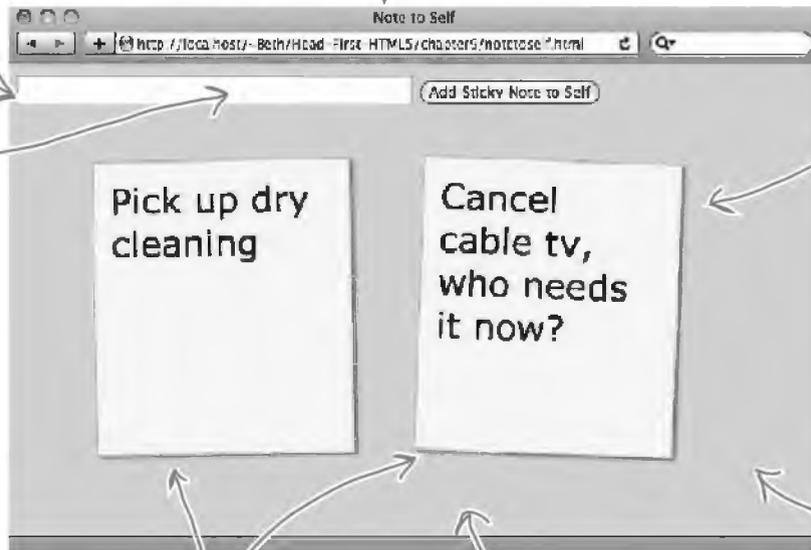
localStorage

Нам нужен способ добавления новых клейких заметок. Поэтому мы создадим форму с полем ввода и кнопкой.

Наше приложение Note to Self будет отображать клейкие заметки, сохраненные в localStorage, а также позволит нам добавлять новые заметки.

Если у нас в хранилище уже будут заметки, то мы хотим, чтобы они отображались на экране после загрузки страницы. Как, например, эти две заметки, которые уже у нас есть.

При щелчке на кнопке Add Sticky Note to Self (Добавить клейкую заметку для себя) в localStorage будет добавляться заметка.



Для отображения заметок мы будем совершать итерации по всем заметкам в localStorage и добавлять их в объектную модель документа (DOM).

Мы также увидим появление новых клейких заметок на экране, что будет происходить благодаря добавлению нового элемента в объектную модель документа (DOM) для каждой заметки.

Как вы помните, ключами для этих двух заметок являются "sticky_0" и "sticky_1". Мы будем придерживаться нашего соглашения и создадим ключи для заметок, используя целые числа с возростанием: sticky_2, sticky_3 и т. д.

Мы стилизуем электронные клейкие заметки с использованием CSS, чтобы они были похожи на настоящие листочки!

Создание интерфейса

Для начала нам необходимо найти способ вводить текст кликаемых заметок. И было бы здорово, если бы мы могли видеть их на странице, поэтому нам потребуется элемент, в котором будут содержаться все заметки, отображаемые на странице.

Напишем соответствующий код, начиная с HTML-разметки. Откройте свой HTML-файл и добавьте элемент `<form>`, элемент `` и CSS-ссылку на него, как показано ниже:



Вот наш основной HTML-файл

```

<!doctype html>
<html>
<head>
<title>Note to Self</title>
<meta charset="utf-8">
<link rel="stylesheet" href="notetoself.css">
<script src="notetoself.js"></script>
</head>
<body>
  <form>
    <input type="text" id="note_text">
    <input type="button" id="add_button" value="Add Sticky Note to Self">
  </form>

  <ul id="stickies">
  </ul>
</body>
</html>

```

Мы добавили небольшое количество CSS, чтобы наши электронные заметки выглядели более похожими на настоящие листочки. Эта книга не посвящена CSS, но вы можете взглянуть для справки в данный файл!

Поместим весь наш JavaScript-код в файл `notetoself.js`.

Мы добавили форму в качестве интерфейса пользователя для ввода новых заметок.

Нам также нужно где-то размещать наши заметки в интерфейсе, поэтому мы собираемся поместить их в неупорядоченный список.

Благодаря CSS каждый элемент списка будет выглядеть более похожим на записку-напоминание.

Теперь добавим JavaScript

У нас уже есть все необходимое на странице, а также пара клейких заметок в localStorage, ожидающих своего отображения. Давайте сделаем так, чтобы они появились на странице, для чего сначала считаем их из localStorage, а затем поместим в неупорядоченный список (элемент), который мы только что создали:

Когда страница загрузится, мы вызовем функцию init...

...которая считает все существующие клейкие заметки из localStorage и добавит их в посредством DOM.

```
window.onload = init;
```

```
function init() {
  for (var i = 0; i < localStorage.length; i++) {
    var key = localStorage.key(i);
    if (key.substring(0, 6) == "sticky") {
      var value = localStorage.getItem(key);
      addStickyToDOM(value);
    }
  }
}
```

Для этого мы будем осуществлять итерации по всем элементам в хранилище.

Извлекаем каждый ключ.

А затем убеждаемся, что данный элемент является заметкой, путем проверки, начинается ли его ключ со "sticky". Зачем мы это делаем? Что ж, в localStorage могут находиться и другие элементы помимо наших клейких заметок (подробнее об этом мы поговорим позже).

Если это клейкая заметка, то мы извлекаем ее значение и добавляем на нашу страницу (посредством DOM).

Итак, теперь нам необходимо написать функцию addStickyToDOM, которая будет вставлять заметки в элемент :

```
function addStickyToDOM(value) {
  var stickies = document.getElementById("stickies");
  var sticky = document.createElement("li");
  var span = document.createElement("span");
  span.setAttribute("class", "sticky");
  span.innerHTML = value;
  sticky.appendChild(span);
  stickies.appendChild(sticky);
}
```

Нам передается текст заметки. Необходимо создать элемент неупорядоченного списка, а затем вставить его.

Извлечем элемент с id в виде "stickies".

Создаем элемент списка и присваиваем ему имя класса "sticky" (чтобы его можно было стилизовать).

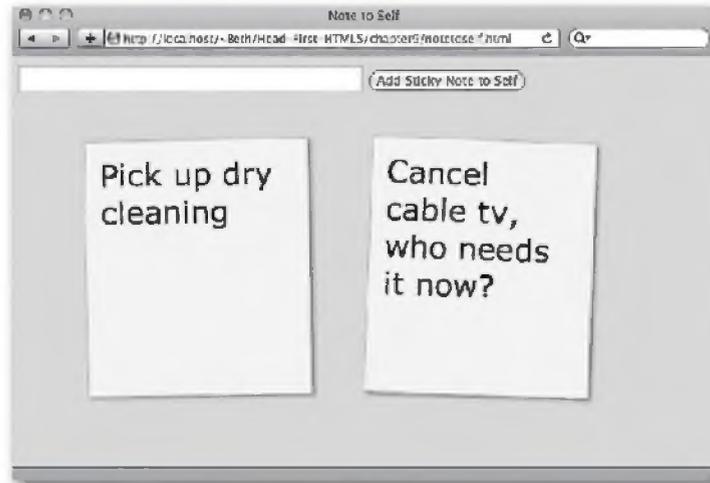
Задаем содержимое span, где находится текст заметки.

И добавляем span в элемент списка "sticky", который, в свою очередь, добавляем в элемент с id в виде "stickies".

Время еще одного тест-драйва!

Наберите приведенный выше код в своем элементе `<script>` и загрузите его в браузере.

Вот что получилось у нас, когда мы загрузили страницу в браузере:



Завершаем создание интерфейса пользователя

Осталось лишь активировать форму, чтобы у нас появился способ добавлять новые заметки. Для этого необходимо добавить обработчик, который будет вызываться при щелчке по кнопке Add Sticky Note to Self (Добавить клейкую заметку для себя), а также написать код для создания новой заметки. Вот наш код для добавления обработчика:

```
function init() {
    var button = document.getElementById("add_button");
    button.onclick = createSticky;

    // for loop goes here
}
```

Добавьте этот новый код
в свою функцию `init`:

Извлечем ссылку на кнопку
Add Sticky Note to Self
(Добавить клейкую за-
метку для себя).

Остальная часть кода
в `init` будет такой же,
как раньше, мы просто
экономим место, не при-
водя ее здесь повторно.

И добавим обра-
ботчик, который
будет вызываться
при ее нажатии.
Данному об-
работчику мы
присвоим имя
`createSticky`.

А вот код для создания повой клейкой заметки:

```
function createSticky() {
  var value = document.getElementById("note_text").value;
  var key = "sticky_" + localStorage.length;
  localStorage.setItem(key, value);
  addStickyToDOM(value);
}
```

Данный обработчик будет вызываться при щелчке кнопкой мыши.

Сначала происходит извлечение текста, введенного в поле ввода формы.

Затем нам нужно создать уникальный ключ для заметки. Воспользуемся "sticky_", конкатенированным со значением length всего локального хранилища; оно продолжит увеличиваться, не так ли?

Далее мы добавляем новую заметку в localStorage с использованием нашего ключа.

И наконец, добавляем новый текст в объектную модель документа для представления клейкой заметки.

И еще один тест-драйв!

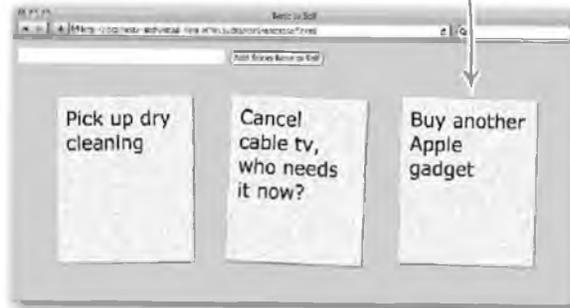


Теперь наше приложение стало по-настоящему интерактивным! Загрузите этот код в своем браузере, введите текст повой заметки и щелкните па кнопке Add Sticky Note to Self (Добавить клейкую заметку для себя). В списке клейких заметок должен появиться повый экземпляр. Вот что мы видим:

Теперь вы можете отправляться в путешествие на Фиджи, а когда вернетесь, ваши клейкие заметки по-прежнему будут находиться здесь, дожидаясь вас!

Ключом для данной заметки является "sticky_2" — значение length локального хранилища (до ее добавления), конкатенированное со "sticky_".

Результат нашего тестового прогона! Выглядит неплохо!



Обязательно попробуйте закрыть окно браузера, а затем снова открыть его. Ну как, вы по-прежнему видите свои заметки?

Часть Задаваемые Вопросы

В: Зачем нам проводить проверку для того, чтобы узнать, начинается ли ключ каждого из элементов со строки "sticky"?

О: Как вы помните, все страницы из одного домена (например, apple.com) смогут «увидеть» любой элемент, сохраненный другими страницами в этом домене. Это означает, что если мы не будем внимательно подходить к присваиванию имен нашим ключам, то можем вступить в конфликт с другой страницей, которая использует аналогичные ключи иным путем. Таким образом, это наш способ убедиться в том, что элемент является заметкой (а не, к примеру, порядковым номером или игровым уровнем), прежде чем мы используем его значение для своей заметки.

В: А что, если в localStorage окажется большое количество элементов, включая те, которые не являются клейкими заметками? Разве будет эффективным осуществление итераций по всему этому набору элементов?

О: Что ж, если речь не идет об очень большом количестве элементов, то мы сомневаемся, что вы заметите разницу. Вместе с тем вы правы, это будет неэффективным, и, возможно, существуют более оптимальные подходы к управлению нашими ключами (о некоторых из них мы вскоре поговорим).

В: Меня удивило использование значения localStorage.length в качестве номера заметки в ключе, например:

```
"sticky_" + localStorage.length
```

Зачем мы так поступили?

О: Нам необходим способ создания новых, уникальных ключей. Мы могли бы использовать что-то вроде значения времени или генерировать целое число, которое увеличивалось бы с каждым разом. Либо, как отмечалось ранее, можно использовать значение length локального хранилища (которое увеличивается каждый раз, когда мы добавляем элемент). Если вы считаете, что такой подход может оказаться сомнительным, то мы еще вернемся к этой теме. А если вы так не считали, не беспокойтесь, мы все равно вернемся к данному вопросу.

В: Я создал набор заметок в браузере Safari, а затем перешел на Chrome, но не смог увидеть их в этом браузере. В чем причина?

О: Каждый браузер поддерживает свое собственное локальное хранилище. Так что если вы создадите клейкие заметки в Safari, то сможете увидеть их только в этом браузере.

В: Я всего лишь перезагрузил страницу, и теперь мои заметки располагаются в другом порядке!

О: Когда вы добавляете новую заметку, ее элемент добавляется путем внедрения его в список заметок, поэтому такой листок всегда будет оказываться в конце списка. Когда вы перезагружаете страницу, заметки добавляются в том порядке, в котором они обнаруживаются в localStorage (а как вы помните, определенный порядок расположения элементов здесь не гарантируется). Вы могли бы подумать, что порядок будет таким же, в каком элементы добавлялись в хранилище, либо они окажутся в каком-то другом обоснованном порядке, однако рассчитывать на это нельзя. Почему? Причина заключается в том, что соответствующая спецификация не определяет порядок, в силу чего разные браузеры могут реализовывать его по-своему. Если окажется так, что ваш браузер будет возвращать элементы в порядке, который имеет для вас смысл, можете считать, что вам повезло, но не стоит рассчитывать на данный порядок, поскольку браузер пользователя может упорядочивать ваши элементы по-другому.

В: Я часто использую форму for in цикла for. А здесь он работает?

О: Сработает. Все будет выглядеть следующим образом:

```
for (var key in localStorage) {
  var value = localStorage[key];
```

Данный код будет осуществлять итерацию по каждому ключу в localStorage. Очень удобно.

В: Можно ли удалять клейкие заметки?

О: Да, удалять элементы из localStorage можно с помощью метода localStorage.removeItem. Вы также можете делать это напрямую, используя консоль браузера. В данной главе мы продемонстрируем оба этих способа.



Учитывая способ реализации клейких заметок, с нашей схемой присваивания имен возникнет проблема, если пользователь будет иметь возможность удалять заметки по своему усмотрению. Как вы думаете, в чем именно будет заключаться эта проблема?

Прервемся для небольшого запланированного мероприятия

Было бы здорово, если бы существовал инструмент для просмотра элементов в вашем localStorage напрямую? Либо инструмент для удаления элементов или даже для полной очистки хранилища, чтобы пачать все запово, когда вы будете записываться отладкой?

Все основные браузеры располагают встроенным инструментарием разработчика, позволяющим напрямую исследовать содержимое вашего локального хранилища. Данный инструментарий разпится от браузера к браузеру, поэтому вместо его полного исследования мы укажем вам правильное направление, после чего вы сможете сами покопаться и узпать, какова специфика вашего собственного браузера. В качестве примера взглянем на то, что предлагает браузер Safari:



Мы щелкнули на вкладке Resources (Ресурсы), чтобы проинспектировать localStorage.

Не говоря уже о новых версиях браузеров, которые появляются быстрее, чем мы пишем страницы!

Инструментарий разработчика в браузере Safari

Key	Value
cat	Oliver
sticky_2	Buy another Apple gadget
sticky_0	Pick up dry cleaning
sticky_1	Cancel cable tv, who needs it now?

Щелкнув здесь, вы увидите хранилище, ассоциированное с данным источником.

Щелкнув правой кнопкой мыши на одном из элементов хранилища, вы сможете отредактировать или удалить этот элемент прямо в данном окне.

Старомодные файлы cookie на случай, если они вам потребуются.

Об этом мы поговорим позже.

В браузере Safari мы можем использовать данные инструменты для перезагрузки представления Storage (Хранилище) и удаления выбранного элемента.

Пара «ключ — значение» для каждого элемента в хранилище находится здесь.

Источник хранилища. Здесь мы используем локальные файлы, загружаемые из `http://localhost`, однако это может быть и доменное имя, если вы проводите тестирование на онлайн-сервере.

Для активации или доступа к инструментарию разработчика, как мы уже говорили, в разных браузерах придется поступать по-разному. Посетите страницу `http://wickedlysmart.com/hfhtml5/devtools.html`, чтобы узпать, как это сделать именно в вашем браузере.

Поддержка типа «сделай сам»

Существует еще один способ очистки хранилища от ваших элементов (а также, как мы вскоре увидим, способ удаления их поодиночке), который потребует от вас самостоятельного обеспечения небольшой поддержки, прямо из JavaScript. API-интерфейс `localStorage` включает удобный метод `clear`, который удаляет все элементы из вашего локального хранилища (по крайней мере, элементы из вашего домена). Посмотрим, как можно использовать вызов данного метода на JavaScript, создав новый файл `maintenance.html`. Сделав это, добавьте туда приведенный ниже код, и мы пошагово разберем, как он работает.



← Это хороший инструмент для вашего набора →

```

<!doctype html>
<html>
<head>
<title>Maintenance</title>
<meta charset="utf-8">
<script>
window.onload = function() {
    var clearButton = document.getElementById("clear_button");
    clearButton.onclick = clearStorage;
}

function clearStorage() {
    localStorage.clear();
}
</script>
</head>
<body>
    <form>
        <input type="button" id="clear_button" value="Clear storage" />
    </form>
</body>
</html>

```

Мы добавили одну кнопку на страницу, а этот код добавит для данной кнопки обработчик событий, инициализируемый при ее нажатии.

← При щелчке на кнопке будет происходить вызов функции `clearStorage`.

← Данная функция лишь вызывает метод `localStorage.clear`. Используйте ее осторожно, поскольку она удалит **все** элементы, ассоциированные с источником данной страницы `Maintenance`!

↑
А вот наша кнопка. Используйте файл `maintenance.html` всякий раз, когда вам понадобится стереть все содержимое `localStorage` (хорошо подходит для тестирования).

Набрав данный код, загрузите его в своем браузере. Теперь у вас есть возможность безопасно (в случае с нашим приложением для работы с электроплыми заметками `Note to Self`) очистить `localStorage`, поэтому попробуйте это сделать! Но сначала убедитесь в том, что вы разобрались в своей инструментарии разработчика, чтобы увидеть изменения.



Будьте осторожны!

Данный код удалит все элементы в вашем домене!

Если у вас имеется сверхценное локальное хранилище, связанное с другим проектом в том же самом домене, то вы лишитесь всех своих элементов, если выполните данный код. Просто имейте это в виду...

У меня возникла проблема. Выполняя упражнения в книге, я воспользовался своими знаниями для создания новой электронной корзины для нашей компании. Мое приложение Note to Self перестало работать. Заглянув в localStorage с помощью инструментария разработчика в браузере Safari, я увидел, что нумерация всех моих заметок стала беспорядочной: "sticky_0", "sticky_1", "sticky_4", "sticky_8", "sticky_15", "sticky_16", "sticky_23", "sticky_42". У меня такое чувство, что причина в том, что я создаю в localStorage и другие элементы одновременно с заметками. Что же происходит?!



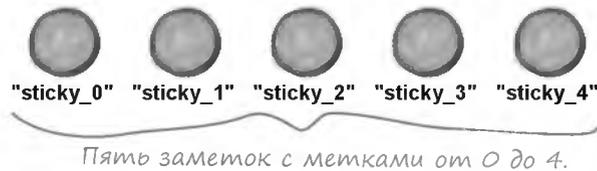
Вы столкнулись с основным недостатком проектирования.

Ладно, пришло время сказать все начистоту: ранее мы создали отличное небольшое приложение, которое должно безупречно работать довольно долгое время, пока вы не начнете вставлять какие-либо другие элементы в localStorage (как поступил Джоэль в случае со своей электронной корзиной). Как только вы сделаете это, вся наша схема отслеживания клейких заметок перестанет работать или, по крайней мере, больше не будет нормально работать.

Причина в следующем.

Если вы готовы жить с этим, что ж, отлично; в противном случае продолжайте читать дальше.

Прежде всего наши клейкие заметки нумеруются от нуля до числа, которое обозначает общее их количество (минус один):

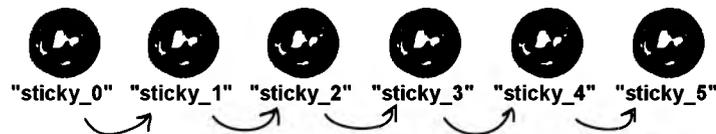


Чтобы добавить новую заметку, мы подсчитываем количество элементов в локальном хранилище и создаем новый ключ исходя из полученного количества:

```
var key = "sticky_" + localStorage.length
```



А чтобы отобразить все клейкие заметки, мы осуществляем итерацию от нуля до значения length локального хранилища (минус один):



Значением length на данный момент является 6, поэтому мы осуществляем итерацию от 0 до 5, отображая заметки от "sticky_0" до "sticky_5".

Теперь добавим элементы из электронной корзины Джозля в `localStorage`:

Вот элементы, которые Джозль использует в коде своей электронной корзины.



Теперь у нас в `localStorage` в сумме имеется девять элементов.

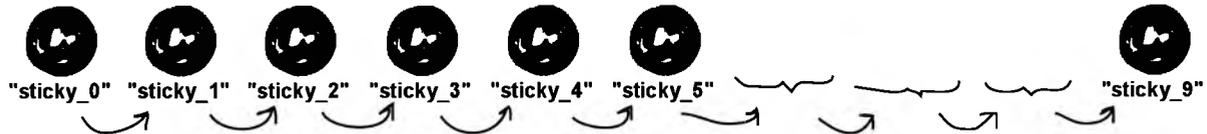
Давайте создадим новую клейкую заметку:

```
var key = "sticky_" + localStorage.length;
```



При создании новой заметки значением `length` локального хранилища будет 9, поэтому мы создаем заметку с именем "sticky_9". Хм, не похоже, что это будет правильным.

Когда нам потребуется совершить итерацию по заметкам для того, чтобы отобразить их, у нас возникнет проблема:



Теперь значением `length` является 10 (мы только что добавили новую заметку), поэтому итерация будет осуществляться от 0 до 9 с отображением каждой заметки от "sticky_0" до "sticky_9".

0, "sticky_6", "sticky_7" или "sticky_8" у нас нет.

Возьми в руку карандаш



Отметьте флажками, какие проблемы может вызывать наша текущая реализация:

- Отображение клейких заметок будет происходить неэффективно, если в `localStorage` присутствует большое количество элементов, которые не являются заметками.
- Клейкая заметка может быть перезаписана методом `setItem`, если объем `localStorage` уменьшится, несмотря на то что другое приложение удалит все свои элементы.
- Сложно быстро сказать, сколько имеется клейких заметок; вам придется осуществлять итерацию по каждому элементу в `localStorage`, чтобы извлечь все заметки.
- Используйте файлы `cookie`, поскольку такой подход должен оказаться проще!



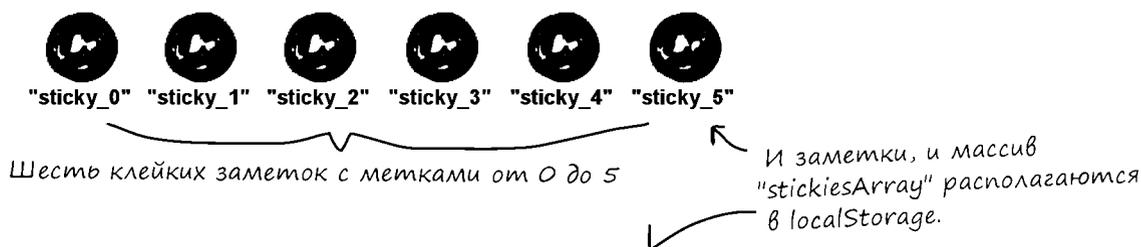
Если бы только можно было сохранить массив в `localStorage`. Мы могли бы использовать его для размещения всех ключей заметок, а также без труда узнать количество хранящихся у нас заметок. Однако всем известно, что в `localStorage` можно сохранять только строки, поэтому об этом можно лишь мечтать...

У нас есть технология...

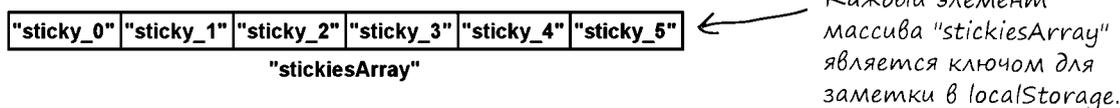
Мы не обманывали вас, когда отмечали, что вы сможете сохранять только строки в качестве значений элементов `localStorage`. Однако это не вся правда, поскольку у вас всегда будет возможность преобразовать массив (или объект) в строку, прежде чем сохранять его. Похоже на жульничество, однако это абсолютно легитимный способ сохранения в `localStorage` типов данных, которые не являются `string`.

Мы знаем, что вам ужасно хочется перейти к рассмотрению подробностей сохранения массивов, но прежде, чем мы это сделаем, взглянем на то, как массив может решить наши (и Джоэля) проблемы.

Вернемся немного назад и представим, что у нас имеется шесть клейких заметок в `localStorage`:



И, допустим, у нас в `localStorage` есть массив `"stickiesArray"`:



Теперь добавим новую клейкую заметку. Назовем ее `"sticky_815"`. Почему такое большое число? А потому, что нас больше не будет волновать, как она называется, пока ее ключ является уникальным. Таким образом, чтобы добавить заметку, мы просто вносим `"sticky_815"` в массив, а затем сохраняем элемент для данной заметки точно так же, как делали раньше. Например:



Дорабатываем наше приложение с использованием массива

Итак, мы приблизительно знаем, как собираемся отслеживать наши заметки с помощью массива. Однако пойдём немного дальше и убедимся, что мы сможем осуществлять итерации и отображать все клейкие заметки. В текущем коде мы отображаем все заметки в функции `init`. А можно ли переписать ее с использованием массива? Сначала взглянем на имеющийся код, а затем посмотрим, как он изменится (падемся, в лучшую сторону) при использовании массива. Пока не печатайте данный код; мы сосредоточимся на изменениях, которые нам необходимо внести сейчас, а не на том, чтобы сделать этот код «нулепробиваемым». Таким мы его сделаем чуть позже.

До...

```
function init() {
  // button code here...
  for (var i = 0; i < localStorage.length; i++) {
    var key = localStorage.key(i);
    if (key.substr(0, 6) == "sticky") {
      var value = localStorage.getItem(key);
      addStickyToDOM(value);
    }
  }
}
```

Вот наш старый код, который полагается на то, что у заметок будут специфические имена: `sticky_0`, `sticky_1` и т. д.

О, это было неаккуратно, если вдуматься.

Как мы теперь знаем, здесь возможны брешки, поскольку нельзя рассчитывать на то, что все заметки обязательно будут присутствовать там, если присваивать им имена исходя из общего количества элементов в `localStorage`.

Новый и улучшенный

```
function init() {
  // button code here...
  var stickiesArray = localStorage["stickiesArray"];
  if (!stickiesArray) {
    stickiesArray = [];
    localStorage.setItem("stickiesArray", stickiesArray);
  }
  for (var i = 0; i < stickiesArray.length; i++) {
    var key = stickiesArray[i];
    var value = localStorage[key];
    addStickyToDOM(value);
  }
}
```

Начинаем с извлечения `stickiesArray` из `localStorage`.

Нам необходимо удостовериться в том, что в `localStorage` есть массив. Если его там не будет, то мы создадим пустой массив.

Совершаем итерацию по массиву.

Каждый элемент массива является ключом клейкой заметки, поэтому мы используем его для извлечения соответствующего элемента из `localStorage`.

ПРИМЕЧАНИЕ: вы все еще не знаете, как сохранять и извлекать массивы, содержащиеся в `localStorage`, поэтому рассматривайте это как псевдокод, пока мы вам все не покажем. Нам потребуется внести совсем небольшое дополнение, чтобы он работал.

Добавляем соответствующее значение в объектную модель документа (DOM) точно так же, как делали это раньше.



Упражнение

Нам по-прежнему нужно выяснить, как фактически осуществляется сохранение массива в localStorage.

Вы уже могли догадаться, что у нас есть возможность использовать JSON для создания строкового представления массива. И если так и было, то вы правы. Располагая таким представлением, вы сможете сохранить его в localStorage.

Как вы помните, в API-интерфейсе JSON имеются только два метода: `stringify` и `parse`. Задействуем данные методы и завершим функцию `init` (посмотрите решение данного упражнения в конце главы, прежде чем двинетесь дальше):

```
function init() {
  // button code here...
  var stickiesArray = localStorage["stickiesArray"];
  if (!stickiesArray) {
    stickiesArray = [];
    localStorage.setItem("stickiesArray", _____(stickiesArray));
  } else {
    stickiesArray = _____(stickiesArray);
  }
  for (var i = 0; i < stickiesArray.length; i++) {
    var key = stickiesArray[i];
    var value = localStorage[key];
    addStickyToDOM(value);
  }
}
```

Мы добавили данное предложение `else`, поскольку вам потребуется кое-что предпринять, если вы извлечете массив из `localStorage` (поскольку это будет строка, а не массив).

Внесение изменений в `createSticky` с целью использования массива

Мы почти полностью рассмотрели наше приложение. Осталось лишь доработать метод `createSticky`, который, как вы помните, извлекает текст для клейкой заметки из формы, сохраняет его локально, а затем отображает. Взглянем на текущую реализацию, прежде чем внесем в нее изменения:

```
function createSticky() {
  var value = document.getElementById("note_text").value;
  var key = "sticky_" + localStorage.length;
  localStorage.setItem(key, value);
  addStickyToDOM(value);
}
```

Вместо использования значения `length` локального хранилища для генерирования ключа (что, как мы уже видели, может стать причиной проблем) нам потребуется создать более уникальный ключ.

Нам также нужно добавить клейкую заметку в массив `stickiesArray` и сохранить этот массив в `localStorage`.

Что именно необходимо изменить?

Есть две вещи, которые необходимо изменить в `createSticky`. Во-первых, нам потребуется новый способ генерирования уникального ключа для каждой клейкой заметки. Во-вторых, нам будет нужно изменить код, чтобы он сохранял определенную заметку в `stickiesArray` в `localStorage`.

1 Создание уникального ключа для клейкой заметки.

Существует масса способов создания уникальных ключей. Мы могли бы использовать значения даты и времени, либо генерировать причудливые случайные 64-битные числа, либо задействовать в сочетании с нашим приложением API-интерфейс, связанный с атомными часами. Использование значений даты и времени представляется хорошим и легким решением. JavaScript поддерживает объект `Date`, который возвращает значение в виде количества миллисекунд, прошедших с 1970 года; данное значение должно оказаться достаточно уникальным (если только вы не собираетесь создавать свои заметки с очень высокой скоростью):

```
var currentDate = new Date();
var time = currentDate.getTime();
var key = "sticky_" + time;
```

Создаем объект Date, а затем извлекаем значение текущего времени в миллисекундах.

Наш новый код для генерирования уникального ключа.

А затем генерируем ключ путем добавления полученного значения в миллисекундах в строку "sticky_".

2 Сохранение новой заметки в массиве.

Теперь, когда у нас имеется способ генерирования уникального ключа, нам необходимо сохранить текст заметки с этим ключом и добавить данный ключ в `stickiesArray`. Посмотрим, как это можно сделать, а потом объединим весь код.

```
var stickiesArray = getStickiesArray();
localStorage.setItem(key, value);
stickiesArray.push(key);
localStorage.setItem("stickiesArray",
    JSON.stringify(stickiesArray));
```

Сначала извлечем массив stickiesArray.

Затем сохраняем ключ с его значением, как делали раньше (только речь здесь идет о нашем новом ключе).

Далее используем метод push, который добавляет ключ в конец массива stickiesArray.

И сохраняем массив обратно в `localStorage`, предварительно преобразовав его с помощью метода `stringify`.

В: Что это за значение в виде количества миллисекунд, прошедших с 1970 года?

О: Вы, возможно, уже знаете, что миллисекунда представляет собой тысячную долю секунды, а метод `getTime` возвращает значение, которое является общим количеством миллисекунд, прошедших с 1970 года. Почему именно с 1970 года? Данное поведение унаследовано от операционной системы Unix, которая определяла время таким способом. Несмотря на то что такой подход неидеален (например, значения времени до 1970 года представляются в виде отрицательных чисел), он окажется кстати, когда вам потребуется уникальное число или возникнет необходимость отслеживать время в JavaScript-коде.

В: Разве применение методов `parse` и `stringify` в случае с JSON-типами не является довольно неэффективным с точки зрения производительности? А если мой массив сильно разрастется, то не окажется ли так, что и сохранение будет осуществляться неэффективно?

О: Теоретически, да, по обоим упомянутым вами пунктам. Однако в случае с типичными задачами программирования веб-страниц обычно это не является проблемой. Вместе с тем, если вы занимаетесь реализацией серьезного приложения с высокими требованиями относительно сохранения данных, то можете столкнуться с проблемами, используя JSON для преобразования элементов в строки и обратно.

Вместо того чтобы повторять весь код для извлечения и проверки stickiesArray, как мы это делали в функции init (на предыдущей странице), мы создадим новую функцию. До этого мы дойдем позже.

Затем сохраняем ключ с его значением, как делали раньше (только речь здесь идет о нашем новом ключе).

Далее используем метод push, который добавляет ключ в конец массива stickiesArray.

Отлично, как только все это заработает, я аналогичным образом доработаю свою электронную корзину, и эти два приложения смогут функционировать из одного и того же источника без проблем. Мне также нравится использовать массив; он значительно облегчает отслеживание всего!



Соединяем Все Воедино

Пора интегрировать весь новый код на основе массива, включая функции `init` и `createSticky`. Для этого мы сначала абстрагируем небольшой фрагмент кода, который необходим в обеих функциях, — это код, извлекающий массив `stickiesArray` из `localStorage`. Вы видели его в функции `init`, и он снова вам понадобится в `createSticky`. Поместим данный код в метод `getStickiesArray` — он уже должен быть вам знаком, если исходить из того кода, который мы с вами уже рассмотрели:

```
function getStickiesArray() {
```

```
  var stickiesArray = localStorage.getItem("stickiesArray");
```

```
  if (!stickiesArray) {
```

```
    stickiesArray = [];
```

```
    localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
```

```
  } else {
```

```
    stickiesArray = JSON.parse(stickiesArray);
```

```
  }
```

```
  return stickiesArray;
```

```
}
```

Сначала мы извлекаем элемент "stickiesArray" из localStorage.

Если это будет первая наша загрузка данного приложения, то элемент "stickiesArray" может отсутствовать.

И если все-таки массив будет отсутствовать, то мы создадим пустой массив, а затем сохраним его в localStorage.

Не забудьте сперва преобразовать его с помощью метода `stringify`!

В противном случае мы все же находим в localStorage массив, в отношении которого нам потребуется применить метод `parse`, чтобы преобразовать его в JavaScript-массив.

В любом случае в итоге у нас будет массив, который мы возвратим.

Соединяем все воедино (продолжение...)

Написав `getStickiesArray`, взглянем на упрощенные, финальные версии функций `init` и `createSticky`. Наберите приведенный далее код:

```
function init() {
    var button = document.getElementById("add_button");
    button.onclick = createSticky;

    var stickiesArray = getStickiesArray();

    for (var i = 0; i < stickiesArray.length; i++) {
        var key = stickiesArray[i];
        var value = localStorage[key];
        addStickyToDOM(value);
    }
}
```

Как вы помните, мы также задали здесь, в методе `init`, обработчик событий, касающийся `button`.

Далее мы извлекаем массив с ключами заметок, которые в нем содержатся.

Теперь мы будем осуществлять итерацию по массиву `stickiesArray` (не по элементам `localStorage`!).

Каждый элемент в массиве является ключом для заметки. Извлечем их.

А также извлечем его значение из `localStorage`.

И добавим в объектную модель документа (DOM) точно так же, как делали раньше.

Закопчив с `init`, нам остается лишь разобраться с `createSticky`:

```
function createSticky() {
    var stickiesArray = getStickiesArray();
    var currentDate = new Date();
    var key = "sticky_" + currentDate.getTime();
    var value = document.getElementById("note_text").value;
    localStorage.setItem(key, value);
    stickiesArray.push(key);
    localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
    addStickyToDOM(value);
}
```

Начинаем с извлечения массива `stickiesArray`.

Далее создадим уникальный ключ для нашей новой заметки.

Добавляем пару «ключ — значение» заметки в `localStorage`.

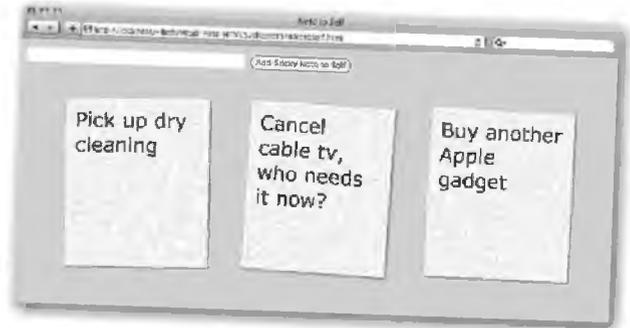
И добавляем новый ключ в массив `stickiesArray`...

А затем применяем метод `stringify` для преобразования массива и записываем его обратно в `localStorage`.

Наконец, обновляем страницу с использованием новой заметки путем ее добавления в DOM.

Тест-драйв!

Наберите весь приведенный выше код, очистите у себя `localStorage`, чтобы пачать с чистого листа. Загрузите данный код, в результате чего вы должны увидеть точно такое же поведение, как и в прошлый раз. Джоэль, ты увидишь, что теперь твой код работает должным образом!



Часть Задаваемые Вопросы

В: Мы используем "sticky_" как префикс для имен наших элементов `localStorage`. А существует ли соглашение относительно схем присваивания имен элементам `localStorage`?

О: Соглашения, касающегося присваивания имен элементам `localStorage`, не существует. Если ваше приложение располагается на небольшом сайте в домене, который находится под вашим контролем, то с присваиванием имен не должно возникнуть проблем, поскольку вы будете осведомлены обо всех именах, которые используются разными страницами на данном сайте. Мы считаем, что хорошей идеей будет использовать имя, которое является индикатором страницы или веб-приложения, полагающегося на элемент с данным именем. Таким образом, "sticky_" помогает нам запомнить, что такие элементы связаны с приложением Note to Self для работы с электронными заметками.

В: Если мое приложение Note to Self является лишь одним из многих приложений в домене, то мне нужно беспокоиться о потенциальных конфликтах?

О: Да. Вам (или тому, кто управляет веб-сайтами в соответствующем домене) в данном случае будет полезно состав-

вить план относительно того, какие имена вы будете присваивать элементам.

В: Если у меня будет много клейких заметок, то мой массив `stickiesArray` станет очень длинным. Это будет проблемой?

О: Если только вы не создадите тысячи заметок. В противном случае это не должно превратиться в проблему (а если вы все же создадите тысячи заметок, то нам хотелось бы узнать, как вам удалось оказаться столько продуктивным!). В наши дни JavaScript работает довольно быстро.

В: Просто чтобы прояснить ситуацию: мы сможем сохранить любой объект в `localStorage`, всего лишь предварительно преобразовав его с помощью метода `stringify` API-интерфейса JSON?

О: Все верно. JSON-строки представляют собой упрощенные версии JavaScript-объектов, а наиболее простые JavaScript-объекты могут быть преобразованы в строки с помощью JSON и сохранены в `localStorage`. Сюда относятся массивы (в чем вы убедились ранее), а также объекты, содержащие имена свойств и значения, как вы вскоре увидите.

Выбирайте схему присваивания имени для своих элементов `localStorage`, которая не приведет к конфликтам с другими приложениями в том же самом домене.

Если вам потребуется сохранить массивы или объекты в `localStorage`, используйте JSON. `stringify` для создания значения, которое можно будет сохранить, и `JSON.parse` после того, как извлечете его.

еще одна требуемая функция: удаление



Удаление клейких заметок

Она права: данное приложение не будет иметь большого успеха, если оно не позволит удалять заметки. Мы уже упоминали метод `localStorage.removeItem`, однако не беседовали о нем. Метод `removeItem` принимает ключ элемента и удаляет данный элемент из `localStorage`:

```
localStorage.removeItem(key);
```

↑
Данный метод удаляет содержащийся в `localStorage` элемент с определенным ключом.

↑ `removeItem` имеет один параметр: ключ элемента, подлежащего удалению.

Все это кажется довольно простым, не так ли? Но если задуматься, то удаление клейкой заметки не ограничивается лишь вызовом метода `removeItem` — нам также потребуется разобраться со `stickiesArray`...

Будьте осторожны с острыми предметами!

Возьми в руку карандаш



Давайте удалим клейкую заметку!

Ниже приведено содержимое `localStorage`. У вас имеется весь необходимый JavaScript-код наряду с методом `removeItem`. Набросайте карандашом на бумаге то, что нужно сделать, чтобы удалить `sticky_1304220006342` из `localStorage`. Сделав это, напишите внизу псевдокод, чтобы показать, как вы собираетесь писать свой настоящий код.



"sticky_1304294652202" "sticky_1304220006342" "sticky_1304221683892" "sticky_1304221742310" "элемент 1
электронной корзины" "элемент 2
электронной корзины"

```
"sticky_1304294652202" "sticky_1304220006342" "sticky_1304221742310" "sticky_1304221683892"
```

"stickiesArray"

← Свой псевдокод
напишите здесь.



Возьми в руку карандаш

Решение

Давайте удалим клейкую заметку!

Ниже приведено содержимое localStorage. У вас имеется весь необходимый JavaScript-код наряду с методом `removeItem`. Набросайте карандашом на бумаге то, что нужно сделать, чтобы удалить `sticky_1304220006342` из localStorage. Сделав это, напишите внизу псевдокод, чтобы показать, как вы собираетесь писать свой настоящий код. Вот наше решение этого упражнения.

```
localStorage.removeItem("sticky_1304220006342");
```



- 1). Удалить клейкую заметку с ключом `"sticky_1304220006342"` из localStorage с помощью метода `localStorage.removeItem`.
- 2). Извлечь `stickiesArray`.
- 3). Удалить элемент с ключом `"sticky_1304220006342"` из `stickiesArray`.
- 4). Записать `stickiesArray` обратно в localStorage (предварительно преобразовав его с помощью метода `stringify`).
- 5). Найти `"sticky_1304220006342"` в объектной модели документа (DOM) и удалить ее.

Функция deleteSticky

Вы составили план того, как будет осуществляться удаление клейких заметок, ноэтому давайте взглянем на функцию deleteSticky:

```
function deleteSticky(key) {
  localStorage.removeItem(key);
  var stickiesArray = getStickiesArray();
  if (stickiesArray) {
    for (var i = 0; i < stickiesArray.length; i++) {
      if (key == stickiesArray[i]) {
        stickiesArray.splice(i, 1);
      }
    }
    localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
  }
}
```

Сначала мы удаляем заметку из localStorage с помощью метода `removeItem`, передав ему ключ заметки, подлежащей удалению.

Используем функцию `getStickiesArray` для извлечения `stickiesArray` из localStorage.

Убеждаемся в том, что у нас имеется `stickiesArray` (на всякий случай), а затем совершаем итерацию по массиву в поисках ключа, который хотим удалить.

Отыскав требуемый ключ, удаляем его из массива с помощью `splice`.

`splice` удаляет элементы из массива, начиная с позиции, обусловленной первым аргументом (`i`), при этом количество удаляемых элементов окажется таким, как указано во втором аргументе (`1`).

Наконец, сохраняем `stickiesArray` (при этом соответствующий ключ из него уже удален) обратно в localStorage.



Я разобралась в коде, однако не могу понять, как мы извлекаем ключ для передачи `deleteSticky`. Если вдуматься, то как пользователь вообще сможет выбрать заметку для удаления?

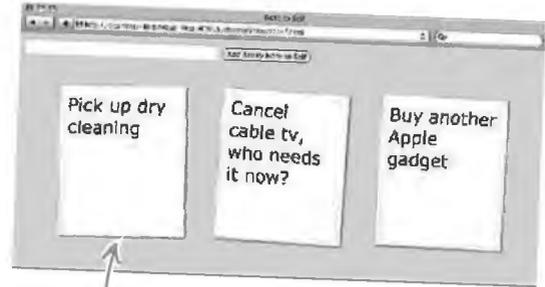
Как выбрать заметку для удаления?

Нам необходимо обеспечить для пользователя способ выбирать заметку для удаления. Мы могли бы пойти затейливым путем и добавить но небольшой никтограмме удаления для каждой заметки, однако в нашем приложении Note to Self мы поступим намного проще: будем просто удалять определенную заметку, если пользователь щелкнет на ней. Это, возможно, не самая лучшая реализация с точки зрения удобства пользования, однако простая.

Для реализации этого нам сначала потребуется внести изменения в наши клейкие заметки, чтобы мы смогли выявлять, когда пользователь щелкает на заметке, после чего передадим ее функции `deleteSticky`. Многое из этого должно происходить в функции `addStickyToDOM`:

Общая картина: мы будем использовать ключ клейкой заметки, который, как вы помните, представляет собой "sticky_" + значение времени, чтобы уникально идентифицировать заметку. Мы будем передавать данный ключ всякий раз при вызове `addStickyToDOM`.

```
function addStickyToDOM(key, value) {
    var stickies = document.getElementById("stickies");
    var sticky = document.createElement("li");
    sticky.setAttribute("id", key);
    var span = document.createElement("span");
    span.setAttribute("class", "sticky");
    span.innerHTML = stickyObj.value;
    sticky.appendChild(span);
    stickies.appendChild(sticky);
    sticky.onclick = deleteSticky;
}
```



Когда мы щелкнем на клейкой заметке с помощью мыши, она будет удалена.

Добавляем уникальный идентификатор в случае с элементом ``, который представляет клейкую заметку в DOM. Мы делаем это для того, чтобы функция `deleteSticky` знала, на какой заметке вы щелкнули. Поскольку нам уже известно, что ключ заметки уникален, будем просто использовать его как идентификатор.

Мы также добавляем обработчик событий `click` для каждой клейкой заметки. При щелчке на заметке будет происходить вызов `deleteSticky`.



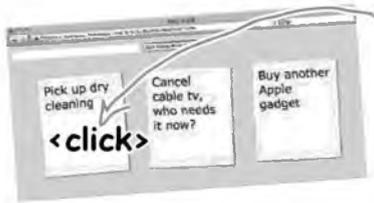
Упражнение

Ваша задача — обновить весь код таким образом, чтобы везде, где мы будем вызывать `addStickyToDOM`, мы передавали ключ, а также значение. Вам не должно составить труда найти эти места. Выполнив данное задание, посмотрите его решение в конце главы, чтобы проверить свои результаты.

Не пропускайте это упражнение, иначе предстоящий тест-драйв окажется неудачным!

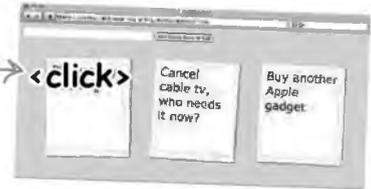
Как извлечь заметку для удаления, используя объект event

Для каждой заметки у нас на данный момент имеется обработчик событий, ведущий прослушивание событий `click`. Когда вы щелкнете на заметке, произойдет вызов функции `deleteSticky`, которой будет передан объект `event` с информацией о соответствующем событии, например на каком элементе щелкнул пользователь. Мы можем обратиться к `event.target`, чтобы сказать, на какой заметке произошел щелчок. Давайте более пристально посмотрим, что произойдет, когда вы щелкнете на заметке.



Если вы щелкнете на желтой части заметки, то `event.target` будет элемент ``. Это именно то, что нам нужно, поскольку `` теперь имеет идентификатор в виде ключа клейкой заметки.

Если вы щелкнете на тексте, то `event.target` будет `` внутри ``, а это не то, что нам нужно.



```
<li id="sticky_1304270008375">
  <span class="sticky">Pick up dry cleaning</span>
</li>
```

← Это HTML для заметки, которую мы создаем в `addStickyToDOM`.

Так или иначе, объект `event`, сгенерированный вашим щелчком, передается `deleteSticky`.

`target` — это элемент, на котором вы щелкнули и который сгенерировал объект `event`. Мы можем извлечь идентификатор данного элемента из свойства `target`. Если `target` является ``, то все в порядке.

```
function deleteSticky(e) {
  var key = e.target.id;
  if (e.target.tagName.toLowerCase() == "span") {
    key = e.target.parentNode.id;
  }
  localStorage.removeItem(key);
  var stickiesArray = getStickiesArray();
  if (stickiesArray) {
    for (var i = 0; i < stickiesArray.length; i++) {
      if (key == stickiesArray[i]) {
        stickiesArray.splice(i, 1);
      }
    }
  }
  localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
  removeStickyFromDOM(key);
}
```

Если `target` окажется ``, то нам потребуется извлечь идентификатор родительского элемента, то есть `` (`` — это элемент с идентификатором, который является необходимым нам ключом).

← Теперь мы можем воспользоваться ключом для удаления элемента из `localStorage`, а также из `stickiesArray`.

← Нам также потребуется удалить ``, содержащий клейкую заметку, из страницы, чтобы данная заметка исчезла, когда вы на ней щелкнете. Этим мы и займемся далее...

Удаление заметки также из DOM

Чтобы завершить удаление, нам понадобится реализовать функцию `removeStickyFromDOM`. Ранее мы обновили функцию `addStickyToDOM` с целью добавления ключа клейкой заметки в качестве идентификатора элемента ``, который содержит заметку, присутствующую в объектной модели документа. Поэтому мы сможем воспользоваться `document.getElementById` для поиска клейкой заметки в DOM. Мы извлечем `parentNode` заметки и применим метод `removeChild` для ее удаления:

Передаем ключ (также являющийся идентификатором) клейкой заметки, которую ищем.

```
function removeStickyFromDOM(key) {
  var sticky = document.getElementById(key);
  sticky.parentNode.removeChild(sticky);
}
```

`` `` удалить дочерний узел ``

Извлекаем элемент `` из DOM...

...и удаляем, для чего сначала извлекаем его `parentNode`, а затем применяем `removeChild`.

Итак, проведем тест... 

Наберите весь приведившийся выше код, загрузите страницу, добавьте и удалите несколько клейких заметок. Завершите работу браузера, затем снова запустите его и хорошенько протестируйте приложение!

Теперь мы можем удалять заметки!



Отличная работа! А не могли бы вы теперь сделать так, чтобы я смогла раскрашивать свои заметки? Ну там, знаете, желтым цветом те, на которых записаны срочные дела, синим — на которых помечены идеи, розовым — заметки со второстепенными делами. В таком духе?



Конечно можно!

Учитывая ваш опыт в этом деле, мы сможем справиться с данной задачей. Как именно мы это сделаем? Что ж, мы создадим объект для размещения текста заметки и ее цвета, а затем сохраним его как значение элемента заметки, предварительно задействовав `JSON.stringify` для преобразования его в строку.

Обновление интерфейса пользователя для выбора цвета заметок

На данный момент все наши заметки имеют желтый цвет. А не будет ли лучше, если нам станет доступен для выбора целый диапазон цветов?

Мы могли бы добавить сюда меню выбора, чтобы вы могли выбирать для своей заметки любой цвет.



Так лучше, не правда ли?

Сначала примемся за легкую часть: обновим HTML, чтобы у нас появилось меню выбора цветов, из которого пользователь сможет выбрать пугный цвет. Отредактируйте файл `notetoself.html` и обновите свою форму с целью добавления цветов, как показано далее:

```

<html>
...
  <form>
    <label for="note_color">Color: </label>
    <select id="note_color">
      <option value="LightGoldenRodYellow">yellow</option>
      <option value="PaleGreen">green</option>
      <option value="LightPink">pink</option>
      <option value="LightBlue">blue</option>
    </select>
    <label for="note_text">Text:</label> <input type="text" id="note_text">
    <input type="button" id="add_button" value="Add Sticky Note to Self">
  </form>
...
</html>

```

Мы вносим изменения только в форму, а все остальное останется прежним.

Обратите внимание на идентификатор `<select>`; он нам потребуется для извлечения значения выбранного параметра на JavaScript.

Мы добавили четыре цвета для заметок, из которых можно будет выбрать нужный.

Каждое значение является именем цвета, которое мы можем вставить прямо в стиль для наших заметок.

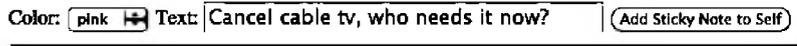
Остальная часть формы останется прежней.

Добавим метку для текста заметки, чтобы пользователь знал, для чего предназначено данное поле.

Мы использовали CSS для определения цвета по умолчанию. Теперь необходимо сохранить цвет вместе с самой заметкой. Возникает вопрос: как мы будем сохранять цвет для заметки в `localStorage`?

Метод JSON.stringify — не только для массивов

Для сохранения цвета заметки вместе с ее текстом у нас есть возможность прибегнуть к той же методике, которую мы использовали со `stickiesArray`: можно сохранить объект, содержащий текст и цвет, как значение для заметки в `localStorage`.



Мы будем брать введенные пользователем значения для цвета и текста заметки и «упаковывать» их в простой объект.

И мы сохраним его в `localStorage` с использованием ключа клейкой заметки.

Точно так же, как и в случае со `stickiesArray`, нам придется вызвать `JSON.stringify` в отношении значения заметки, прежде чем мы вызовем `localStorage.setItem` для сохранения данного значения.

```
var stickyObj = {
  "value": "Cancel cable tv, who needs it now?",
  "color": "LightPink"
};
```

Перенишем функцию `createSticky` для сохранения цвета вместе с текстом клейкой заметки. Для представления текста и цвета будем использовать наш удобный объект:

```
function createSticky() {
  var stickiesArray = getStickiesArray();
  var currentDate = new Date();
  var colorSelectObj = document.getElementById("note_color");
  var index = colorSelectObj.selectedIndex;
  var color = colorSelectObj[index].value;
  var key = "sticky_" + currentDate.getTime();
  var value = document.getElementById("note_text").value;
  var stickyObj = {
    "value": value,
    "color": color
  };
  localStorage.setItem(key, JSON.stringify(stickyObj));
  stickiesArray.push(key);
  localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
  addStickyToDOM(key, stickyObj);
}
```

Мы делаем то, что обычно принято для извлечения значения выбранного цвета.

Затем используем данное значение цвета для создания `stickyObj` — объекта, содержащего два свойства, текст заметки и цвет, выбранный пользователем.

Преобразуем `stickyObj` с помощью метода `JSON.stringify`, прежде чем поместим его в `localStorage`.

Передаем объект вместо текстовой строки функции `addStickyToDOM`. А это означает, что вам также потребуется обновить `addStickyToDOM`, не так ли?

Использование нового объекта stickyObj

Когда мы передаем stickyObj функции addStickyToDOM, нам необходимо обновить эту функцию, чтобы использовать объект вместо строки, которую мы передавали прежде, а также чтобы задать цвет фона для заметки. Эти изменения довольно легко внести:

```
function addStickyToDOM(key, stickyObj) {
    var stickies = document.getElementById("stickies");
    var sticky = document.createElement("li");
    sticky.setAttribute("id", key);
    sticky.style.backgroundColor = stickyObj.color;
    var span = document.createElement("span");
    span.setAttribute("class", "sticky");
    span.innerHTML = stickyObj.value;
    sticky.appendChild(span);
    stickies.appendChild(sticky);
    sticky.onclick = deleteSticky;
}
```

Объекты элементов HTML обладают свойством style, которое вы можете использовать для доступа к стилю соответствующего элемента.

Нужно изменить здесь параметр, чтобы им был stickyObj, а не текстовое значение заметки.

Извлекаем цвет из объекта stickyObj, который передаем addStickyToDOM.

Обратите внимание: когда мы указываем свойство, связанное с цветом фона, на JavaScript, мы определяем его как backgroundColor, а НЕ background-color, как на CSS.

Затем нужно извлечь текстовое значение, которое мы собираемся использовать для заметки, из объекта.

Есть еще одно место, где нам необходимо обновить код. И находится оно в функции init, где мы извлекаем заметки из localStorage и передаем их addStickyToDOM, когда в первый раз загружаем страницу.

```
function init() {
    var button = document.getElementById("add_button");
    button.onclick = createSticky;

    var stickiesArray = getStickiesArray();

    for (var i = 0; i < stickiesArray.length; i++) {
        var key = stickiesArray[i];
        var value = JSON.parse(localStorage[key]);
        addStickyToDOM(key, value);
    }
}
```

Теперь, когда мы будем извлекать значение заметки из localStorage, нам потребуется преобразовать его с помощью JSON.parse, поскольку это больше не строка, а объект.

И мы передадим данный объект функции addStickyToDOM вместо строки (код выглядит так же, однако передаем мы уже что-то другое).

Тест-драйв цвета заметок



Прежде чем снова вынолнять приложение Note to Self, вам нотребуется очистить localStorage, носкольку нредыдущие версии наших клейких заметок не имели никакого цвета, и сейчас мы иснользуем другой формат для значений наших заметок. Ранее мы иснользовали строки, а тенерь — объекты. Поэтому очистите у себя localStorage, нерезагрузите страницу и добавьте несколько заметок, выбрав при этом разный цвет для каждой из них. Вот как выглядят наши заметки (кромне того, мы заглянем в наше локальное хранилище).

Вы можете воспользоваться файлом maintenance.html для очистки своего localStorage либо обратиться к консоли.

Мы выбрали желтый, розовый и синий цвет для заметок.



Значение каждой заметки теперь является объектом (преобразованным с помощью метода JSON.stringify), который содержит текстовое значение и цвет соответствующей заметки.

Я думал, раз мы можем сохранять объекты и массивы, то почему бы просто не сохранить все заметки в самом массиве, зачем нам нужны все эти остальные элементы? Похоже, все только усложняется, хотя мы могли бы просто вложить их в один элемент в `localStorage`.



В некоторых ситуациях так поступать весьма разумно.

Используя свои знания, мы, несомненно, смогли бы разработать клейкие заметки таким образом, чтобы они представляли собой объекты, вложенные в массив. В дальнейшем вы могли бы именно так и поступить. Это также имело бы смысл в случае с электронной корзиной. Единственный недостаток заключается в том, что методам `JSON.stringify` и `JSON.parse` приходится проделывать массу дополнительной работы всякий раз, когда вы вносите изменения: например, чтобы добавить заметку, нам придется преобразовать с помощью `JSON.parse` целый набор заметок, потом добавить заметку, а затем снова преобразовать все заметки уже с помощью `JSON.stringify`, прежде чем записывать их обратно в хранилище. Учитывая количество данных в нашем случае, это не должно стать проблемой (однако задумайтесь о мобильных устройствах с не очень мощными процессорами и о влиянии использования ресурсов процессора на уровень заряда аккумулятора).

Таким образом, решение о том, «унаковать» вам все в один объект или же в массив в `localStorage`, будет зависеть от количества элементов данных, которое вам потребуется сохранить, а также от того, насколько большим является каждый из них и обработку какого рода вы собираетесь осуществлять в отношении этих элементов.

Несмотря на то что наша реализация может быть слегка громоздкой для ограниченного количества заметок, мы надеемся, что она отлично помогла вам понять, что такое API-интерфейс `localStorage` и как обращаться с имеющимися в нем элементами.

НЕ ПЫТАЙТЕСЬ СДЕЛАТЬ ЭТО ДОМА (ИЛИ КАК РАЗИЕСТИ В ПУХ И ПРАХ ВАШИ 5 МБАЙТ)

Как мы уже говорили, в сумме у вас будет 5 Мбайт хранилища для браузера каждого из пользователей. Несмотря на то что такой объем может показаться большим, все ваши данные сохраняются в виде строк, а не в байтовом формате. Возьмите, к примеру, размер государственного долга: если выразить его в виде значения с плавающей точкой, то для его размещения в хранилище потребуется не много места, однако если выразить его в виде строкового значения, то для его сохранения потребуется гораздо больший объем. Таким образом, хранилище размером 5 Мбайт способно вместить не так много, как могло показаться.

Так что же произойдет, когда вы исчерпаете 5 Мбайт? К сожалению, это одно из поведений, которые не определяются спецификацией HTML5, и браузеры могут поступать по-разному, когда вы превысите свой лимит. Браузер может спросить у вас, хотите ли вы увеличить объем хранилища, либо сгенерирует исключение `QUOTA_EXCEEDED_ERR`, которое можно перехватить следующим образом:

try/catch перехватывает любые исключения, генерируемые в блоке try.

Это область JavaScript, которую мы не затрагивали, и вы можете захотеть исследовать этот вопрос подробнее.

Не все браузеры в настоящий момент генерируют исключение `QUOTA_EXCEEDED_ERR`. Однако они все же сгенерируют исключение, когда вы превысите свой лимит, поэтому не лишним будет разобраться с общим случаем исключения при сохранении того или иного элемента в хранилище.

```
try {  
    localStorage.setItem(myKey, myValue);  
} catch (e) {  
    if (e == QUOTA_EXCEEDED_ERR) {  
        alert("Out of storage!");  
    }  
}
```

Вызов `setItem` в середине блока `try`; если что-нибудь пойдет не так и `setItem` сгенерирует исключение, то будет задействован блок `catch`.

Проверяем, ошибка ли это квоты хранилища (а не какой-то другой тип исключения). Если так оно и есть, мы выведем для пользователя диалоговое окно `alert` с соответствующим сообщением. Вы, скорее всего, захотите сделать что-то более значительное, чем просто вывод окна `alert`.





ОПАСНО Взрывное задание

Давайте попробуем довести свой браузер до предела, посмотреть, из чего он сделан и как далеко может пойти, узнать, как он себя поведет в стрессовой ситуации. Напишем немного кода, который заставит ваш браузер выйти за лимит объема его хранилища:

```

<html>
<head>
<script>

localStorage.setItem("fuse", "-");
while(true) {
    var fuse = localStorage.getItem("fuse");
    try {
        localStorage.setItem("fuse", fuse + fuse);
    } catch(e) {
        alert("Your browser blew up at" + fuse.length + " with exception: " + e);
        break;
    }
}
localStorage.removeItem("fuse");
</script>
</head>
<body>
</body>
</html>

```

Начнем с односимвольной строки с ключом "fuse".

И просто будем, не останавливаясь, увеличивать ее размер...
...путем удваивания этой строки (посредством конкатенации ее с самой собой).

Затем попытаемся записать ее обратно в localStorage.

Если браузер аварийно завершит работу — дело сделано! Выведем для пользователя диалоговое окно alert с соответствующим сообщением и выйдем из данного цикла.

Не будем оставлять беспорядок и удалим данный элемент из localStorage.

Если у вас хватит духу воспользоваться этим кодом, то опишите здесь свои результаты.

Наберите этот код, «зажгите фитиль», загрузив его, и поразвлекайтесь! Испытайте его в разных браузерах.



**Будьте
осторожны!**

Вы используете этот код на свой страх и риск!

Данный код способен вызвать серьезный сбой браузера, из-за чего операционная система «очень расстроится», что может привести к потере результатов вашей работы. Используйте его на свой страх и риск!



Я провожу бета-тестирование своей электронной корзины. Пользователи не хотят, чтобы содержимое их корзины оставалось в браузере. Как можно удалить все элементы электронной корзины, когда пользователь закрывает браузер? Я выбрал не ту технологию?

Нет, Люк, есть еще один Скайукер.

Оказывается, у `localStorage` имеется «сестра» по имени `sessionStorage`. Если вы подставите глобальную переменную `sessionStorage` везде, где использовали `localStorage`, то ваши элементы будут сохраняться только в течение сеанса браузера. Таким образом, как только данный сеанс закончится (другими словами, пользователь закроет окно браузера), сохраненные элементы будут удалены.

Объект `sessionStorage` поддерживает точно такой же API-интерфейс, что и `localStorage`, но этому вам уже известно о нем все необходимое.

Воспользуйтесь им!

* КТО И ЧТО ДЕЛАЕТ? *

На данный момент вы полностью изучили API-интерфейс `localStorage`. Чуть ниже приведены главные действующие лица данного API-интерфейса, скрытые под масками. Посмотрите, сможете ли вы разобраться, кто из них для чего используется. В качестве примера мы соотнесли одно из описаний с нужной позицией.

clear	Используйте меня для сохранения элементов на длительный срок.
sessionStorage	Я припимаю ключи и значения, которые затем записываю в <code>localStorage</code> . Имейте в виду, что если в <code>localStorage</code> уже имеется такой ключ, я не буду предупреждать вас об этом, а просто перезапишу его, поэтому вам следует понимать, о чем вы просите.
key	Если вы станете злоупотреблять гостеприимством в <code>localStorage</code> и использовать слишком много пространства, то будет сгенерировано исключение и вы получите от меня известие.
setItem	Нужно удалить элемент? Я аккуратно выполню эту работу.
removeItem	
length	Я — разновидность хранилища на короткий срок: буду сохранять ваши данные, пока открыто окно браузера. Закройте окно браузера и — бац! — все ваши данные исчезли.
getItem	Когда все элементы в <code>localStorage</code> больше вам не нужны, я паваю порядок и выбрасываю их, оставляя вам чистое и пустое локальное хранилище (имейте в виду, что я могу паводить порядок только в своем собственном источнике).
localStorage	Нужно узнать количество элементов в вашем <code>localStorage</code> ? Я помогу вам с этим.
QUOTA_EXCEEDED_ERR	Дайте мне индекс, и я предоставлю вам ключ от него в <code>localStorage</code> .

Теперь, когда вы изучили localStorage, как вы собираетесь использовать его?

Существует масса способов использования localStorage. В приложении Note to Self для работы с электронными заметками мы использовали его таким образом, что нам не потребовался сервер, но даже при паличии сервера localStorage может припести довольно много пользы. Ниже приведем ряд других вариантов, которыми пользуются разработчики:



В своем новом Твиттер-клиенте я буду кэшировать с помощью localStorage поисковые результаты Твиттера для повышения производительности. Когда мои пользователи будут выполнять поиск, я сначала проверю локальные результаты. Все это по-настоящему поможет моим мобильным пользователям.

Я буду сохранять плейлисты с метаданными для своих пользователей. Они смогут сохранить свои любимые клипы вместе с временным кодом, указывающим, где они прервали просмотр.



Я использую sessionStorage для своей новой электронной корзины в коммерческой интернет-библиотеке. Если пользователь закрывает окно браузера, необходимо, чтобы содержимое его корзины удалялось.



У меня есть действительно классная игра, которая работает в двух разных окнах браузера, и я использую `localStorage` для синхронизации состояния.



Я сохраняю множество локальных данных, чтобы сделать приложения моих клиентов быстрыми на мобильных устройствах. Наличие большого хранилища на клиентской стороне дает огромный выигрыш.



Я использую новый способ сохранения состояния пользователей. Зачастую мне требовалось своего рода сеансовое хранилище на стороне сервера. Теперь я могу просто сохранять состояние моих пользователей локально и задействовать код на стороне сервера, только когда мне приходится это делать.



КЛЮЧЕВЫЕ МОМЕНТЫ

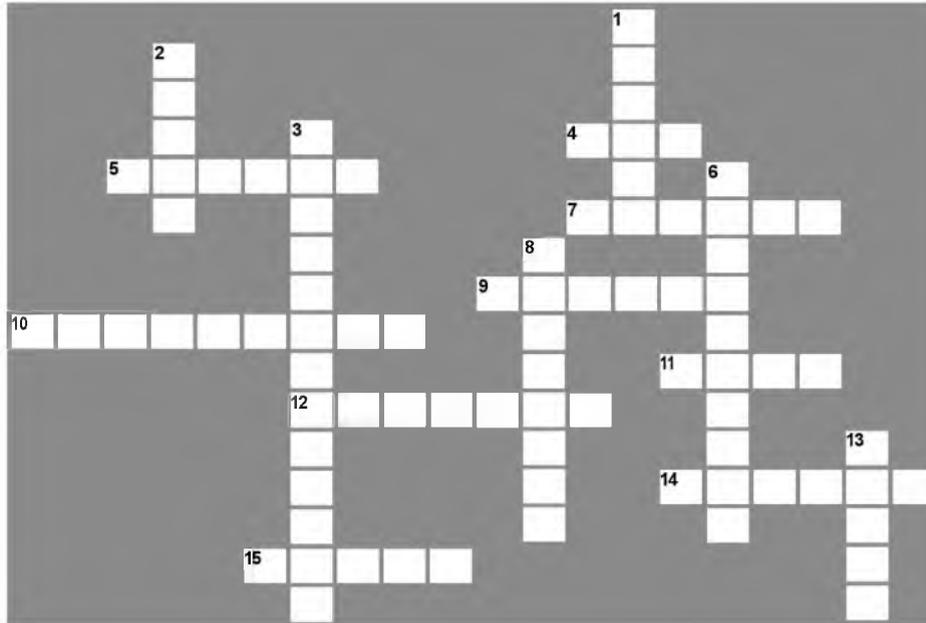


- Web Storage — это хранилище в вашем браузере и API-интерфейс, который вы можете использовать для сохранения и извлечения элементов из хранилища.
- Большинство браузеров обеспечивают по крайней мере по 5 Мбайт хранилища на каждый источник.
- Web Storage состоит из `localStorage` и `sessionStorage`.
- Локальное хранилище сохраняет данные на постоянной основе, даже если вы закроете окно браузера или вообще завершите его работу.
- Элементы в `sessionStorage` будут удалены, если вы закроете окно браузера или завершите его работу. `sessionStorage` хорошо подходит для временных элементов, но не годится как хранилище для данных на более длительный срок.
- И `localStorage`, и `sessionStorage` используют один и тот же API-интерфейс.
- Web Storage предусматривает организацию по источнику (считайте — домену). Источник — это местоположение документа в Интернете (например, `wickedlysmart.com` или `headfirstlabs.com`).
- Каждый домен располагает отдельным хранилищем, в силу чего элементы, сохраненные в одном источнике, не будут видны веб-страницам в другом источнике.
- Используйте `localStorage.setItem(ключ)` для добавления значения в хранилище.
- Используйте `localStorage.getItem(ключ)` для извлечения значения из хранилища.
- Вы можете использовать тот же синтаксис, который применяется в случае с ассоциативными массивами, для сохранения элементов в хранилище и извлечения их оттуда. Используйте для этого `localStorage[ключ]`.
- Используйте метод `localStorage.key()` для перечисления ключей в `localStorage`.
- `localStorage.length` — это количество элементов в `localStorage` в определенном источнике.
- Используйте консоль в своем браузере для просмотра и удаления элементов в `localStorage`.
- Вы можете удалять элементы напрямую из `localStorage`, щелкнув правой кнопкой мыши на том или ином элементе и выбрав в появившемся меню пункт Delete (Удалить) (данный подход работает не во всех браузерах).
- Вы можете удалять элементы из `localStorage` в коде, используя методы `removeItem(ключ)` и `clear`. Следует отметить, что метод `clear` удаляет все элементы в `localStorage` в том источнике, где вы проводите очистку.
- Ключи для всех элементов `localStorage` должны быть уникальными. Если вы примените тот же ключ, который уже имеется у существующего элемента, то перезапишете значение данного элемента.
- Сгенерировать уникальный ключ можно с использованием значения текущего времени в миллисекундах, прошедших с 1970 года, прибегнув к методу `getTime()` объекта `Date`.
- Важно разработать схему присваивания имен для своих приложений, которая по-прежнему сможет работать, если элементы будут удалены из хранилища либо если другое приложение поместит элементы в данное хранилище.
- Web Storage в текущий момент поддерживает сохранение строк как значений для ключей.
- Вы можете преобразовывать числовые значения, сохраненные в `localStorage` как строки, в числовые значения, используя `parseInt` или `parseFloat`.
- Если вам потребуется сохранить комплексные данные, можете воспользоваться JavaScript-объектами и преобразовать их в строки перед сохранением, прибегнув к `JSON.stringify`, а после извлечения преобразовать их снова в объекты с помощью `JSON.parse`.
- Локальное хранилище может оказаться особенно полезным на мобильных устройствах для снижения требований к пропускной способности канала.
- Сеансовое хранилище подобно локальному хранилищу за исключением того, что элементы, которые в него помещены, не сохраняются там на постоянной основе и удаляются, если вы закрываете вкладку, окно или выходите из браузера. Сеансовое хранилище подходит для сохранения элементов на короткий срок, например во время сеанса, связанного с посещением интернет-магазина и использованием электронной корзины.



HTML5-Кроссворд

Уделите некоторое количество времени тестированию своего собственного локального хранилища.



По горизонтали

4. Имя сестры Люка Скайуокера.
5. Когда мы использовали значение _____ `localStorage` для генерирования имен ключей, то столкнулись с проблемой — в последовательности имен наших клейких заметок оказались брешаи.
7. `localStorage` позволяет сохранять только _____.
9. Мы можем выяснить, на какой заметке щелкнул пользователь, обратившись к `event`. _____.
10. Нам необходимо преобразовать объект с помощью метода _____, прежде чем сохранять его в `localStorage`.
11. Большинство браузеров предлагает _____ мегабайт хранилища на каждый источник.
12. Данный метод используется для сохранения элементов в `localStorage`.
14. Мы полагали, что можно лишь мечтать о возможности сохранить _____ в `localStorage`, но, как оказалось, она существует, и дает ее JSON.
15. Используйте `try/_____` для выявления ошибок, возникающих из-за превышения квоты хранилища, в `localStorage`.

По вертикали

1. Мы создаем _____ для размещения текста заметки и ее цвета в одном элементе `localStorage`.
2. У файлов `cookie` имеется проблема с _____.
3. Мы использовали _____ для размещения ключей всех наших клейких заметок, благодаря чему отыскать их в `localStorage` не составит труда.
6. Сеансовое хранилище подобно локальному хранилищу за исключением того, что элементы, которые в него помещены, не сохраняются там на _____ основе и удаляются, если вы закрываете окно браузера.
8. Используйте _____ для преобразования строки в целочисленное значение.
13. Если вы сохраните что-то в своем браузере и полетите на _____, то оно по-прежнему будет там, когда вы вернетесь.



Игра в скорлупки. Решение

Готовы испытать удачу (или, скорее, сноровку)? Данная игра позволит проверить, насколько хорошо вы знаете localStorage, однако вам потребуется проявить решительность. Используйте свои знания об извлечении и сохранении пар «ключ — значение» в localStorage, чтобы уследить за горошиной, когда она будет перемещаться от одной скорлупки к другой. Вот наше решение этого задания.

```
function shellGame() {
    localStorage.setItem("shell1", "pea");
    localStorage.setItem("shell2", "empty");
    localStorage.setItem("shell3", "empty");
    localStorage["shell1"] = "empty";
    localStorage["shell2"] = "pea";
    localStorage["shell3"] = "empty";
    var value = localStorage.getItem("shell2");
    localStorage.setItem("shell1", value);
    value = localStorage.getItem("shell3");
    localStorage["shell2"] = value;
    var key = "shell2";
    localStorage[key] = "pea";
    key = "shell1";
    localStorage[key] = "empty";
    key = "shell3";
    localStorage[key] = "empty";

    for (var i = 0; i < localStorage.length; i++) {
        var key = localStorage.key(i);
        var value = localStorage.getItem(key);
        alert(key + ": " + value);
    }
}
```

Под какой скорлупкой находится горошина ("pea")?

Ключ	Значение
shell1	empty
shell2	pea
shell3	empty

Горошина ("pea") находится под скорлупкой 2 ("shell2")..



Упражнение Решение

Ваша задача заключалась в том, чтобы обновить весь код таким образом, чтобы везде, где мы стали бы вызывать addStickyToDOM, мы передавали ключ, а также значение.

Вам следовало обновить все вызовы addStickyToDOM в init и createSticky, чтобы они выглядели следующим образом:

```
addStickyToDOM(key, value);
```

Возьми в руку карандаш



Решение

Отметьте флажками, какие проблемы может вызывать наша текущая реализация:

- Отображение клейких заметок будет происходить неэффективно, если в `localStorage` присутствует большое количество элементов, которые не являются заметками.
- Клейкая заметка может быть перезаписана методом `setItem`, если объем `localStorage` уменьшится, несмотря на то что другое приложение удалит все свои элементы.
- Сложно быстро сказать, сколько имеется клейких заметок; вам придется осуществлять итерацию по каждому элементу в `localStorage`, чтобы извлечь все заметки.
- Используйте файлы `cookie`, поскольку такой подход должен оказаться проще!



Иллюстрация
Решение

Нам по-прежнему нужно выяснить, как фактически осуществляется сохранение массива в `localStorage`

Вы уже могли догадаться, что у нас есть возможность использовать JSON для создания строкового представления массива. И если так и было, то вы правы. Располагая таким представлением, вы сможете сохранить его в `localStorage`.

Как вы помните, в API-интерфейсе JSON имеются только два метода: `stringify` и `parse`. Задействуем данные методы и завершим функцию `init`:

Извлекаем массив из `localStorage`.

Если в `localStorage` не окажется массива, то мы создадим пустой массив и присвоим его переменной `stickiesArray`.
В данный момент переменная `stickiesArray` будет строкой.

```
function init() {
  // здесь будет код, касающийся button...
  var stickiesArray = localStorage["stickiesArray"];
  if (!stickiesArray) {
    stickiesArray = [];
    localStorage.setItem("stickiesArray", JSON.stringify(stickiesArray));
  } else {
    stickiesArray = JSON.parse(stickiesArray);
  }
  for (var i = 0; i < stickiesArray.length; i++) {
    var key = stickiesArray[i];
    var value = localStorage[key];
    addStickyToDOM(value);
  }
}
```

Если нам придется создать массив, то мы воспользуемся `JSON.stringify` для генерирования строкового представления массива, а затем сохраним его...

Если массив `stickiesArray` уже будет сохранен в `localStorage` (в виде строки), нам потребуется преобразовать его, используя метод `parse` API-интерфейса JSON. После этого в нашем распоряжении окажется массив ключей, присвоенный переменной `stickiesArray`.

Чтобы вам все было ясно: мы берем строку, на которую указывает `stickiesArray`, преобразовываем ее в массив с помощью метода `parse`, а затем снова присваиваем этот массив переменной `stickiesArray`.

НЕ ПЫТАЙТЕСЬ СДЕЛАТЬ ЭТО ДОМА (ИЛИ КАК РАЗНЕСТИ В ПУХ И ПРАХ ВАШИ 5 МБАЙТ)

Как мы уже говорили, в сумме у вас будет 5 Мбайт хранилища для браузера каждого из пользователей. Несмотря на то что такой объем может показаться большим, все ваши данные сохраняются в виде строк, а не в байтовом формате. Возьмите, к примеру, размер государственного долга: если выразить его в виде значения с плавающей точкой, то для его размещения в хранилище потребуется не много места, однако если выразить его в виде строкового значения, то для его сохранения потребуется гораздо больший объем. Таким образом, хранилище размером 5 Мбайт способно вместить не так много, как могло показаться.

Так что же произойдет, когда вы исчерпаете 5 Мбайт? К сожалению, это одно из поведений, которые не определяются спецификацией HTML5, и браузеры могут поступать по-разному, когда вы превысите свой лимит. Браузер может спросить у вас, хотите ли вы увеличить объем хранилища, либо сгенерирует исключение `QUOTA_EXCEEDED_ERR`, которое можно перехватить следующим образом:

try/catch перехватывает любые исключения, генерируемые в блоке try.

```
try {
    localStorage.setItem(myKey, myValue);
} catch(e) {
    if (e == QUOTA_EXCEEDED_ERR) {
        alert("Out of storage!");
    }
}
```

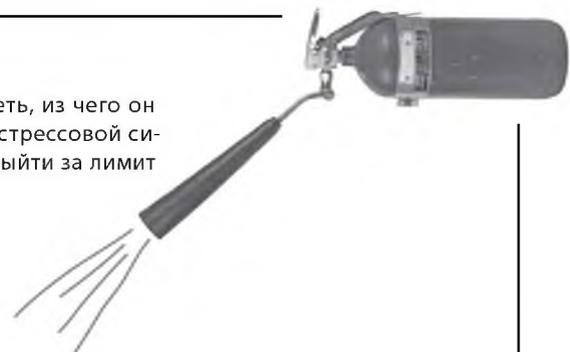
Вот вызов setItem в середине блока try; если что-нибудь пойдет не так и setItem сгенерирует исключение, то будет задействован блок catch.

Проверяем, ошибка ли это квоты хранилища (а не какой-то другой тип исключения). Если так оно и есть, мы выведем для пользователя диалоговое окно alert с соответствующим сообщением. Вы, скорее всего, захотите сделать что-то более значительное, чем просто вывод окна alert.

Не все браузеры в настоящий момент генерируют исключение QUOTA_EXCEEDED_ERR. Однако они все же сгенерируют исключение, когда вы превысите свой лимит, поэтому не лишним будет разобраться с общим случаем исключения при сохранении того или иного элемента в хранилище.



Давайте попробуем довести свой браузер до предела, посмотреть, из чего он сделан и как далеко может пойти, узнать, как он себя поведет в стрессовой ситуации. Напишем немного кода, который заставит ваш браузер выйти за лимит объема его хранилища:



```
<html>
<head>
<script>

localStorage.setItem("fuse", "-");
while(true) {
    var fuse = localStorage.getItem("fuse");
    try {
        localStorage.setItem("fuse", fuse + fuse);
    } catch(e) {
        alert("Your browser blew up at" + fuse.length + " with exception: " + e);
        break;
    }
}
localStorage.removeItem("fuse");
</script>
</head>
<body>
</body>
</html>
```

Начнем с односимвольной строки с ключом "fuse".

И просто будем, не останавливаясь, увеличивать ее размер...

...путем удваивания этой строки (посредством конкатенации ее с самой собой)..

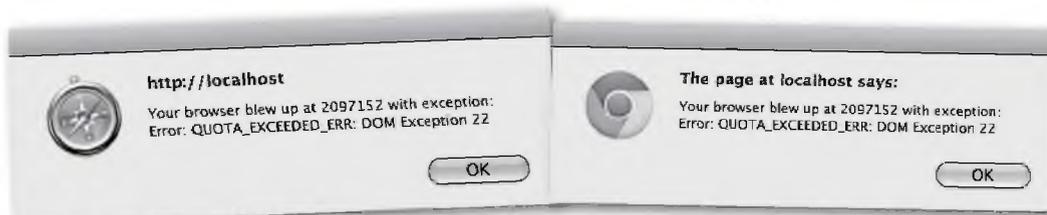
Затем попытаемся записать ее обратно в localStorage.

Если браузер аварийно завершит работу — дело сделано! Выведем для пользователя диалоговое окно alert с соответствующим сообщением и выйдем из данного цикла.

Не будем оставлять беспорядок и удалим данный элемент из localStorage.

Результаты, полученные у нас при использовании браузеров Safari и Chrome.

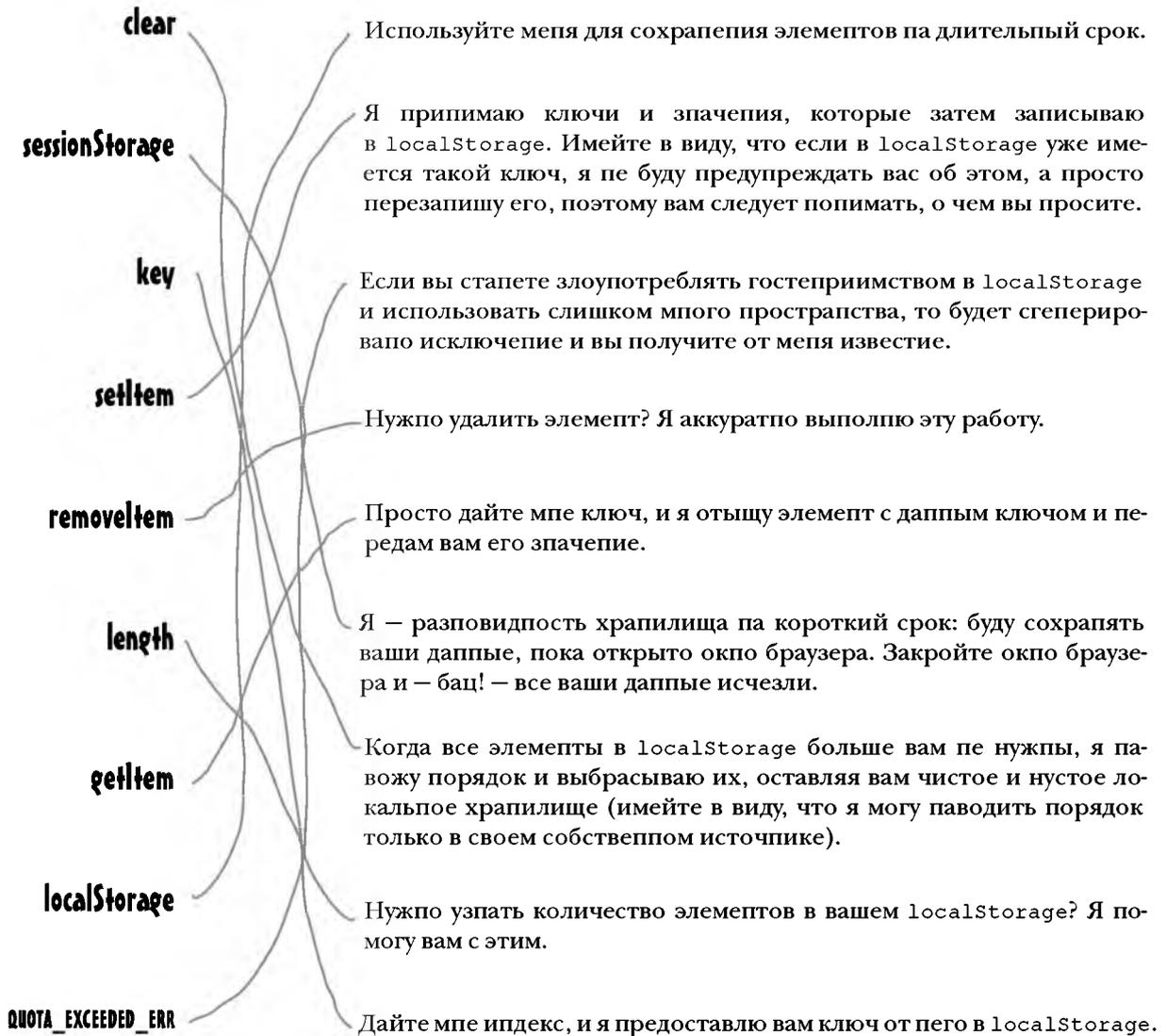
Наберите этот код, «зажгите фитиль», загрузив его, и поразвлекайтесь! Испытайте его в разных браузерах.



* КТО И ЧТО ДЕЛАЕТ? *

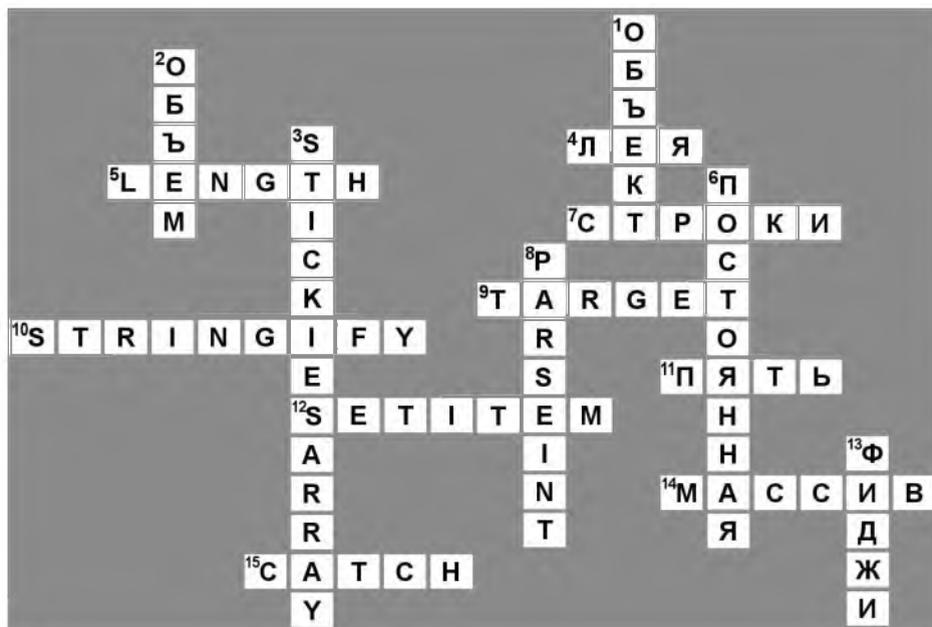
РЕШЕНИЕ

На данный момент вы полностью изучили API-интерфейс `localStorage`. Чуть ниже приведены главные действующие лица данного API-интерфейса, скрытые под масками. Посмотрите, сможете ли вы разобраться, кто из них для чего используется. В качестве примера мы соотнесли одно из описаний с нужной позицией.





HTML5-кроссворд. Решение



API-интерфейс *Web Workers*

Да, я не смогу справиться здесь со ВСЕМ, мне потребуется небольшая помощь.

Я могу помочь тебе разобраться с этой шахтой лифта.



Медленный сценарий — **хотите продолжить его выполнение?** Если вам доводилось тесно работать с JavaScript или путешествовать по Интернету, то вы, вероятно, сталкивались с диалоговым окном *Slow Script* (Медленный сценарий). Но как же сейчас, когда в компьютерах устанавливаются многоядерные процессоры, сценарии могут выполняться *слишком медленно*? Все потому, что JavaScript поддерживает выполнение только одного действия за раз. Однако с появлением HTML5 и *Web Workers* *все изменилось*. Теперь у вас есть возможность создавать *собственные* множественные JavaScript-объекты `Worker` для одновременного выполнения нескольких действий. Независимо от того, пытаетесь вы создать более отзывчивое приложение либо просто хотите по максимуму использовать возможности центрального процессора — API-интерфейс *Web Workers* придется кстати. Итак, надевайте шляпу директора предприятия под названием JavaScript и заставьте своих подчиненных `worker` попотеть!

Устрашающее диалоговое окно Slow Script (Медленный сценарий)

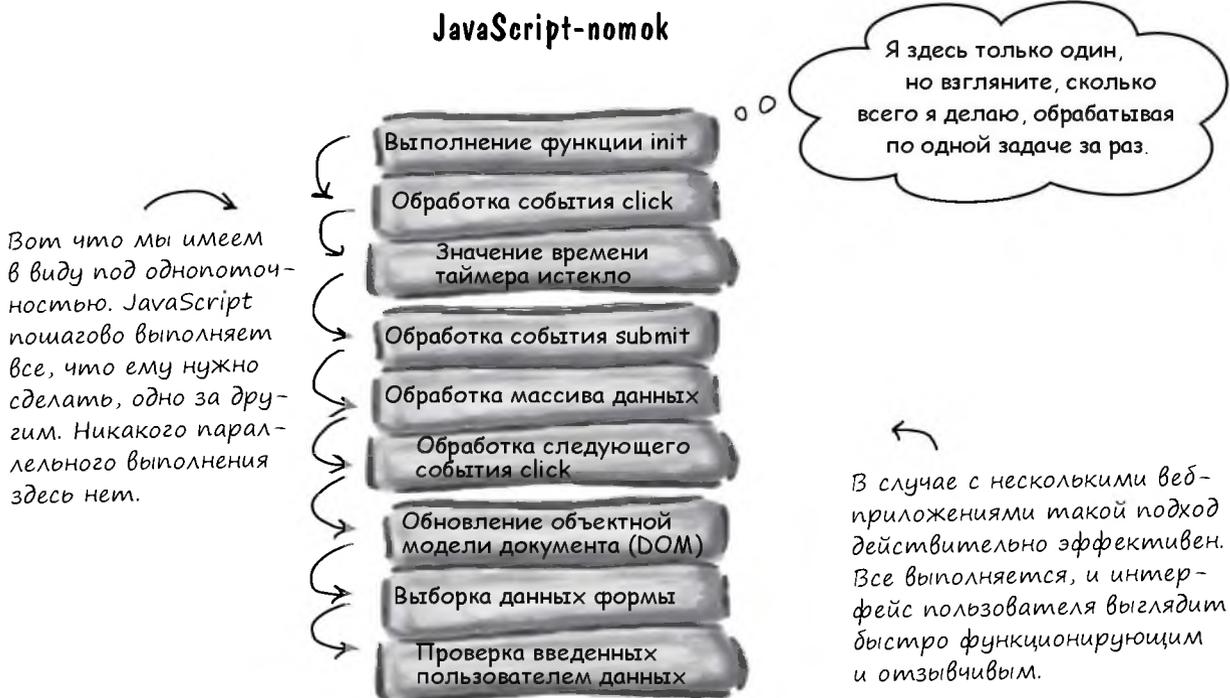
Одной из примечательных особенностей JavaScript является то, что он поддерживает выполнение только одного действия за раз — мы обычно называем это однопоточностью. Почему данная особенность примечательна? А потому, что она делает процесс программирования простым. Когда у вас имеется множество потоков, протекающих одновременно, написание корректно работающей программы может превратиться в сложную задачу.



Недостаток однопоточности заключается в том, что если взвалить на JavaScript-программу слишком много работы, она может не успевать с ней справиться, и в итоге на экране мы увидим окно Slow Script (Медленный сценарий). Другое последствие однопоточности состоит в том, что если у вас имеется JavaScript-код, которому приходится интенсивно работать, то будет оставаться не очень много вычислительных ресурсов для интерфейса пользователя или взаимодействий пользователей, и ваше приложение может показаться медленно работающим или неотзывчивым.

Как JavaScript проводит свое время

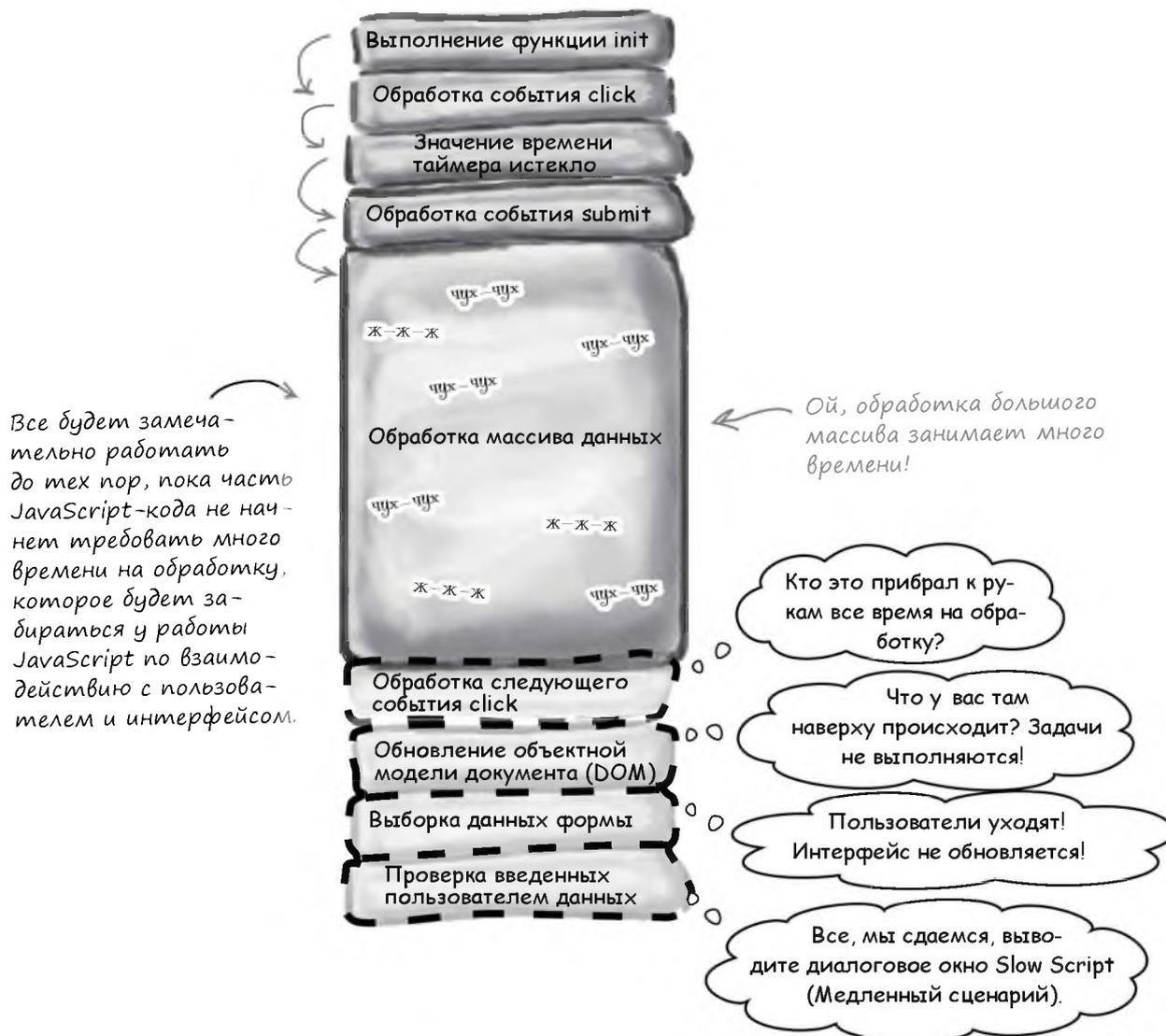
Давайте посмотрим, что это все означает, взглянув, как JavaScript обрабатывает задачи, касающиеся типичной веб-страницы:



Когда однопоточность — это ПЛОХО

Действительно, во многих ситуациях данный однопоточный режим вычислений с использованием JavaScript отлично работает и, как мы уже отмечали, делает процесс программирования простым. Однако если вы напишете код, который будет требовать настолько большого объема вычислений, что это пагубно сказывается на способностях JavaScript, однопоточная модель станет разваливаться.

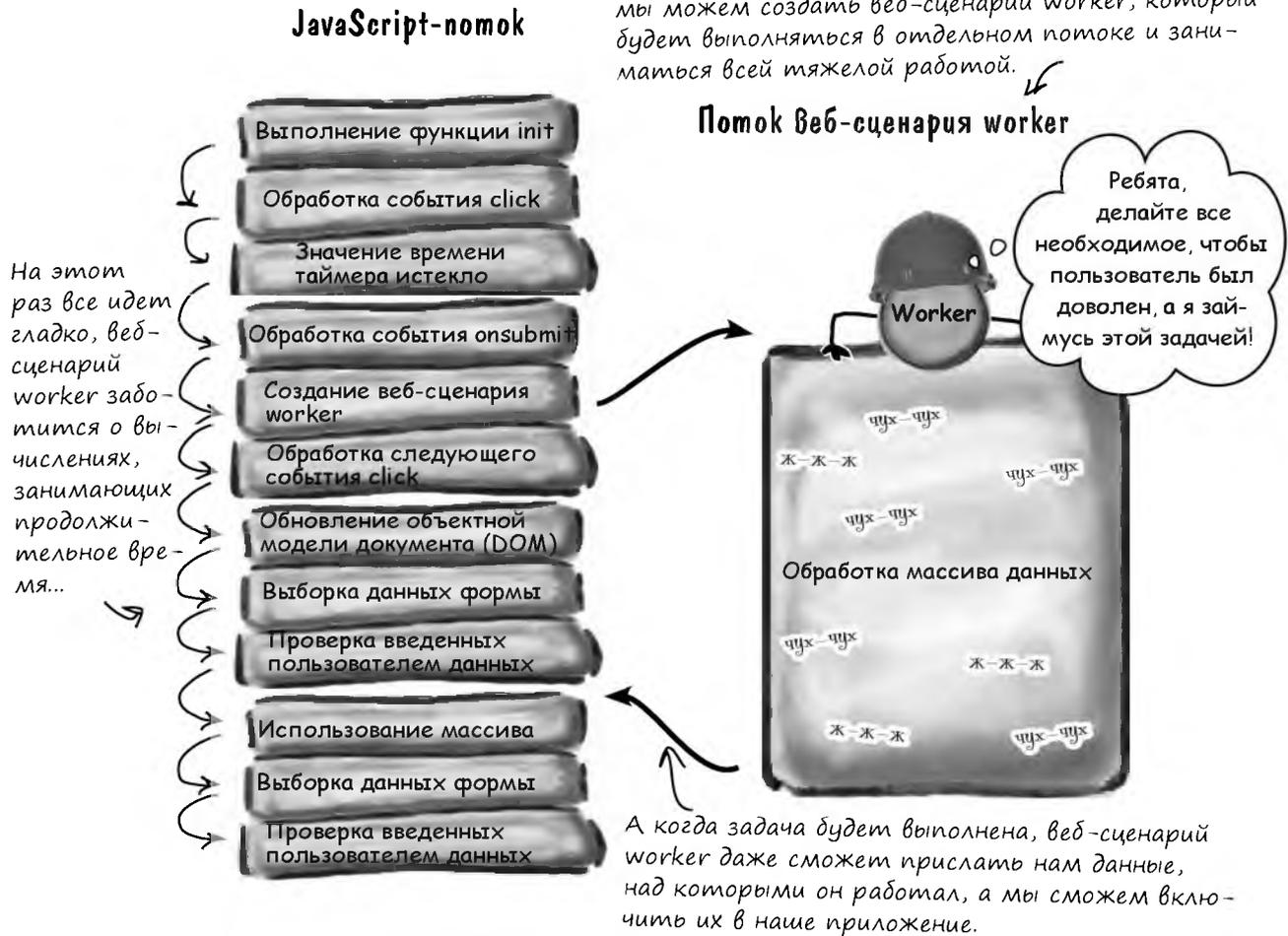
JavaScript-помок



Добавление еще одного потока управления в качестве помощника

До появления HTML5 мы придерживались подхода на основе одного потока управления для своих страниц и приложений, однако благодаря Web Workers у нас появился способ создать еще один поток управления для оказания помощи главному потоку. Таким образом, если ваш код будет отнимать много времени на обработку, вы сможете создать фоновый веб-сценарий worker (блок JavaScript-кода, выполняющийся в фоновом режиме), который займется обработкой задачи, пока главный JavaScript-поток управления будет заботиться о том, что касается браузера и пользователя.

Вместо того чтобы замедлять выполнение задач из-за большого объема вычислений JavaScript, мы можем создать веб-сценарий worker, который будет выполняться в отдельном потоке и заниматься всей тяжелой работой.



Мы особо подчеркивали важность того факта, что один поток управления обеспечивает простоту и легкость процесса программирования, и это правда. Как вы еще увидите, API-интерфейс Web Workers был тщательно проработан, чтобы все оставалось простым и надежным для программиста. Чуть позже мы узнаем насколько.



ВСТРЕЧАЕМ JAVASCRIPT (СНОВА)

Интервью недели:

Где JavaScript проводит свое время?

Head First: С возвращением, JavaScript, приятно Вас видеть.

JavaScript: Рад быть здесь, по давайте придерживаться моего графика, а то у меня еще очень много дел.

Head First: Именно на этом, как я полагал, мы и могли бы сосредоточиться во время сегодняшнего разговора. Вы стали сверхуспешным парнем, у Вас так много всего происходит — как Вам удается со всем справляться?

JavaScript: Что ж, у меня есть философия: я делаю что-то одно за раз, по делаю это по-настоящему хорошо.

Head First: Как это Вы делаете только что-то одно за раз? Как нам кажется, вы извлекаете данные, отображаете страницы, взаимодействуете с пользователем, управляете таймерами и диалоговыми окнами alert, не останавливаясь...

JavaScript: Да, я все это делаю, по в определенный момент времени только что-то одно. Таким образом, если я взаимодействую с пользователем, то этим все и будет ограничиваться до тех пор, пока я не разберусь с данной задачей.

Head First: Неужели это правда? А если значение времени таймера истечет, либо по сети поступят данные, либо произойдет что-то другое — разве Вы не остановитесь и не займетесь этим?

JavaScript: При наступлении события вроде тех, о которых вы упомянули, они будут добавлены в очередь. Я даже не взгляну на них, пока не закончу то, над чем работаю. Используя такой подход, я делаю все правильно, надежно и эффективно.

Head First: А случалось ли когда-нибудь так, что Вы с опозданием добирались до одной из тех задач в очереди?

JavaScript: О, так бывает. К счастью, я являюсь технологией, стоящей за браузерными веб-страницами, поэтому что уж такого плохого в том, если я буду немного запаздывать? Вам лучше поговорить с парнями, которым приходится выполнять код, управляющий работой двигателей космических кораблей и контроллеров атомных электростанций... Эти ребята вынуждены жить по другим правилам — вот почему они много зарабатывают.

Head First: Мне всегда было интересно узнать, что происходит, когда у меня в браузере появляется диалоговое окно с сообщением о медленном сценарии и вопросом о том, хочу ли я продолжить его выполнение. Это из-за того, что Вы берете перерыв?

JavaScript: Беру перерыв! Ха! Так бывает, когда кто-то неграмотно структурировал свою страницу, — таким образом и на меня сваливается столько работы, что я не могу с ней справиться! Если вы напишете небольшой блок JavaScript-кода, который будет отнимать все мое время, то ваше взаимодействие с пользователем пострадает. Я могу выполнять только что-то одно за раз.

Head First: Похоже, Вам требуется помощь.

JavaScript: Эта помощь мне оказывается благодаря HTML5, поскольку именно здесь в дело вступает API-интерфейс Web Workers. Если вам нужно написать код, требующий большого объема вычислений, — используйте Web Workers, чтобы снять с меня часть работы. Благодаря этому я смогу сосредоточиться на своих задачах, а веб-сценарии worker сделают за меня некоторую тяжелую работу (не мешая при этом мне).

Head First: Это интересно, мы исследуем данный аспект. А теперь следующий вопрос... Постойте-ка, он исчез. Похоже, ушел заниматься своей следующей задачей. Серьезный он парень, не правда ли?

Как работают веб-сценарии worker

Давайте взглянем на один день из жизни веб-сценариев worker: как они создаются, откуда они знают, что делать, и как они возвращают результаты вашему основному браузерному коду.

Для их использования браузеру сначала потребуется создать один веб-сценарий worker (или более), который станет помощником в решении задач. Каждый worker определяется посредством своего собственного JavaScript-файла, содержащего весь код (или ссылки на код), необходимый ему для выполнения своей работы.



Все веб-сценарии worker живут в очень ограниченном мире; у них нет доступа к множеству объектов времени выполнения, который имеется у вашего основного браузерного кода, например к объектной модели документа (DOM) или любым переменным либо функциям в коде.



Чтобы worker начал работать, браузер обычно отправляет ему сообщение. Код worker получает данное сообщение, изучает его на предмет каких-либо особых инструкций и приступает к работе.



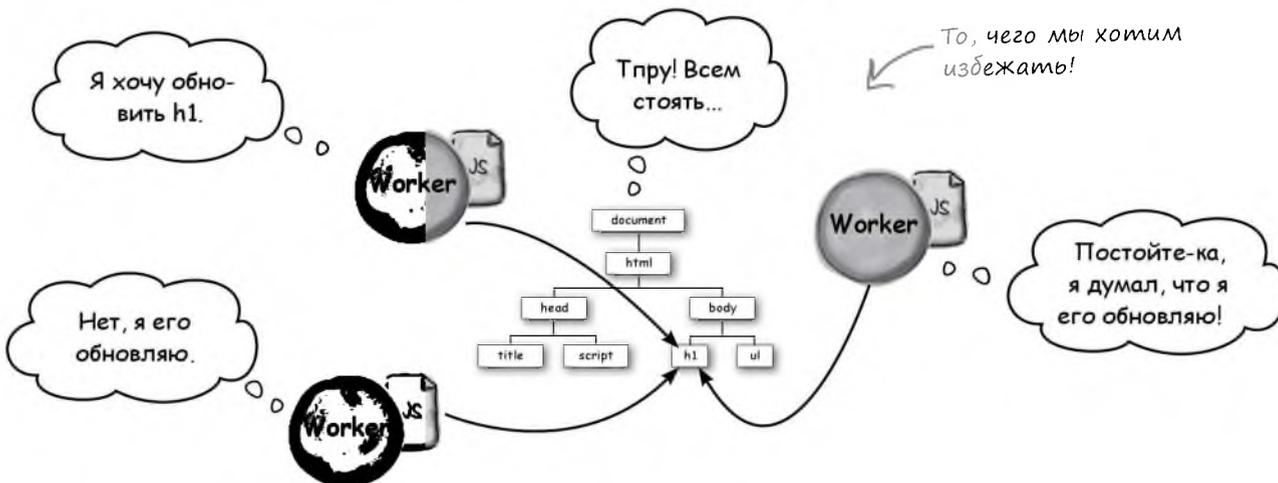
Когда worker сделает свою работу, он отправит сообщение обратно вместе с результатами того, над чем трудился. Основной браузерный код затем возьмет эти результаты и включит их в страницу определенным образом.



Почему бы не разрешить веб-сценариям worker доступ к объектной модели документа? Похоже, отправка сообщений туда-сюда доставит массу хлопот, если все веб-сценарии worker будут выполняться в одном и том же браузере.

Отсутствие у них доступа к DOM объясняется стремлением обеспечить эффективность.

Причина, по которой объектная модель документа и JavaScript являются столь успешными, заключается в том, что мы можем значительно оптимизировать операции с DOM, поскольку у нас имеется только один поток с доступом к DOM. Если мы позволим сразу нескольким вычислительным потокам одновременно вносить изменения в DOM, это серьезно ударит по уровню производительности (а разработчикам браузеров пришлось бы прилагать большие усилия, чтобы убедиться в безопасности внесения изменений в объектную модель документа). Честно говоря, если разрешить одновременное внесение кучи изменений в DOM, это может привести к ситуациям, когда DOM окажется в несогласованном состоянии, а это плохо. Очень плохо.



Возьми в руку карандаш



Взгляните на потенциальные варианты применения веб-сценариев `worker`, приведенные ниже. Что из перечисленного могло бы улучшить дизайн и производительность приложения?

- | | | | |
|--------------------------|--|--------------------------|--|
| <input type="checkbox"/> | Кэширование данных для использования в ваших страницах. | <input type="checkbox"/> | Проверка орфографии при вводе пользователем текста на странице. |
| <input type="checkbox"/> | Обработка большого количества данных в массивах или объемных JSON-ответов, поступающих от веб-служб. | <input type="checkbox"/> | Опрос веб-служб и предупреждение основной страницы, когда что-то случается. |
| <input type="checkbox"/> | Управление подключениями к базам данных наряду с добавлением и удалением записей на основной странице. | <input type="checkbox"/> | Обработка изображений в <code>canvas</code> . |
| <input type="checkbox"/> | Автоматизированный агент для ставок на ипподроме. | <input type="checkbox"/> | Подсветка синтаксиса кода или иного текста. |
| <input type="checkbox"/> | Анализ видео. | <input type="checkbox"/> | Предварительная выборка данных в зависимости от того, что делает пользователь. |
| <input type="checkbox"/> | | <input type="checkbox"/> | |
| <input type="checkbox"/> | | <input type="checkbox"/> | |
| <input type="checkbox"/> | | <input type="checkbox"/> | |

↖
Свои идеи напишите здесь!

Перенесите в поле ответы, представляющие собой короткие варианты применения веб-сценариев `worker`, однако с парой из них все же можно поспорить: проверка орфографии и подсветка синтаксиса, возможно, лучше осуществляются в коде основной страницы, а ставка на ипподроме, может быть, вообще не стоит. :-)



Будьте осторожны!

Браузер Google Chrome предусматривает дополнительные ограничения (для обеспечения безопасности), которые не позволяют вам запускать веб-сценарии worker напрямую из файла. Если вы попытаетесь

это сделать, страница не будет работать и вы не получите никаких уведомлений о причинах такого поведения (в том числе никаких сообщений об ошибках, где будет сказано, в чем дело!).

Поэтому для приведенных здесь примеров рекомендуем вам использовать другой браузер или свой собственный сервер и запускать их с `http://localhost`. Либо можете загрузить их на онлайн-сервер, если у вас имеется к нему доступ.

Вы также можете воспользоваться переключателем времени выполнения Chrome `--allow-file-access-from-files`, однако мы рекомендуем вам использовать данный параметр только при тестировании кода.



Будьте осторожны!

Почти все современные браузеры поддерживают API-интерфейс Web Workers, но есть одно исключение — Internet Explorer 9. Хорошая новость заключается в том, что в версии 10 (и выше) этого веб-обозревателя вы можете рассчитывать на Web Workers, однако для Internet Explorer 9 и всех предшествующих версий вам придется искать альтернативное решение.

Вы можете легко проверить и узнать, поддерживает ли тот или иной браузер API-интерфейс Web Workers:

Если API-интерфейс Web Workers поддерживается, то свойство Worker будет определено в глобальном объекте window.

А если свойство Worker не определено, то такая поддержка в браузере будет отсутствовать.

```
if (window["Worker"]) {  
    var status = document.getElementById("status");  
    status.innerHTML = "Bummer, no Web Workers";  
}
```

В подобной ситуации вам придется поступить так, как будет нужно для вашего приложения. Здесь мы просто уведомляем пользователя путем размещения сообщения в элементе с `id="status"`.

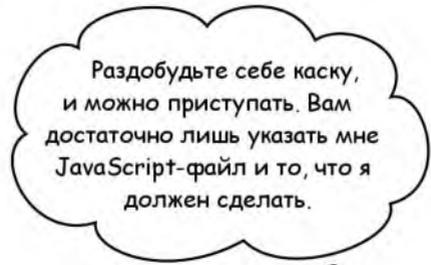
Ваш первый веб-сценарий worker...

Давайте создадим worker. Для этого нам потребуется страница, в которой все будет размещаться. Мы напишем самую простую HTML5-разметку, которой сможем обойтись в данном случае; впишите в pingpong.html то, что показано далее:

```
<!doctype html>
<html lang="en">
  <head>
    <title>Ping Pong</title>
    <meta charset="utf-8">
    <script src="manager.js"></script>
  </head>
  <body>
    <p id="output"></p>
  </body>
</html>
```

Данный JavaScript-код создаст все веб-сценарии worker и будет осуществлять управление ими.

Генерируемый веб-сценарием worker вывод мы будем размещать здесь.



Веб-сценарий worker

Как создать веб-сценарий worker

Перед началом реализации manager.js взглянем на то, как фактически создается веб-сценарий worker:

Для создания нового worker мы генерируем новый объект Worker...

```
var worker = new Worker("worker.js");
```

Присваиваем новый Worker JavaScript-переменной worker.

...JavaScript-файл worker.js будет содержать код для worker.

Вот как создается один worker. Однако вам необязательно на этом останавливаться — можете создать столько worker, сколько нужно:

```
var worker2 = new Worker("worker.js");
var worker3 = new Worker("worker.js");
```

Мы можем легко создать два дополнительных worker, которые будут использовать тот же код, что и наш первый worker.

```
var another_worker = new Worker("another_worker.js");
```

Либо мы можем создать дополнительные worker, основанные на другом JavaScript-файле.

Чуть позже мы посмотрим, как использовать вместе сразу несколько worker...

Написание manager.js

Теперь, когда вы знаете, как создать worker (и насколько легко это делается), поработаем над кодом manager.js. Он будет простым, и сейчас мы сформируем только один worker. Создайте файл с именем manager.js и добавьте в него следующий код:

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
}
```

Мыждемся полной загрузкой страницы.
А затем создадим новый worker.

Это отличный старт, однако теперь нам необходимо сделать так, чтобы worker занялся работой. Как уже отмечалось ранее, чтобы заставить worker что-то сделать, нужно отправить ему сообщение. Для этого мы воспользуемся методом postMessage объекта worker. Вот как он применяется:

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
  
    worker.postMessage("ping");  
}
```

Метод postMessage определен для вас в API-интерфейсе Web Workers.

Используем метод postMessage объекта worker для отправки ему сообщения. Наше сообщение представляет собой простую строку "ping".

Хотите отправлять более сложные сообщения? Вот как это делается...



postMessage под увеличительным стеклом

Вы сможете отправлять нечто большее, чем просто строки, используя postMessage. Давайте взглянем, что можно отправлять в сообщении:

```
worker.postMessage("ping"); // Вы можете отправлять строку...  
worker.postMessage([1, 2, 3, 5, 11]); // ...массив...  
worker.postMessage({"message": "ping", "count": 5}); // ...или даже JSON-объект.
```

Отправлять функции **нельзя**:

```
worker.postMessage(updateTheDOM);
```

Отправлять ту или иную функцию нельзя — она может содержать ссылку на объектную модель документа, позволяя worker вносить изменения в DOM!

Получение сообщений от веб-сценария worker

Мы пока не совсем довольны кодом `manager.js` — нам еще нужна возможность получать сообщения от `worker`, если мы собираемся всецело использовать его тяжелый труд. Чтобы получить сообщение от `worker`, потребуется определить обработчик для свойства `worker` с именем `onmessage`, чтобы каждый раз при поступлении сообщения от `worker` происходил вызов нашего обработчика (и передача ему сообщения). Вот что мы сделаем:

```

window.onload = function() {
  var worker = new Worker("worker.js");

  worker.postMessage("ping");
}

```

Определяем функцию, которая будет вызываться всякий раз при получении сообщения от данного `worker`. Сообщение от `worker` будет обернуто в объект `event`.

```

worker.onmessage = function (event) {
  var message = "Worker says " + event.data;
  document.getElementById("output").innerHTML = message;
};

```

Объект `event`, передаваемый нашему обработчику, обладает свойством `data`, содержащим данные сообщения (нужные нам), которые отправились от `worker`.

При получении сообщения от `worker` мы поместим его в элемент `<p>` в HTML-странице.



`onmessage` под увеличительным стеклом

Кратко рассмотрим сообщение, которое наш обработчик `onmessage` получает от `worker`. Как мы уже говорили, данное сообщение будет обернуто в объект `event`, который обладает двумя интересующими нас свойствами: `data` и `target`:

```

worker.onmessage = function (event) {
  var message = event.data;
  var worker = event.target;
}

```

Это объект, который `worker` посылает коду в вашей странице при отправке сообщения.

Свойство `data` содержит сообщение, которое отправил `worker` (например, строку вроде "pong").

`target` — это ссылка на `worker`, который отправил сообщение. Данное свойство пригодится кстати, когда вам потребуется узнать, от какого `worker` исходит сообщение. Мы будем использовать его далее в главе.

А теперь напишем worker

Приступая к написанию worker, первым делом нам необходимо позаботиться о том, чтобы наш worker смог принимать сообщения, которые поступают от `manager.js`, — так worker будет получать задания по работе. Для этого нам также потребуется задействовать обработчик `onmessage`, который будет в самом worker. Каждый worker готов к приему сообщений, вам нужно лишь дать ему обработчик для их обработки. Создайте файл `worker.js` и добавьте в него приведенный далее код:

```
onmessage = pingPong;
```

↑
Присваиваем свойство `onmessage` в worker функции `pingPong`.

↑
Мы собираемся написать функцию `pingPong` для обработки всех поступающих сообщений.

Написание обработчика сообщений для worker

Напишем обработчик сообщений `pingPong` для worker. Поначалу все будет просто. Вот что будет происходить (вы уже могли об этом догадаться, взглянув на имя `pingPong`): worker будет проверять любое получаемое им сообщение, чтобы убедиться в том, что оно содержит строку "ping", и если эта строка присутствует, то мы отправим назад сообщение со строкой "pong". Таким образом, работа worker в действительности будет заключаться лишь в приеме "ping" и отправке ответа в виде "pong" — мы не собираемся производить какие-либо интеллигентные вычисления, а просто убедимся в том, что `manager.js` и worker общаются друг с другом. Если же сообщение не будет содержать строку "ping", мы просто проигнорируем его.

Таким образом, функция `pingPong` будет принимать сообщение и отвечать на него "pong". Добавьте приведенный далее код в `worker.js`:

```
onmessage = pingPong;
```

```
function pingPong(event) {  
  if (event.data == "ping") {  
    postMessage("pong");  
  }  
}
```

↑
Когда worker получит сообщение от основного кода, произойдет вызов функции `pingPong`, которой будет передано данное сообщение.

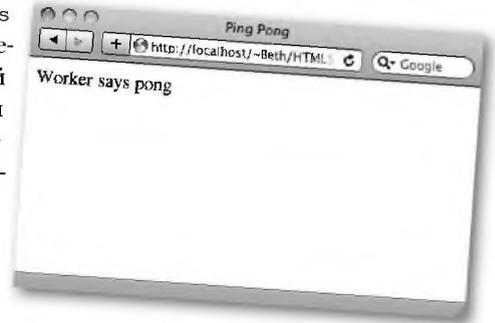
←
Если сообщение будет содержать строку "ping", мы отправим назад сообщение со строкой "pong". Ответное сообщение worker отправится коду, создавшему данный worker.

↑
Обратите внимание, что worker тоже использует `postMessage` для отправки сообщений.

Проведение тест-драйва



Убедитесь в том, что вы пабрали и сохранили весь необходимый код в `pingpong.html`, `manager.js` и `worker.js`. Теперь держите эти файлы открытыми, чтобы можно было заглядывать в них, и давайте задумаемся пад тем, как все работает. Сначала `manager.js` геперирует повый `worker`, присваивает ему обработчик сообщений, а затем отправляет этому `worker` сообщение со строкой "ping". В свою очередь, `worker` убеждается в том, что функция `pingPong` заапа в качестве его обработчика сообщений, после чего пачипает ждать. В какой-то момент `worker` получает сообщение от `manager.js` и проверяет его па предмет содержания строки "ping", которая и будет в нем присутствовать. Затем `worker` выполняет массу совсем пемного тяжелой работы и отправляет в ответ сообщение со строкой "pong".



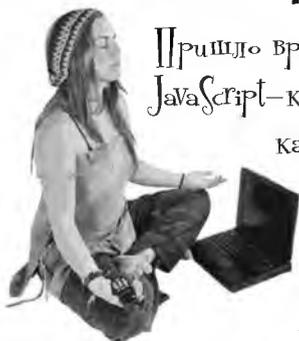
В этот момент оспвной браузерный код получает сообщение от `worker`, которое передает обработчику сообщений. Обработчик затем просто добавляет "Worker says " перед этим сообщением и выводит его па экран.

Итак, проводимые нами вычисления говорят о том, что па странице должна появиться фраза "Worker says pong"... Ладно, ладно, мы понимаем, что вы больше пе можете выпосить папряжеппого ожидания. Загрузите же, пакопец, эту страницу!

Постойте-ка, просто думая наперед... Если нам когда-либо придется создать более одного `worker`, любящего играть в такой «пинг-понг», то мне действительно придется попотеть.



СТАНЬ браузером



Пришло время притвориться браузером, оценивающим JavaScript-код. Попробуйте себя в роли браузера для каждого блока кода, приведенного ниже, и напишите генерируемый им вывод на строках справа. Можете считать, что данный код использует тот же файл `worker.js`, который мы только что написали.

← Свои ответы вы сможете проверить в решении к этому заданию в конце главы.

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
    }  
    for (var i = 0; i < 5; i++) {  
        worker.postMessage("ping");  
    }  
}
```

```
window.onload = function() {  
    var worker = new Worker("worker.js");  
    worker.onmessage = function(event) {  
        alert("Worker says " + event.data);  
    }  
    for(var i = 5; i > 0; i--) {  
        worker.postMessage("pong");  
    }  
}
```

```

window.onload = function() {
    var worker = new Worker("worker.js"); .....
    worker.onmessage = function(event) { .....
        alert("Worker says " + event.data); .....
        worker.postMessage("ping"); .....
    } .....
    worker.postMessage("ping");
}

```



Будете осторожны с этими блоками кода. Возможно, вам придется принудительно завершить работу браузера, чтобы прекратить их выполнение...



```

window.onload = function() {
    var worker = new Worker("worker.js"); .....
    worker.onmessage = function(event) { .....
        alert("Worker says " + event.data); .....
    } .....

    setInterval(pinger, 1000); .....

    function pinger() {
        worker.postMessage("ping");
    }
}

```

Возьми в руку карандаш



Несмотря на то что обычно веб-сценарии `worker` получают рабочие задания посредством сообщений, такой подход вовсе не обязателен. Взгляните на данный отличный и компактный способ сделать необходимую работу с помощью веб-сценариев `worker` и HTML. Когда вы разберетесь в том, что делает данный код, опишите это внизу. Свой ответ вы сможете проверить в решении к этому упражнению в конце главы.

```

<!doctype html>      quote.html
<html lang="en">      ←
  <head>
    <title>Quote</title>
    <meta charset="utf-8">
  </head>
<body>
  <p id="quote"></p>
  <script>
    var worker = new Worker("quote.js");
    worker.onmessage = function(event) {
      document.getElementById("quote").innerHTML = event.data;
    }
  </script>
</body>
</html>

```

quote.js
↓

```

var quotes = ["I hope life isn't a joke, because I don't get it.",
              "There is a light at the end of every tunnel... just pray it's not a train!",
              "Do you believe in love at first sight or should I walk by again?"];
var index = Math.floor(Math.random() * quotes.length);
postMessage(quotes[index]);

```

Попробуйте ввести и выполнить данный код!

Свое описание приведите здесь:

.....

.....

.....



Упражнение

Давайте добавим несколько worker в нашу игру pingPong. Ваша задача заключается в том, чтобы устранить имеющиеся пробелы и завершить приведенный внизу код, чтобы в результате происходила отправка трех сообщений со строкой "ping" веб-сценариям worker, а в ответ от них поступали три сообщения со строкой "pong".

Устраните пробелы, заполнив их соответствующим кодом.

```

window.onload = function() {
    var numWorkers = 3;
    var workers = [];
    for (var i = 0; i < .....; i++) {
        var worker = new .....("worker.js");
        worker..... = function(event) {
            alert(event.target + " says "
                + event.....);
        };
        workers.push(worker);
    }
    for (var i = 0; i < .....; i++) {
        workers[i].....("ping");
    }
}
    
```

Мы создаем три worker и сохраняем их в массиве workers.

Здесь мы добавляем новый worker в массив workers.

Часто задаваемые вопросы

В: Можно ли просто передать функцию вместо JavaScript-файла при создании worker? Наверняка так было бы проще и это лучше соответствовало бы стандартному поведению JavaScript.

О: Нет, нельзя. И вот почему: как вы знаете, одно из требований, касающихся веб-сценариев worker, заключается в том, что они не должны иметь доступа к объектной модели документа (или к любому состоянию главного браузерного потока). Если бы можно было передать конструктору Worker функцию и при этом оказалось бы, что ваша функция содержит ссылку на DOM или другие части основного JavaScript-кода, это нарушило бы данное требование. Таким образом, разработчики API-интерфейса Web Workers предпочли сделать так, чтобы вы передавали URL-адрес JavaScript-файла во избежание данной проблемы.

В: Если отправить веб-сценарию worker объект в сообщении, станет ли он объектом, совместно используемым основной страницей и этим worker?

О: Нет, когда вы отправляете объект веб-сценарию worker, он получает его копию. Любые изменения, который вносит worker, не затронут объект в вашей основной странице. worker выполняется в среде, отличной от вашей основной страницы, поэтому у вас не будет доступа к находящимся там объектам. Аналогичным образом дело обстоит и с объектами, которые worker отправляет вам: вы будете получать их копии.

В: А есть ли у веб-сценариев worker доступ к localStorage либо возможность совершать запросы с использованием XMLHttpRequest?

О: Да, у веб-сценариев worker имеется доступ к localStorage и возможность совершать запросы с использованием XMLHttpRequest.



Упражнение
Решение

Давайте добавим несколько worker в нашу игру pingPong. Ваша задача заключается в том, чтобы устранить имеющиеся пробелы и завершить приведенный внизу код, чтобы в результате происходила отправка трех сообщений со строкой "ping" веб-сценариям worker, а в ответ от них поступали три сообщения со строкой "pong". Вот наше решение этого упражнения.

Мы используем numWorkers для трехкратного совершения итерации и создания трех worker (вы можете свободно изменить значение данной переменной, чтобы их было больше!).

```

window.onload = function() {
  var numWorkers = 3;
  var workers = [];
  for (var i = 0; i < numWorkers; i++) {
    var worker = new Worker("worker.js");
    worker.onmessage = function(event) {

      alert(event.target + " says "
            + event.data);

    };
    workers.push(worker);
  }
  for (var i = 0; i < workers.length; i++) {
    workers[i].postMessage("ping");
  }
}

```

Мы задали обработчик сообщений в коде нашей основной страницы, используя свойство worker с именем onmessage.

Используем свойство data для извлечения содержимого сообщения.

Вы также можете использовать здесь numWorkers, если захотите.

Отправляем веб-сценарию worker сообщение со строкой «ping» с помощью postMessage.

Следует отметить, что в коде worker никаких изменений не потребуется. Каждый worker с радостью будет делать свою работу независимо от других.

Данное диалоговое окно alert появится на экране трижды.





Интересно, как включить дополнительные JavaScript-файлы в мой worker? У меня имеется несколько связанных с финансами библиотек, которые я хотел бы использовать, однако в результате их копирования и вставки в worker получится огромный файл, который не очень удобен в сопровождении.

Вам стоит взглянуть на importScripts.

API-интерфейс Web Workers включает глобальную функцию `importScripts`, которую вы можете использовать для импорта одного или более JavaScript-файлов в ваш worker. Для использования функции `importScripts` вам нужно просто передать ей список разделенных запятыми файлов либо URL-адресов, которые вы хотите импортировать:

```
importScripts("http://bigscience.org/nuclear.js",
              "http://nasa.gov/rocket.js",
              "mylibs/atomsmasher.js");
```

Поместите в `importScripts` нужное количество (или нуль) разделенных запятыми URL-адресов JavaScript-файлов.

Затем, когда функция `importScripts` будет вызвана, каждый из URL-адресов JavaScript-файлов будет извлекаться и оцениваться по порядку.

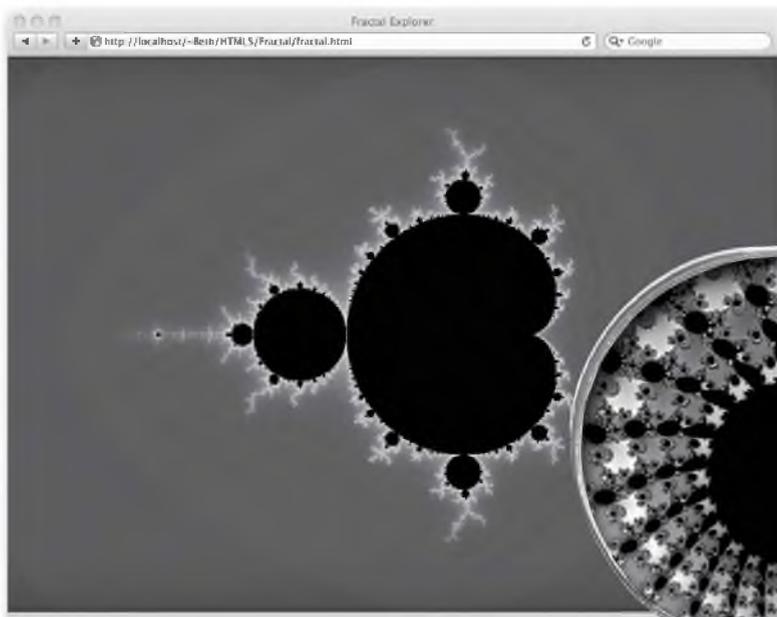
Следует отметить, что `importScripts` является полноценной функцией, в силу чего (в отличие от операторов `import` во многих языках) вы сможете принимать решения об импорте во время выполнения, как показано далее:

```
if (taskType == "songdetection") {
    importScripts("audio.js");
}
```

Поскольку `importScripts` — функция, вы сможете импортировать код по мере того, как этого будет требовать определенная задача.

Захват виртуальных земель

Исследователи множества Мандельброта уже захватили области виртуальной сельской местности и дали им названия вроде красивых «Долина морских коньков» (*Seahorse Valley*), «Радужные острова» (*Rainbow Islands*) или устрашающего «Черная дыра» (*Black Hole*). А учитывая сегодняшние цены на реальную недвижимость, ноже, что единственный простор остался в виртуальном пространстве. Поэтому мы собираемся создать приложение для множества Мандельброта под названием Fractal Explorer, чтобы поучаствовать в этом действе. Вообще-то мы должны признаться, что уже создали приложение, однако оно медленно работает — перемещение по всему множеству Мандельброта может занимать очень долгое время, поэтому надеемся, что вместе сможем ускорить его. Как мы подозреваем, решением данной проблемы может стать API-интерфейс Web Workers.



Не хотели бы приобрести дом с видом на пляж у самого края «Лазурного водоворота»?



Осмотритесь

Посетите страницу <http://wickedlysmart.com/hfhtml5/chapter10/singlethread/fractal.html>, чтобы увидеть визуализацию множества Мандельброта. Щелкните в любом месте кнопкой мыши — и вы увеличите соответствующую область карты. Продолжайте щелкать, чтобы исследовать различные области, либо перезагрузите страницу и начните все сначала. Остерегайтесь областей с черными дырами, поскольку они будут пытаться затянуть вас. На наш взгляд, хоть пейзажи и выглядят превосходно, программа просмотра могла бы работать немного быстрее... Как вы считаете? Кроме того, было бы здорово, если бы наше приложение обладало такой производительностью, чтобы можно было развернуть представление на все окно браузера! Давайте обеспечим все это, добавив веб-сценарии `worker` в приложение Fractal Explorer.

Мандель... что?

Если вдруг вы математик,

то вам известно, что множество Мандельброта генерируется посредством уравнения:

$$z_{n+1} = z_n^2 + c$$

и что его открыл и исследовал Бенуа Мандельброт. Вы также знаете, что это просто множество комплексных чисел (чисел с вещественной и мнимой частью), генерируемых с помощью данного уравнения.

Если же вы не математик, то наилучший способ представить себе множество Мандельброта — это считать его бесконечно сложным фрактальным изображением, под которым понимается изображение, которое можно увеличивать до любой степени и находить любопытные структуры. Взгляните на некоторые картины, которые можно отыскать путем перемещения по множеству:



Почему же оно нас так интересует? Данное множество обладает рядом любопытных свойств. Во-первых, оно генерируется посредством очень простого уравнения (оно приводилось выше), которое может быть выражено всего лишь несколькими строками кода. Во-вторых, генерирование множества Мандельброта занимает изрядное количество вычислительных циклов, что делает его отличным примером для использования в сочетании с API-интерфейсом Web Workers. И наконец, это же классная вещь, с которой можно поработать, и у нас есть замечательное приложение, чтобы закончить им книгу.



*Покойся с миром,
Бенуа Мандельброт,
скончавшийся, когда
мы писали эту книгу.
Нам повезло быть
знакомыми с таким
человеком, как ты.*

Как вычисляется множество Мандельброта

Прежде чем использовать веб-сценарии `worker`, взглянем, как обычно приходится структурировать код для вычисления множества Мандельброта. Мы не хотим уделять много внимания подробностям вычисления значений пикселей множества Мандельброта, так как уже полностью позаботились о соответствующем коде и собираемся продемонстрировать его чуть позже. А сейчас мы просто хотим, чтобы вы разобрались в общей картине:

Для вычисления множества Мандельброта мы совершаем цикл по каждой строке изображения.

```
for (i = 0; i < numberOfRows; i++) {
    var row = computeRow(i);
    drawRow(row);
}
```

И для каждой строки вычисляем пиксели.

А затем рисуем каждую строку на экране. Вы, вероятно, сможете увидеть построчную отрисовку изображения, когда запустите тестовый код в браузере.

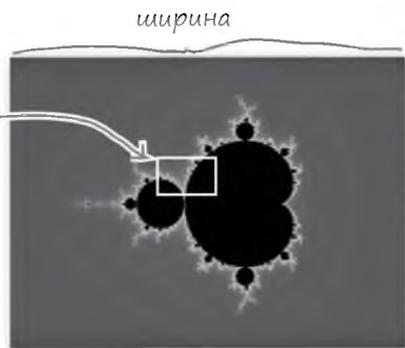
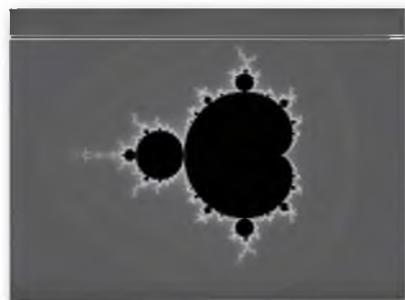
В текущий момент данный код призван быть простым псевдокодом. Когда же дело дойдет до написания кода по-настоящему, нам потребуется разобраться с дополнительными деталями: например, для вычисления строки нам потребуется знать ее ширину, коэффициент увеличения, численное разрешение, до которого мы хотим произвести ее вычисление, а также прочие мелкие детали. Мы можем собрать все эти детали в одном объекте `task`:

```
for (i = 0; i < numberOfRows; i++) {
    var taskForRow = createTaskForRow(i);
    var row = computeRow(taskForRow);
    drawRow(row);
}
```

Передаем `taskForRow` функции `computeRow`, которая возвращает вычисленную строку.

Объект `taskForRow` содержит все данные, необходимые для вычисления строки.

Следует отметить, что наша цель заключается не в том, чтобы сделать из вас специалиста по численному анализу (который умеет писать уравнения с использованием комплексных чисел), а в том, чтобы адаптировать требующее большого объема вычислений приложение к использованию веб-сценариев `worker`. Если вас интересуют численные аспекты множества Мандельброта — «Википедия» поможет начать исследование данного вопроса.



Уровень точности для вычисления



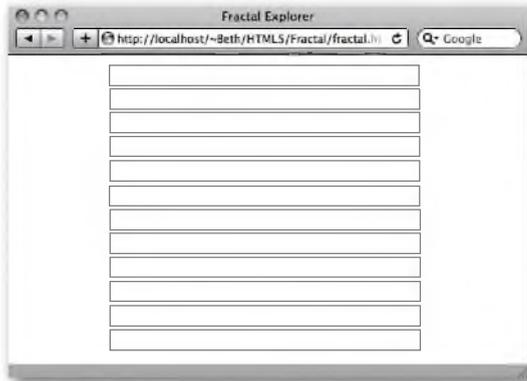
Как задействовать сразу несколько веб-сценариев worker

Вы уже знаете, как создавать новые веб-сценарии worker, однако как их использовать для чего-то более сложного, например вычисления строк множества Мандельброта? Или применения к изображению эффекта, как в Photoshop? Или генерирования сцены из видеофильма методом трассировки лучей? Во всех этих случаях мы можем разбить работу на небольшие задачи, над которыми веб-сценарии worker смогут трудиться независимо друг от друга. Мы будем придерживаться множества Мандельброта (однако шаблон, который мы собираемся использовать, может быть применен к любому из этих примеров).

Для начала браузер создает группу веб-сценариев worker в качестве помощников (но не очень много, ибо веб-сценарии worker могут дорого обойтись, если создать их слишком много, — поговорим об этом позже). В нашем примере мы используем пять worker:

Браузерный JavaScript-код создает группу worker для выполнения некоторой работы.

Новые worker прибывают на службу. Чем мы можем помочь?

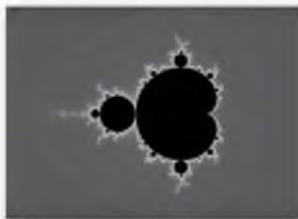


Браузер

Здесь изображение разбивается на небольшие области. Мы будем раздавать эти области веб-сценариям worker для вычисления.

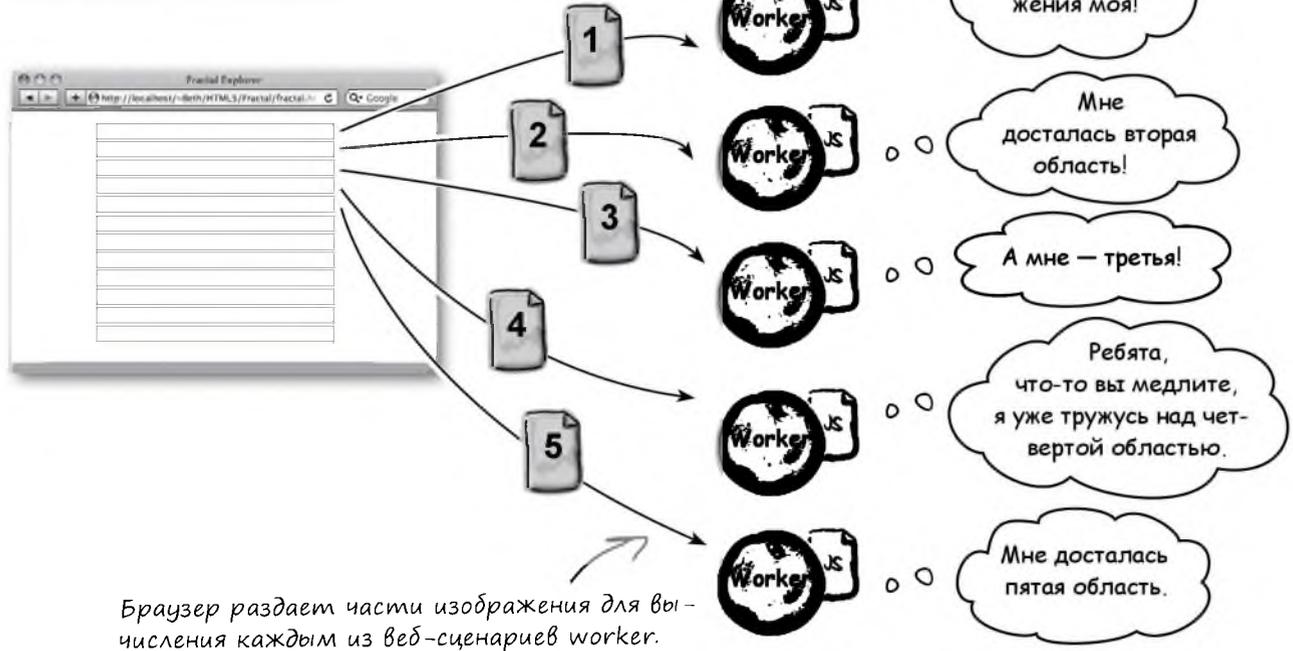


Вот результат, который мы хотим получить путем вычислений. Браузерному коду потребуются разбить работу по вычислению данного изображения на несколько небольших задач, выполнением которых займутся веб-сценарии worker.



Наши worker готовы приступить к вычислениям!

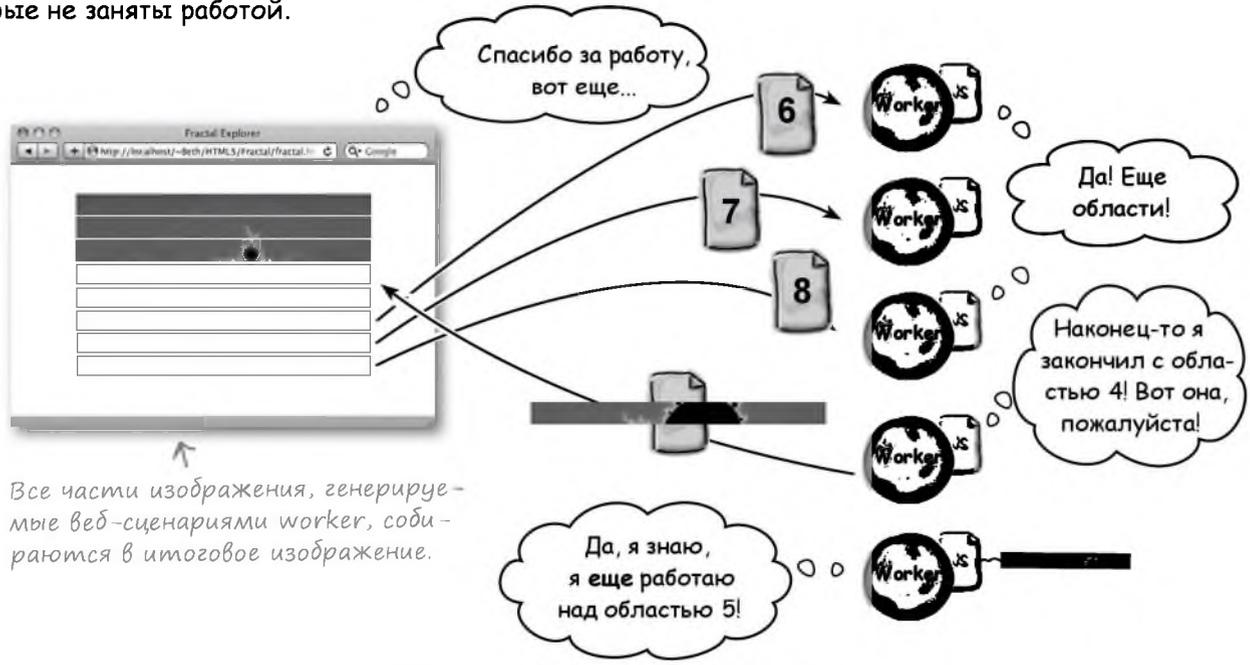
Далее браузерный код начинает раздавать разные части изображения каждому из веб-сценариев worker для вычисления:



Каждый worker работает над своей частью изображения независимо от других. Как только worker завершает выполнение своей задачи, он упаковывает результат и отправляет его обратно.



По мере того как части изображения поступают обратно от веб-сценариев worker, они собираются в единое изображение в браузере. И если еще остаются части, вычисление которых необходимо произвести, эти новые задачи поручаются веб-сценариям worker, которые не заняты работой.



Когда будет вычислена последняя часть изображения, его сборка завершится и веб-сценарии worker будут бездельничать, пока пользователь не щелкнет кнопкой мыши для увеличения изображения, после чего все начнется сначала...



Что произойдет, если я разделю работу на части и распределю их между веб-сценариями `worker`? Я хочу сказать, что у меня в компьютере останется все тот же процессор, поэтому как вычисления смогут ускориться?

Их ускорение возможно двумя путями...

Сначала представьте себе приложение, в котором выполняется масса вычислений и которое должно быть отзывчивым по отношению к пользователю. Если ваше приложение станет отнимать уйму JavaScript-времени, то пользователи столкнутся с медленной работой интерфейса, который, по их ощущениям, будет подтормаживать (онять-таки, из-за того JavaScript функционирует в однопоточном режиме). Добавление веб-сценариев `worker` в такое приложение сразу же положительно скажется на ощущениях пользователей при работе с ним. Почему? А потому, что JavaScript получит возможность реагировать на взаимодействие пользователей в промежутках между получением результатов от веб-сценариев `worker`, чего он не сможет делать, если все будет вычисляться в главном потоке. Таким образом, интерфейс пользователя станет более отзывчивым, — и пользователи будут *ощущать*, что ваше приложение работает быстрее (даже если внутри оно не будет функционировать хоть чуть-чуть быстрее). Не верите? Попробуйте это сделать и дайте реальным пользователям поработать с вашим приложением. А затем спросите, что они думают.

Второй путь *действительно более быстрый*. Почти все современные настольные компьютеры и мобильные устройства снабжаются многоядерными процессорами (или даже несколькими процессорами). Многоядерность подразумевает, что процессор сможет параллельно выполнять множество задач. При одном потоке управления JavaScript в браузере не задействует ваши дополнительные ядра или процессоры, они простаивают без дела. Однако если вы примените API-интерфейс Web Workers, то веб-сценарии `worker` смогут воспользоваться преимуществами выполнения на разных ядрах, и вы увидите реальное ускорение своего приложения, поскольку на него будет тратиться больше процессорных ресурсов. Если у вас многоядерный компьютер, нужно лишь подождать, и вскоре вы увидите разницу.





А может ли у меня
быть столько веб-
сценариев worker, сколько
я пожелаю?

В теории — да, но не на практике.

Веб-сценарии `worker` не предназначены для использования в большом количестве. Несмотря на то что создание `worker` выглядит простым в коде, он требует дополнительной памяти и нотока в операционной системе, что может дорого обойтись в плане времени запуска и ресурсов. Таким образом, обычно у вас будет возникать желание создать ограниченное количество веб-сценариев `worker`, которые вы станете повторно использовать по прошествии времени.

Возьмите наш пример с множеством Мандельброта: в теории вы могли бы назначить `worker`, который стал бы заниматься вычислением каждого одиночного пиксела, что, вероятно, было бы намного проще с точки зрения разработки кода. Однако, учитывая то, что веб-сценарии `worker` являются «тяжеловесными» средствами, мы никогда бы не выбрали такой подход при разработке нашего приложения. Вместо этого мы используем горстку веб-сценариев `worker` и структурируем вычисления таким образом, чтобы они задействовали их преимущества.

Давайте немного углубимся в разработку `Fractal Explorer`, а затем вернемся и поэкспериментируем с количеством веб-сценариев `worker`, чтобы разобраться, как этот показатель влияет на уровень производительности.



МОЗГОВОЙ ШТУРМ

У вас, несомненно, уже имеется масса знаний о том, как создавать приложения с применением API-интерфейса Web Workers, как генерировать и использовать веб-сценарии `worker`. Вы также немного знаете о том, как решать проблемы с объемными вычислениями путем разбиения их на небольшие задачи, которые могут быть выполнены веб-сценариями `worker`, и даже чуть-чуть в курсе того, как вычисляются множества Мандельброта. Попробуйте соединить все это воедино и задумайтесь над тем, как бы вы переписали приведенный ниже псевдокод, чтобы он задействовал веб-сценарии `worker`. Сначала можете предположить, что у вас будет так много веб-сценариев `worker`, как вам потребуется (например, по одному `worker` на каждую одиночную строку), а затем введите ограничение на количество `worker` (число веб-сценариев `worker` меньше, чем количество строк):

```
for (i = 0; i < numberOfRows; i++) {  
    var taskForRow = createTaskForRow(i);  
    var row = computeRow(taskForRow);  
    drawRow(row);  
}
```

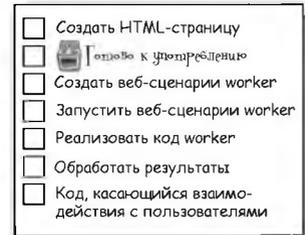
Вот наш псевдокод. Что необходимо сделать для того, чтобы добавить сюда веб-сценарии `worker`?

Свои заметки напишите здесь:

Займемся созданием приложения Fractal Explorer

Вот что нам потребуется сделать:

- Создать HTML-страницу.
- Ввести весь **готовый к употреблению код** (или загрузить его на свой компьютер).
- Создать веб-сценарии `worker` и раздать им задачи для выполнения.
- Запустить веб-сценарии `worker`, чтобы они выполняли свои задачи.
- Реализовать код `worker`.
- Обработать результаты веб-сценариев `worker`, когда они закончат выполнять свои задачи.
- Обработать события `click` и `resize` в интерфейсе пользователя.



Создание HTML-разметки для приложения Fractal Explorer

Сначала нужно сформировать HTML-страницу для размещения нашего приложения. Нам потребуется создать файл `fractal.html` и добавить туда приведенную ниже разметку.

```

<!doctype html>
<html lang="en">
  <head>
    <title>Fractal Explorer</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="fractal.css">
    <script src="mandellib.js"></script>
    <script src="mandel.js"></script>
  </head>
  <body>
    <canvas id="fractal" width="800" height="600"></canvas>
  </body>
</html>
    
```

Данный код будет размещаться в файле `fractal.html`.

← Как обычно, стандартный HTML5-файл.

Вот весь готовый к употреблению код, куда целиком входит числовой код, а также код для обработки графики.

А вот тут будет JavaScript-код, который мы собираемся написать...

Если вам интересно, где будет размещаться код `worker`, то помните, что мы не станем указывать прямую ссылку на JavaScript-файл `worker`, а будем ссылаться на данный файл, когда создадим `worker` в коде.

← Посмотрите-ка! Наш друг `<canvas>` вернулся!

`<body>` включает элемент `<canvas>`. Мы задали для него исходный размер 800 × 600 пикселей, однако позже мы займемся изменением его размера с помощью JavaScript. В конце концов, нам необходимо такое большое множество Мандельброта, какое мы только сможем получить!



Готово к употреблению

Напоминание: вы можете скачать весь код по адресу <http://wickedlysmart.com/hfhtml5>

Должны признаться, что планировали отвести целую главу чудесам вычислений множества Мандельброта... Мы собирались детально вам их объяснить, в том числе рассказать историю Бенуа Мандельброта, поведать о том, как он открыл множество, о его удивительных свойствах, пиксельных оптимизациях, цветовых картах и т. д., но после звонка от нашего редактора... Ну вы сами понимаете, какой это был ЗВОНОК. Мы немного запаздывали с работой над данной книгой, поэтому приносим свои извинения, однако все же дадим вам **готовый к употреблению код** для осуществления низкоуровневых вычислений графики, касающейся множества Мандельброта. Однако есть и положительный момент: мы сможем сосредоточиться на использовании API-интерфейса *Web Workers*, не тратя время на математику и графику.

Сначала взглянем на код, используемый для управления задачами и рисования строк фрактальных изображений. Введите данный код и сохраните его в файле `mandellib.js`.

```
var canvas;
var ctx;
```

← Обратите внимание, что здесь присутствуют наши `canvas` и `context`.

```
var i_max = 1.5;
var i_min = -1.5;
var r_min = -2.5;
var r_max = 1.5;
```

← Глобальные переменные, используемые для вычисления множества Мандельброта и его отображения.

```
var max_iter = 1024;
var escape = 1025;
var palette = [];
```

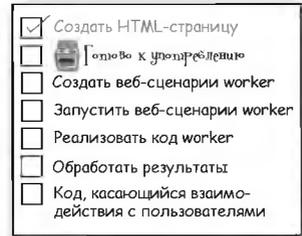
```
function createTask(row) {
    var task = {
        row: row,
        width: rowData.width,
        generation: generation,
        r_min: r_min,
        r_max: r_max,
        i: i_max + (i_min - i_max) * row / canvas.height,
        max_iter: max_iter,
        escape: escape
    };
    return task;
}
```

← Данная функция упаковывает все данные, необходимые веб-сценарию `worker` для вычисления строки пикселей, в объект. Позже вы увидите, как мы будем передавать этот объект веб-сценарию `worker` для использования.

Данный код будет размещаться в файле `mandellib.js`.



Готово к употреблению (продолжение)



```
function makePalette() {
  function wrap(x) {
    x = ((x + 256) & 0x1fff) - 256;
    if (x < 0) x = -x;
    return x;
  }
  for (i = 0; i <= this.max_iter; i++) {
    palette.push([wrap(7*i), wrap(5*i), wrap(11*i)]);
  }
}
```

makePalette преобразует большое множество чисел в массив цветов RGB. Мы будем использовать данный массив palette в drawRow (внизу) для преобразования значения, получаемого обратно от worker, в цвет для графического отображения множества (фрактального изображения).

```
function drawRow(workerResults) {
  var values = workerResults.values;
  var pixelData = rowData.data;
  for (var i = 0; i < rowData.width; i++) {
    var red = i * 4;
    var green = i * 4 + 1;
    var blue = i * 4 + 2;
    var alpha = i * 4 + 3;
    pixelData[alpha] = 255; // set alpha to opaque
    if (values[i] < 0) {
      pixelData[red] = pixelData[green] = pixelData[blue] = 0;
    } else {
      var color = this.palette[values[i]];
      pixelData[red] = color[0];
      pixelData[green] = color[1];
      pixelData[blue] = color[2];
    }
  }
}
```

Функция drawRow принимает результаты от worker и рисует соответствующие пиксели в canvas.

Для этого она использует переменную rowData; rowData — это однострочный объект imageData, содержащий фактические пиксели для соответствующей строки canvas.

Вот где мы используем palette для преобразования результата от worker (который представляет собой число) в цвет.

```
ctx.putImageData(this.rowData, 0, workerResults.row);
```

А вот где мы записываем пиксели в объект imageData в context элемента canvas!

Данный код уже должен быть вам знаком; он подобен коду, который мы применяли в главе 8 для элементов video и canvas.

Данный код будет размещаться в файле mandellib.js.



Готово к употреблению (продолжение)

```
function setupGraphics() {  
  
    canvas = document.getElementById("fractal");  
    ctx = canvas.getContext("2d");  
  
    canvas.width = window.innerWidth;  
    canvas.height = window.innerHeight;  
  
    var width = ((i_max - i_min) * canvas.width / canvas.height);  
    var r_mid = (r_max + r_min) / 2;  
    r_min = r_mid - width/2;  
    r_max = r_mid + width/2;  
  
    rowData = ctx.createImageData(canvas.width, 1);  
  
    makePalette();  
}
```

← *setUpGraphics задает глобальные переменные, используемые кодом для рисования всей графики, а также применяемые при вычислении множества Мандельброта.*

← *Вот где мы извлекаем canvas и context, а также задаем исходную ширину и высоту canvas.*

← *Переменные, используемые для вычисления множества Мандельброта.*

← *Здесь мы инициализируем переменную rowData (используемую для записи пикселей в canvas).*

← *А здесь мы инициализируем палитру цветов, которую используем для рисования множества как фрактального изображения.*

Данный код будет размещаться в файле mandellib.js.



Готово к употреблению (продолжение)

Данный **готовый к употреблению код** будет использоваться `worker` для математических вычислений множества Мандельброта. Именно здесь происходит вся магия чисел. Напечатайте и сохраните данный код в файле `workerlib.js`:

```
function computeRow(task) {
  var iter = 0;
  var c_i = task.i;
  var max_iter = task.max_iter;
  var escape = task.escape * task.escape;
  task.values = [];
  for (var i = 0; i < task.width; i++) {
    var c_r = task.r_min + (task.r_max - task.r_min) * i / task.width;
    var z_r = 0, z_i = 0;

    for (iter = 0; z_r*z_r + z_i*z_i < escape && iter < max_iter; iter++) {
      // z -> z^2 + c
      var tmp = z_r*z_r - z_i*z_i + c_r;
      z_i = 2 * z_r * z_i + c_i;
      z_r = tmp;
    }
    if (iter == max_iter) {
      iter = -1;
    }
    task.values.push(iter);
  }
  return task;
}
```

`computeRow` вычисляет одну строку данных множества Мандельброта. Этой функции передается объект, в который упакованы все значения, необходимые ей для вычисления данной строки.

Обратите внимание, что в случае с каждой строкой отображения мы совершаем два цикла — один для каждого пиксела в строке... ← Это масса вычислений. Хорошо!

...и еще один цикл, чтобы отыскать соответствующее значение для данного пиксела. В этом внутреннем цикле и заключается вычислительная сложность, и именно поэтому код будет выполняться намного быстрее, если ваш компьютер имеет многоядерный процессор!

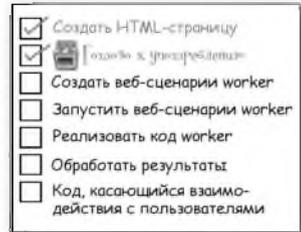
Результатом всех этих вычислений является значение, добавляемое в массив именованных значений, который помещается назад в объект `task`, чтобы `worker` смог отослать результат обратно основному коду.

Чуть позже мы пристальнее взглянем на эту часть.

Данный код будет размещаться в файле `workerlib.js`.

Создание веб-сценариев worker и раздача им задач...

Разобравшись с готовым к употреблению кодом, переключим внимание на написание кода, который будет генерировать и раздавать задачи веб-сценариям worker.



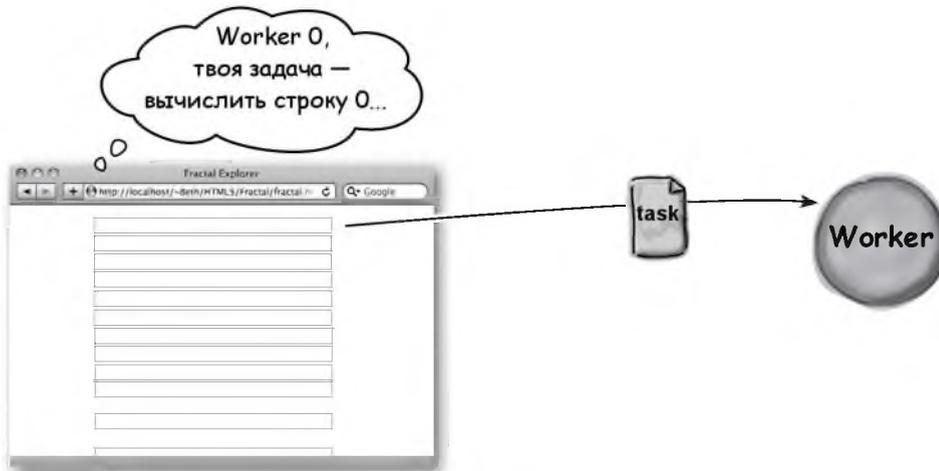
- 1 Мы создадим массив веб-сценариев worker, которые первоначально ничем не будут заняты. А также изображение, для которого ничего не будет вычисляться ($nextRow = 0$).



$nextRow = 0$



- 2 Мы будем совершать итерацию по массиву и генерировать объект task для каждого свободного worker.



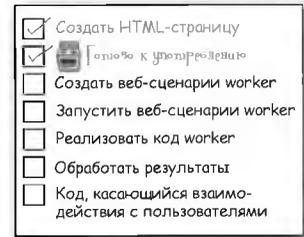
- 2 Продолжим совершать итерацию в поисках следующего незанятого worker, чтобы поручить ему задачу. Следующим по очереди у нас будет идти $nextRow = 1$. И так далее...

$nextRow = 1$



Написание кода

Теперь, когда мы знаем, как будем создавать паши веб-сценарии worker и управлять ими, напишем соответствующий код. Для этого нам потребуются пачальная функция, поэтому давайте создадим в `mandel.js` функцию `init`, — мы также предусмотрим там многое другое для обеспечения функционирования нашего приложения (например, позаботимся об инициализации графики).



Почему 8? Так случилось, что у нас имеется компьютер с 8 ядрами, поэтому такое количество хорошо сочетается с возможностями нашего компьютера. Но даже если вы не будете располагать таким количеством ядер, число 8 станет подходящим показателем, который следует испробовать в первую очередь.

```

var numberOfWorkers = 8;
var workers = [];

window.onload = init;

function init() {
  setupGraphics();

  for (var i = 0; i < numberOfWorkers; i++) {
    var worker = new Worker("worker.js");

    worker.onmessage = function(event) {
      processWork(event.target, event.data);
    }

    worker.idle = true;

    workers.push(worker);
  }

  startWorkers();
}

```

Сначала определим переменную, которая будет содержать нужное нам количество веб-сценариев worker. Мы используем число 8, и вы можете свободно поэкспериментировать с этим показателем, когда ваше приложение заработает.

Пустой массив для размещения наших worker

Зададим обработчик `onload`, который будет вызывать функцию `init`, когда страница полностью загрузится.

Данная функция определена в готовом коде и обрабатывает извлечение `context` элемента `canvas`, изменение размеров `canvas` в соответствии с размерами окна браузера и прочие графические детали.

Теперь мы совершаем итерацию по определенному количеству worker... ..и создаем новый worker из "worker.js", который мы еще не написали.

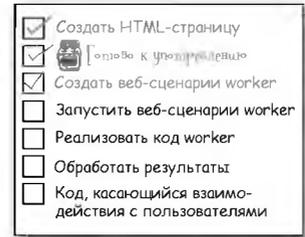
Затем задаем для обработчика сообщений каждого worker функцию, которая вызывает функцию `processWork`, и передаем ей `event.target` (worker, который только что закончил выполнение задачи) и `event.data` (результаты от данного worker).

Помните, что нам будет нужно узнать, какие веб-сценарии worker заняты делом, а какие бездельничают. Для этого нам потребуется добавить в worker свойство "idle". Это наше собственное свойство, которое не является частью API-интерфейса Web Workers. На данный момент мы присваиваем ему значение `true`, поскольку мы еще не поручили веб-сценариям worker никакой работы.

Добавляем только что созданный worker в массив `workers`.

И наконец, в какой-то момент нам потребуется запустить эти веб-сценарии worker. Мы поместим соответствующий код в функцию `startWorkers`, которую нам еще нужно написать.

Данный код будет размещаться в файле `mandel.js`.



Запуск веб-сценариев worker

Итак, нам необходимо разобраться с еще несколькими делами: запустить веб-сценарии worker, написать функцию, которая сможет обработать результаты, поступающие обратно от веб-сценариев worker, а также написать код для worker. Начнем с написания кода для запуска веб-сценариев worker:

Добавляем еще две глобальные переменные в `mandel.js`.

```
var nextRow = 0;
var generation = 0;
```

Сначала идет `nextRow`, которая отслеживает, на какой строке мы находимся, по мере того как мы продвигаемся по вычисляемому изображению.

Каждый раз, когда пользователь увеличивает количество Мандельброта, мы запускаем вычисление нового изображения. Переменная `generation` отслеживает, сколько раз мы это сделали. Более подробно об этом поговорим позже.

```
function startWorkers() {
  generation++;
  nextRow = 0;
```

Функция `startWorkers` будет запускать веб-сценарии worker, а также перезапускать их, если пользователь увеличит изображение. Таким образом, при каждом запуске веб-сценариев worker мы будем сбрасывать значение `nextRow` до нуля и увеличивать значение `generation`.

```
  for (var i = 0; i < workers.length; i++) {
    var worker = workers[i];
```

Как используются обе эти переменные, вам станет яснее чуть позже...

```
    if (worker.idle) {
```

Теперь мы совершаем цикл по всем веб-сценариям worker в массиве `workers`...

```
      var task = createTask(nextRow);
```

...и проверяем, не сидит ли без дела определенный worker.

```
      worker.idle = false;
      worker.postMessage(task);
```

Если выясняется, что worker ничем не занят, мы генерируем объект `task`, чтобы поручить ему задачу. Данная задача будет заключаться в вычислении строки множества Мандельброта. Функция `createTask` определена в `mandellib.js` и возвращает объект `task` со всеми данными, которые необходимы worker для вычисления соответствующей строки.

```
      nextRow++;
    }
  }
}
```

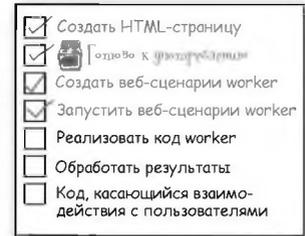
Теперь мы подошли к тому, чтобы дать worker кое-какую работу, поэтому задаем для свойства `idle` значение `false` (это подразумевает, что он будет занят делом).

А здесь мы даем указание worker начать работу, для чего отправляем сообщение, содержащее объект `task`. worker ведет прослушивание на предмет поступления сообщений, поэтому когда он получит данное сообщение, он начнет выполнение соответствующей задачи.

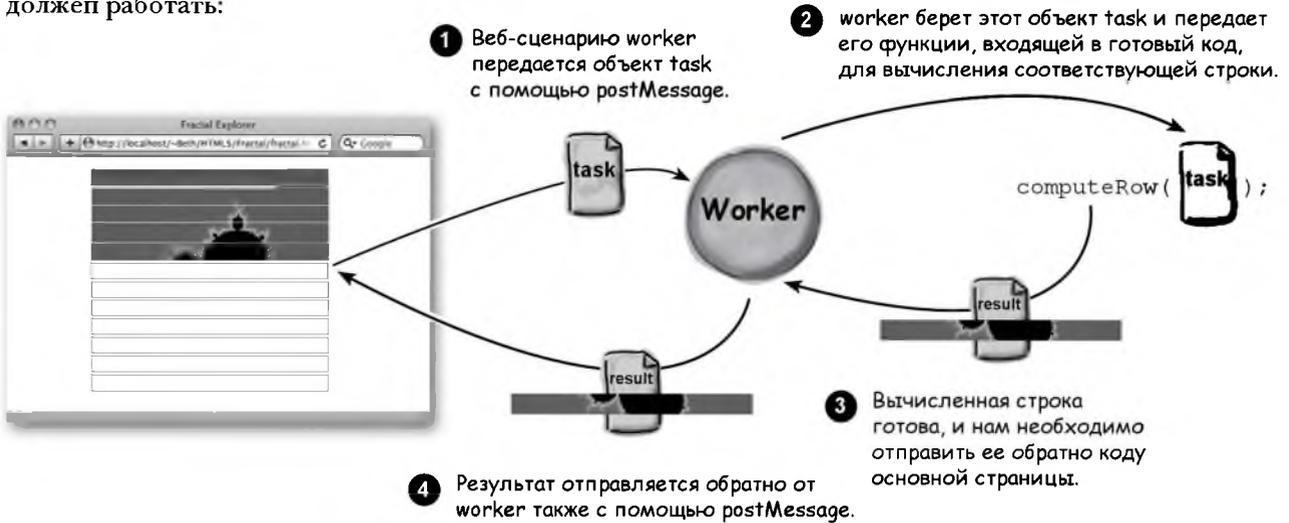
И наконец, мы увеличиваем значение `nextRow`, чтобы следующий worker приступил к вычислению следующей строки.

Данный код будет размещаться в файле `mandel.js`.

Реализация koga worker



Теперь, когда у нас есть код для запуска worker путем передачи каждому из них объекта task, займемся написанием кода worker. Затем нам останется лишь вернуться и обработать результаты от определенного worker, как только он закончит вычисление своей части фрактального изображения. Однако прежде, чем мы начнем писать код для worker, давайте взглянем на то, как он должен работать:



Итак, приступим к реализации. Введите и сохраните в файле worker.js код, который приведен далее.

Мы используем importScripts для импорта готового кода workerlib.js, чтобы worker смог вызвать функцию computeRow, определенную в данном библиотечном файле.

```
importScripts("workerlib.js");

onmessage = function (task) {

    var workerResult = computeRow(task.data);

    postMessage(workerResult);

}
```

Все, что делает worker, — это задает обработчик onmessage. Ему нет нужды делать что-либо еще, поскольку ему лишь требуется ждать сообщений от mandel.js для начала работы!

Он извлекает содержимое в виде данных из task и передает его функции computeRow, которая выполняет тяжелую работу по вычислению множества Мандельброта.

Результат вычислений, сохраненный в переменной workerResult, отправляется обратно основному JavaScript-коду с помощью postMessage.

Данный код будет размещаться в файле worker.js.

Короткий пит-стоп...



Мы с вами рассмотрели массу кода на предыдущих страницах. Давайте сделаем короткий пит-стоп, чтобы «дозаправить» наши баки и желудки.

Вам наверняка захочется заглянуть украдкой «за кулисы» и узреть, на что похожи `task` и `workerResult` веб-сценария `worker` (они очень похожи друг на друга, в чем вы убедитесь далее). Итак, берите бутылку любимой минералки и давайте взглянем на них, пока вы будете отдыхать...



`task` под увеличительным стеклом

Итак, ранее вы видели вызов `createTask` и `postMessage`, для которых используется `task`:

```
var task = createTask(nextRow);  
worker.postMessage(task);
```

При этом вам, возможно, интересно, на что похож `task`. Что ж, это объект, состоящий из свойств и значений:

`task` содержит все значения, необходимые веб-сценарию `worker` для выполнения его вычислений.

```
task = {  
  row: 1,  
  width: 1024,  
  generation: 1,  
  r_min: 2.074,  
  r_max: -3.074,  
  i: -0.252336,  
  max_iter: 1024,  
  escape: 1025  
};
```

← Определяет строку, для которой мы генерируем значения пикселей.

← Определяет ширину строки.

← Определяет, сколько раз мы прибежали к увеличению...

← Определяют вычисляемую нами область множества Мандельброта.

← Контролируют точность вычислений.



workerResult под увеличительным стеклом

А как насчет результатов (`workerResult`), которые мы получаем, когда `worker` заканчивает вычисление строки?

```
var workerResult = computeRow(task.data);
postMessage(workerResult);
```

На что похож объект `workerResult`? Он очень схож с `task`:

worker принимает переданный ему объект `task`, а затем добавляет в него свойство `values`, содержащее данные, необходимые для рисования строки в `canvas`.

```
workerResult = {
  row: 1,
  width: 1024,
  generation: 1,
  r_min: 2.074,
  r_max: -3.074,
  i: -0.252336,
  max_iter: 1024,
  escape: 1025,
  values: [3, 9, 56, ... -1, 22]
};
```

Здесь все то же самое, что и в `task`. И это здорово, поскольку когда мы получим `workerResult` от `worker`, мы будем знать все о `task`.

А это что-то новенькое. Здесь представлены значения каждого из пикселей, которые нам все еще нужно преобразовать в цвета (что делается в `drawRow`).



Пора снова в путь...

Благодарим, что вы уделите некоторое время и взглянули вместе с нами на `task` и `workerResult`. Сделайте последний большой глоток своей мипералки — мы снова отправляемся в путь!



Возвращаемся к коду: как осуществляется обработка результатов работы worker

Вы уже видели, как веб-сценарии worker генерируют результаты. Теперь взглянем, что произойдет, когда мы получим их от worker. Как вы помните, когда мы создавали наши веб-сценарии worker, мы задали обработчик сообщений с именем processWork:

```
var worker = new Worker("worker.js");

worker.onmessage = function(event) {
    processWork(event.target, event.data);
}
```

Наш обработчик сообщений вызывает функцию processWork, передавая ей event.data от worker, а также event.target, представляющий собой ссылку на worker, который прислал соответствующие данные.

Когда worker отправит нам обратно сообщение со своими результатами, их обработкой займется функция processWork. Как вы можете видеть, ей передаются два параметра: target сообщения, представляющий собой ссылку на worker, который его отправил, и data сообщения (это объект task со значениями для строки изображения). Таким образом, теперь наша работа будет заключаться в том, чтобы написать processWork (введите приведенный далее код в mandel.js):

```
function processWork(worker, workerResults) {
    drawRow(workerResults);
    reassignWorker(worker);
}
```

Передаем результаты drawRow для рисования пикселей в canvas.

Наш worker оказывается полностью свободным, поэтому мы можем поручить ему уже новую задачу. Для этого напомним функцию reassignWorker.

Мы почти достигли цели, так что давайте быстро напишем функцию reassignWorker, раз уж мы о ней заговорили. Вот как она работает: мы проверяем строку, которую вычисляем, используя глобальную переменную nextRow, и если остаются еще строки для вычисления (что можно выяснить, взглянув на количество строк в нашем canvas), мы поручаем worker новую задачу. В противном случае, если больше не останется работы, которую требуется сделать, мы просто присвоим свойству worker с именем idle значение true. Введите приведенный далее код тоже в mandel.js:

```
function reassignWorker(worker) {
    var row = nextRow++;

    if (row >= canvas.height) {
        worker.idle = true;
    } else {
        var task = createTask(row);
        worker.idle = false;
        worker.postMessage(task);
    }
}
```

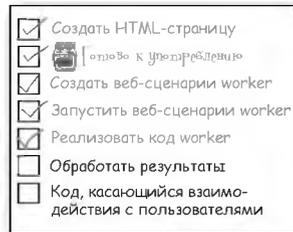
Мы собираемся поручить этому worker следующую строку, которую необходимо вычислить, поэтому извлекаем номер данной строки из nextRow и увеличиваем значение очередной строку). следующий worker получил для вычисления очередную строку).

Если строка превысит высоту canvas либо сравняется с ней по положению, дело сделано! Мы заполнили весь canvas результатами от веб-сценариев, использованных для вычисления множества Мандельброта.

canvas — это глобальная переменная, которая была задана, когда мы вызвали setupGraphics в нашей функции init.

Однако если у нас все же останутся строки для вычисления, то мы создадим новый объект task, касающийся следующей строки, которую нужно вычислить, убедимся, что свойство idle нашего worker имеет значение false, и отправим сообщение с новым объектом task нашему worker.

Данный код будет размещаться в файле mandel.js.



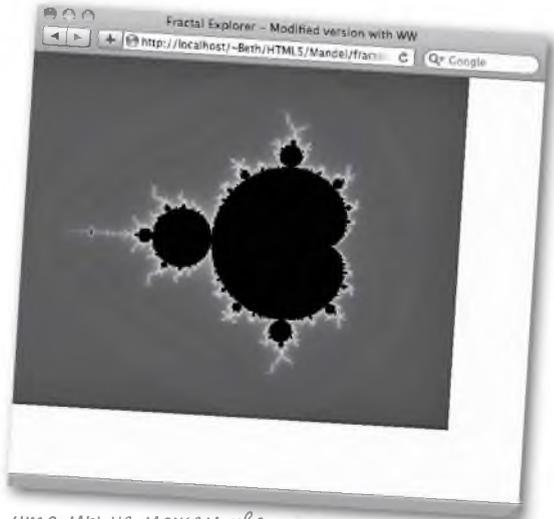
Психоделический тест-драйв



Ну хватит уже кода! Давайте проведем тест-драйв нашего приложения. Загрузите файл `fractal.html` в браузере и посмотрите, как веб-сценарии `worker` займутся работой. В зависимости от «пачипки» вашего компьютера скорость приложения Fractal Explorer должна стать пемпого выше, чем была ранее.

Мы, собственно говоря, еще не написали никакого кода для обработки изменения размеров окна браузера или щелчков с целью увеличения фрактального изображения. Поэтому все, что вы сможете увидеть на данный момент, — это изображение, которое приведено справа.

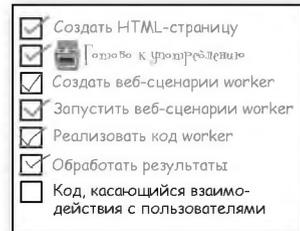
Тем не менее пока все хорошо, да?



Такие дела! Очень жаль, что мы не можем увеличить изображение и что оно пока еще не заполняет все окно, но мы дойдем до этого...

Обработка события `click`

Наши веб-сценарии `worker` трудятся над вычислением множества Мандельброта и возвращают результаты, чтобы мы могли нарисовать их в `canvas`. Но что произойдет, если вы щелкнете кнопкой мыши, чтобы увеличить изображение? К счастью, поскольку мы используем веб-сценарии `worker` для осуществления интесивных вычислений в фоновом режиме, интерфейс пользователя должен живо среагировать соответствующим образом на ваш щелчок. Тем не менее нам потребуется написать пемпого кода для фактической обработки события `click`.



- 1 Первое действие, которое нам потребуется предпринять, — это добавить обработчик, чтобы позаботиться о событиях, инициируемых щелчками мыши (помните, что щелчки осуществляются в нашем элементе `canvas`). Для этого мы просто добавим обработчик для свойства `canvas` с именем `onclick`:

```
canvas.onclick = function(event) {
    handleClick(event.clientX, event.clientY);
};
```

Если пользователь щелкнет на `canvas`, мы вызовем функцию `handleClick` с использованием координат `x` и `y` места, где был сделан щелчок.

Добавьте данный код ниже вызова `setUpGraphics` в функции `init` в `mandel.js`.

- 2 Остается лишь написать функцию `handleClick`. Прежде чем мы это сделаем, на секунду задумаемся вот над чем: когда пользователь щелкает на `canvas`, это означает, что он хочет увеличить соответствующую область (вы можете вернуться к однопоточной версии по адресу <http://wickedlysmart.com/hfhtml5/chapter10/singlethread/fractal.html>, чтобы увидеть данное поведение). Таким образом, когда пользователь щелкнет на `canvas`, нам нужно будет получить координаты области, которую он хочет увеличить, а затем привлечь все веб-сценарии `worker` к работе по генерированию нового изображения. Также не забывайте, что у нас уже имеется функция `startWorkers` для поручения новой работы любому незанятому `worker`. Давайте испытаем ее...

Вызов `handleClick` происходит, когда пользователь щелкает на `canvas`, чтобы увеличить фрактальное изображение.

Мы передаем координаты `x`, `y` места, где был сделан щелчок, то есть мы будем знать, в какой именно части экрана щелкнул пользователь.

```
function handleClick(x, y) {  
    var width = r_max - r_min;  
    var height = i_min - i_max;  
    var click_r = r_min + width * x / canvas.width;  
    var click_i = i_max + height * y / canvas.height;  
  
    var zoom = 8;  
  
    r_min = click_r - width/zoom;  
    r_max = click_r + width/zoom;  
    i_max = click_i - height/zoom;  
    i_min = click_i + height/zoom;  
  
    startWorkers();  
}
```

Данный код изменяет размеры области фрактального изображения, которую мы вычисляем, используя координаты `x`, `y` в центре новой области. Он также следит за тем, чтобы у новой области было такое же соотношение ширины и высоты, как у существующей области.

Задаем значения для глобальных переменных, используемых для создания объектов `task` для веб-сценариев `worker`: уровень увеличения определяет, насколько сильно мы увеличили фрактальное изображение, что, в свою очередь, определяет, какие значения множества Мандельброта будут вычисляться.

← Теперь мы готовы к перезапуску веб-сценариев `worker`.

Данный код будет размещаться в файле `mandel.js`.

Еще один тест-драйв



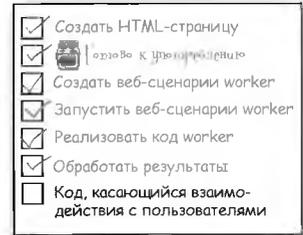
Проверим, какой эффект произведут внесенные нами в код изменения. Загрузите `fractal.html` в браузере и на этот раз щелкните где-нибудь в `canvas`. Сделав это, вы увидите, как веб-сценарии `worker` начнут работать над увеличенным представлением.

Теперь у вас должна появиться возможность приступить к исследованию! После того как вы немного поиграете, внесем ряд финальных изменений, чтобы довести нашу реализацию до конца.

Прекрасно! У нас появилась возможность увеличивать изображение, однако нам все еще нужно изменить размеры `canvas`, чтобы полностью подогнать его под размеры окна браузера.



Подгоняем canvas под размеры окна браузера



Как уже отмечалось ранее, нам необходимо, чтобы фрактальное изображение заполняло окно браузера, а это означает, что нам потребуется изменить размеры canvas, если изменятся размеры окна. Кроме того, если мы изменим размеры canvas, то нам также придется поручить веб-сценариям worker новый набор задач по перерисовке фрактального изображения, чтобы оно заполняло canvas с новыми размерами. Давайте напишем код для подгонки размеров canvas под размеры окна браузера, а также перезапустим веб-сценарии worker, раз уж мы об этом заговорили.

```
function resizeToWindow() {
    canvas.width = window.innerWidth;
    canvas.height = window.innerHeight;
    var width = ((i_max - i_min) * canvas.width / canvas.height);
    var r_mid = (r_max + r_min) / 2;
    r_min = r_mid - width/2;
    r_max = r_mid + width/2;
    rowData = ctx.createImageData(canvas.width, 1);

    startWorkers();
}
```

Функция `resizeToWindow` следит за тем, чтобы ширина и высота canvas задавалась в соответствии с новыми размерами окна.

Она также обновляет значения, которые worker будет использовать для осуществления своих вычислений, взяв за основу новую ширину и высоту (мы заботимся о том, чтобы фрактальное изображение всегда совпадало по размерам с canvas и сохраняло соотношение ширины и высоты окна).

И снова перезапускаем веб-сценарии worker.

Есть одна административная деталь, использующая глобальную переменную, о которой мы вам еще не говорили: `rowData`. `rowData` — это объект `ImageData`, который мы используем для рисования пикселей в строке canvas. Таким образом, при изменении размеров canvas нам потребуется вновь создать объект `rowData`, чтобы он имел значение ширины, совпадающее с новым значением ширины canvas. Взгляните на функцию `drawRow` в `mandellib.js`, чтобы увидеть, как `rowData` используется для рисования пикселей в canvas.

Теперь нам нужно установить `resizeToWindow` в качестве обработчика для события, иницируемого при изменении размеров окна браузера:

```
window.onresize = function() {
    resizeToWindow();
};
```

Данный код нужно поместить в функцию `init` в `mandel.js`, прямо под вызовом `setUpGraphics`.

Данный код будет размещаться в файле `mandel.js`.

Дотошный ~~шеф-повар~~ программист

Осталась еще одна вещь, без которой код попросту не будет корректным. Давайте вместе поразмыслим над следующей ситуацией: у вас имеется группа веб-сценариев `worker`, которые успешно трудятся над вычислением своих строк, и вдруг у пользователя возникает необходимость щелкнуть по изображению, чтобы увеличить его. Ничего хорошего в этом нет, поскольку веб-сценарии `worker` уже работают над вычислением своих строк, а теперь пользователю пощелкало изменить изображение целиком, что делает всю их работу бесполезной. Еще хуже то, что веб-сценарии `worker` не будут иметь понятия о том, что пользователь произвел щелчок, и в любом случае станут отправлять свои результаты обратно. А еще хуже то, что код в основной странице будет охотно принимать и отображать данные строки! Это не конец света, но мы столкнемся с точно такой же проблемой, если пользователь изменит размеры окна.

Заметка для редактора: извиняемся за небольшую напыщенность здесь, но после столь большого количества пройденных страниц эта новость может добить вас... ↗

На данный момент вы, вероятно, ни разу бы не обратили на это внимание, поскольку у вас имеется не так много веб-сценариев `worker`, при этом они очень быстро вычисляют одну и те же строки для нового изображения, перезаписывая предыдущие, некорректные строки. Тем не менее возникает ощущение, что здесь что-то не так. Кроме того, внести исправления настолько легко, что мы просто обязаны это сделать.

Следует признаться: мы знали, что так будет, и вы, возможно, помните маленькую переменную `generation`, с которой мы сталкивались ранее. При каждом перезапуске наших веб-сценариев `worker` мы увеличиваем значение `generation`. Также не забывайте об объекте `workerResults`, который поступает обратно от `worker`: у каждого результата имеется собственное поколение в качестве свойства. Таким образом, мы можем использовать переменную `generation` для того, чтобы узнать, получили мы результат, имеющий отношение к текущей или же к предыдущей визуализации.

Внесем необходимые исправления в код, а затем сможем поговорить о том, как они работают. Отредактируйте функцию `processWork` в `mandel.js` и добавьте туда эти две строки:

```
function processWork(worker, workerResults) {
  if (workerResults.generation == generation) {
    drawRow(workerResults);
  }
  reassignWorker(worker);
}
```

Мы проверяем результат от `worker`, чтобы выяснить, соответствует ли его поколение текущему.

Если соответствует, то мы рисуем строку, в противном случае строка, должно быть, является старой, и мы проигнорируем ее.

В любом случае мы поручаем `worker` уже новую работу!

Таким образом, мы удостоверяемся в том, что текущее поколение, над которым мы работаем, соответствует поколению результата, возвращаемого веб-сценарием `worker`. Если так оно и есть — отлично, тогда нам потребуется нарисовать строку. Если же соответствия нет, строка, скорее всего, является старой, поэтому мы просто будем ее игнорировать (очень жаль, что наш `worker` впустую потратил на нее свое время, однако мы не хотим рисовать старую строку из предыдущего изображения на экране).

Итак, вот и все, на этот раз по-настоящему. Пора убедиться в том, что вы внесли все приведенные выше изменения, и подготовиться к...

Время финального тест-драйва!



- Создать HTML-страницу
- Получить к узел/результату
- Создать веб-сценарии worker
- Запустить веб-сценарии worker
- Реализовать код worker
- Обработать результаты
- Код, касающийся взаимодействия с пользователями

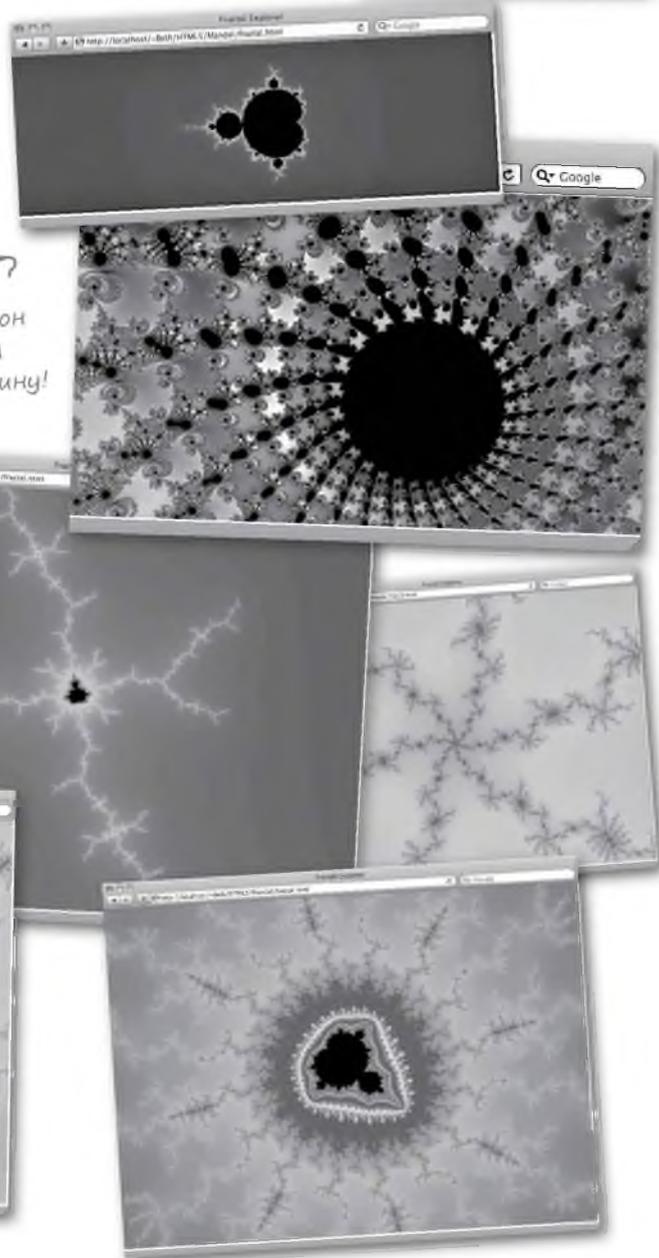
Вот и всё! Ваш код должен быть полностью готов к работе. Загрузите файл fractal.html в браузере и посмотрите, как веб-сценарии worker займутся делом. Данная версия должна работать быстрее и быть более отзывчивой, чем оригинальная однопоточная версия (если в вашем компьютере установлен процессор с более чем одним ядром, то она будет работать *намного* быстрее).

Поразвлекайтесь... попробуйте увеличить изображения... исследуйте их. Дайте нам знать, если найдете во множестве Мандельброта какую-нибудь ранее неизвестную «территорию» (отправьте свои скриншоты на #h5html5 через Твиттер, если пожелаете).

Щелкайте,
увеличивайте,
исследуйте!



Изменяйте размер окон
браузера, придавая им
любую форму и величину!



В ЛАБОРАТОРИИ

Если вы пишете высокопроизводительный код, то захотите проверить, как количество веб-сценариев `worker` повлияет на время выполнения работы вашим приложением.

Для этого вы можете воспользоваться монитором задач как в операционной системе OS X, так и в Windows. Если вернуться к нашей оригинальной однопоточной версии (по адресу <http://wickedlysmart.com/hfhtml5/chapter10/singlethread/fractal.html>), то картину использования ядер при выполнении приложения на нашем компьютере вы можете увидеть на диаграмме, приведенной справа.

Наш компьютер имеет восемь ядер, которые будут доступны приложению Fractal Explorer с веб-сценариями `worker`, и мы задаем количество `worker`, соответствующее этому числу ядер, указав `numberOfWorkers = 8`. Как вы можете видеть на мониторе активности, все 8 ядер используются по максимуму.

Как вы думаете, что будет, если мы зададим количество веб-сценариев `worker`, равное 2, 4, 16 или 32? Или равное какому-то промежуточному значению?

Попробуйте так сделать на своем компьютере и выясните, какие значения являются наиболее подходящими для вас.



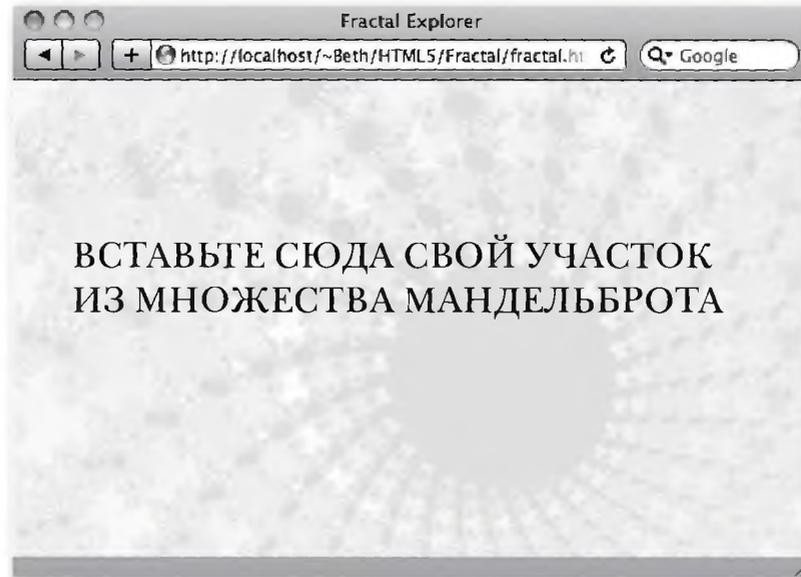
В нашем компьютере — восемь ядер. Одно из них используется по максимуму и не может вычислять быстрее, чем позволяет его рабочая частота. Остальные семь ядер ничего не делают, чтобы помочь ему.

Теперь все наши восемь ядер загружены работой, и вычисление фрактального изображения осуществляется НАМНОГО быстрее.



ЗАСТОЛЬТЕ СЕБЕ УЧАСТОК!

Вы сделали это! Вы создали полностью функциональное приложение Fractal Explorer, которое готово к исследованию областей множества Мандельброта. Так чего же вы ждете — приступайте и найдите свой кусочек виртуальной Вселенной. А когда отыщете, запечатлейте и вставьте его сюда, а затем присвойте своему новому участку название.



Придумайте название для своего участка: _____





Вам не кажется, что следует знать кое-что еще об API-интерфейсе Web Workers? Изучите следующие пару страниц, чтобы узнать обо всем, что мы не рассказали в этой главе.

Прерывание выполнения worker

Вы создали веб-сценарии worker для выполнения задачи, и после того как она была выполнена, захотели избавиться от всех этих worker (они отнимают ценную память браузера). Вы можете прервать выполнение worker из кода своей основной страницы следующим образом:

```
worker.terminate();
```

Данная строка приведет к отмене выполняющегося сценария worker, поэтому используйте ее осторожно. Прервав выполнение worker, вы не сможете повторно использовать его; вам придется создать новый worker.

worker также может сам остановить свое выполнение, вызвав `close()`; (изнутри worker).

Обработка ошибок worker

Что будет, если с worker случится какая-то ужасная ошибка? Как ее устранить? Используйте обработчик `onerror` для перехвата любых ошибок, а также для получения отладочной информации:

```
worker.onerror = function(error) {  
    document.getElementById("output").innerHTML =  
        "There was an error in " + error.filename +  
        " at line number " + error.lineno +  
        " : " + error.message;  
}
```

Использование `importScripts` для JSONP-запросов

Вы не сможете вставлять новые элементы `<script>` для выполнения JSONP-запросов из `worker`, однако у вас будет возможность использовать `importScripts` для совершения JSONP-запросов следующим образом:

```
function makeServerRequest() {
    importScripts("http://SomeServer.com?callback=handleRequest");
}

function handleRequest(response) {
    postMessage(response);
}

makeServerRequest();
```

Помните JSONP? Включите свою функцию обратного вызова в URL-запрос, и она будет вызываться с передачей JSON-результатов в параметре `response`.

Использование `importScripts` в `worker`

Вы могли упустить это из виду (мы быстро прошли мимо данного аспекта, взглянув на него только в одном примере), однако вы можете применять `setInterval` (и `setTimeout`) в веб-сценариях `worker` для повторного выполнения одних и тех же задач. Например, вы можете обновить `worker`, отправляющий цитаты (`quote.js`), таким образом, чтобы он отправлял произвольную цитату каждые 3 секунды:

```
var quotes = ["I hope life isn't a joke, because I don't get it.",
    "There is a light at the end of every tunnel...just pray it's not a train!",
    "Do you believe in love at first sight or should I walk by again?"];

function postAQuote() {
    var index = Math.floor(Math.random() * quotes.length);
    postMessage(quotes[index]);
}

postAQuote();
setInterval(postAQuote, 3000);
```

Помещаем эти две строки в функцию `postAQuote`...

...вызываем `postAQuote` для немедленной отправки цитаты, а затем задаем интервал для отправки дополнительных цитат, равный 3 секундам.

Подмастерье `subworker`

Если вашему `worker` потребуется помощь в выполнении его задачи, он сможет создать свои собственные веб-сценарии `worker`. Допустим, вы поручаете своему `worker` области изображения, которые он должен вычислить, а `worker` при этом может решить, что если область превышает некий размер, то работу по ее вычислению следует распределить между его собственными субсценариями `subworker`.

`worker` генерирует `subworker` тем же путем, посредством которого код в вашей странице создает `worker`:

```
var worker = new Worker("subworker.js");
```

Помните, что все `subworker`, как и обычные `worker`, являются довольно «тяжеловесными»: они занимают память и выполняются как отдельные потоки. Поэтому будьте осторожны с количеством создаваемых `subworker`.

КЛЮЧЕВЫЕ МОМЕНТЫ

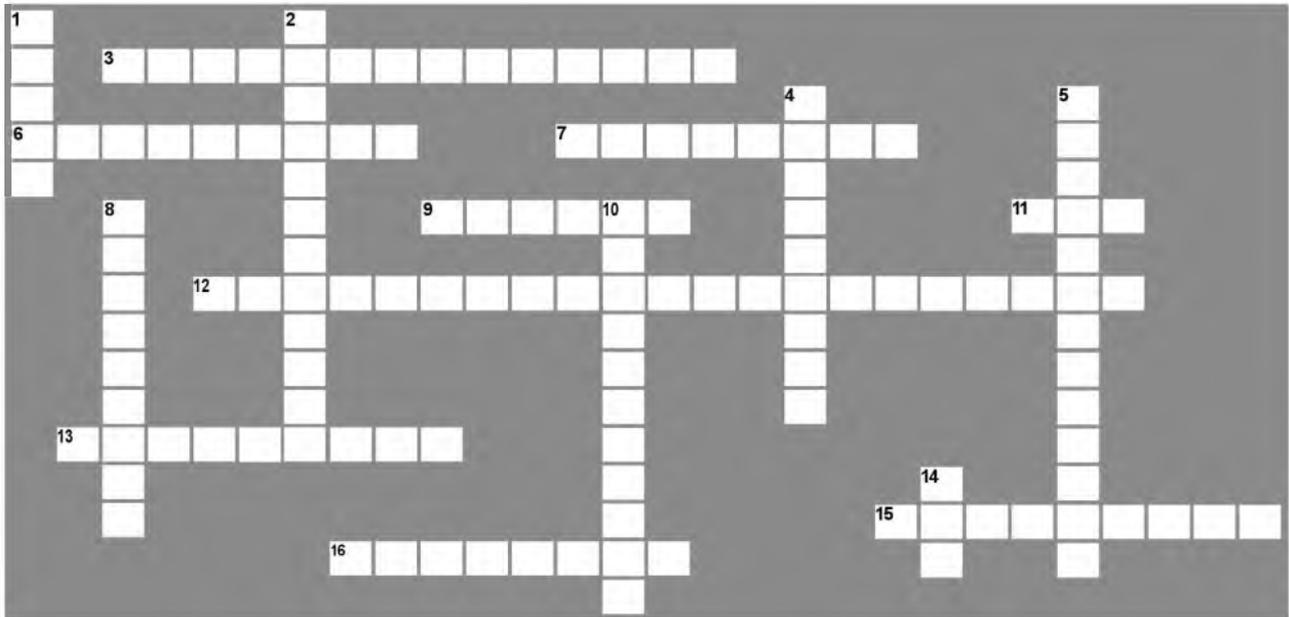


- Без веб-сценариев `worker` JavaScript является однопоточным, то есть может выполнять только какое-то одно действие за раз.
- Если вы взвалите на JavaScript-программу слишком много работы, то на экране может появиться диалоговое окно `Slow Script` (Медленный сценарий).
- Веб-сценарии `worker` обрабатывают задачи в отдельном потоке, благодаря чему ваш основной JavaScript-код сможет продолжать выполняться, а интерфейс пользователя будет оставаться отзывчивым.
- Код для веб-сценария `worker` располагается в отдельном файле и обособлен от вашего основного кода.
- У веб-сценариев `worker` нет доступа к любым функциям в коде вашей страницы, а также к объектной модели документа (DOM).
- Код в вашей странице и веб-сценарий `worker` общаются посредством сообщений.
- Для отправки сообщения веб-сценарию `worker` используйте `postMessage`.
- Вы можете отправлять строки и объекты веб-сценарию `worker` с помощью `postMessage`. Отправлять функции веб-сценарию `worker` нельзя.
- Для приема сообщений обратно от `worker` необходимо задать для свойства `worker` с именем `onmessage` функцию обработчика.
- Для получения сообщений веб-сценарием `worker` от кода вашей страницы необходимо задать для его свойства `onmessage` функцию обработчика.
- Когда `worker` готов к отправке результата обратно, он вызывает функцию `postMessage` и передает ей данный результат в качестве аргумента.
- Результаты работы `worker` инкапсулируются в объекте `event` и располагаются в свойстве `data`.
- Вы можете выяснить, какой именно `worker` отправил сообщение, воспользовавшись свойством `event.target`.
- Сообщения копируются, а не совместно используются кодом вашей основной страницы и `worker`.
- Вы можете использовать сразу несколько веб-сценариев `worker` для осуществления объемных вычислений, которые могут быть разбиты на ряд небольших задач (например, при проведении вычисления фрактальной визуализации или создании изображения методом трассировки лучей).
- Каждый `worker` выполняется в своем собственном потоке, поэтому ваш компьютер обладает многоядерным процессором, веб-сценарии `worker` смогут выполняться параллельно, что повышает скорость вычислений.
- Вы можете прервать выполнение `worker`, вызвав `worker.terminate()` из кода своей страницы. В результате выполнение сценария `worker` будет прекращено. `worker` также может сам остановить свое выполнение, вызвав `close()`.
- Все `worker` располагают свойством `onerror`. Вы можете задать для него значение в виде функции обработчика ошибок, которая будет вызываться в случае возникновения ошибки при выполнении сценария `worker`.
- Для включения и использования JavaScript-библиотек в файле вашего `worker` используйте `importScripts`.
- Вы также можете использовать `importScripts` в сочетании с JSONP. Реализуйте функцию обратного вызова, которая передается в URL-запросе, в файле `worker`.
- Несмотря на то что у веб-сценариев `worker` нет доступа к объектной модели документа (DOM) и к функциям в вашем основном коде, они могут использовать `XMLHttpRequest` и `localStorage`.



HTML5-Кроссворд

Здорово! Вы добрались до конца главы 10. Откиньтесь на спинку стула, расслабьтесь, а затем закрепите изученный материал, немного поработав другим полушарием своего мозга и разгадав кроссворд.



По горизонтали

3. Аппаратная особенность процессора, дающая ему возможность выполнять более одной задачи за раз.
6. Свойство, используемое для регистрации обработчика с целью приема сообщений.
7. В нашем первом примере мы использовали эту игру.
9. Вы можете передавать _____ веб-сценариям `worker` с помощью `postMessage`.
11. У веб-сценариев `worker` нет доступа к _____.
12. Наиболее широко известный фрактал.
13. _____/веб-сценарий `worker`.
15. Человек, который написал оригинальную версию `Fractal Explorer`.
16. Красивая область виртуальной сельской местности в множестве `Мандельброта` называется «_____ острова».

По вертикали

1. _____ выполнения.
2. Веб-сценарии `worker` могут использовать `XMLHttpRequest` и у них имеется доступ к _____.
4. Посредством них общаются `manager.js` и веб-сценарии `worker`.
5. Инструмент для импорта дополнительного кода в `worker`.
8. Что нужно ввести для того, чтобы прекратить выполнение `worker`?
10. В случае с множеством `Мандельброта` используются _____ числа.
14. Что нужно ввести для того, чтобы создать `worker`.

Важно, мы никогда вам этого не говорим, но это `Ажентс Хендлридж`.



СТАНЬ браузером. Решение

Пришло время притвориться браузером,
оценивающим JavaScript-код.

```

window.onload = function() {
    var worker = new Worker("worker.js");
    worker.onmessage = function(event) {
        alert("Worker says " + event.data);
    }
    for (var i = 0; i < 5; i++) {
        worker.postMessage("ping");
    }
}

```

Данный код отправит пять сообщений со строкой "ping" веб-сценарию worker, который в ответ отошлет пять сообщений со строкой "pong", поэтому на экране появится пять диалоговых окон alert с текстом "Worker says pong".

```

window.onload = function() {
    var worker = new Worker("worker.js");
    worker.onmessage = function(event) {
        alert("Worker says " + event.data);
    }
    for (var i = 5; i > 0; i--) {
        worker.postMessage("pong");
    }
}

```

Данный код отправит пять сообщений со строкой "pong" веб-сценарию worker, который проигнорирует их, поскольку они не включают строку "ping". Никакого вывода генерироваться не будет.

```

window.onload = function() {
    var worker = new Worker("worker.js");
    worker.onmessage = function(event) {
        alert("Worker says " + event.data);
        worker.postMessage("ping");
    }
    worker.postMessage("ping");
}

```

Данный код отправит сообщение со строкой "ping", а затем, когда каждый раз назад будет поступать сообщение со строкой "pong", он станет отправлять новое такое же сообщение, поэтому в результате мы будем наблюдать бесконечный цикл из появляющихся диалоговых окон alert с текстом "Worker says pong".

```

window.onload = function() {
    var worker = new Worker("worker.js");
    worker.onmessage = function(event) {
        alert("Worker says " + event.data);
    }

    setInterval(pinger, 1000);

    function pinger() {
        worker.postMessage("ping");
    }
}

```

Данный код будет отправлять сообщение со строкой "ping" каждую секунду, поэтому мы станем получать в ответ сообщение со строкой "pong" всякий раз, когда он будет отправлять "ping".

Возьми в руку карандаш



Решение

Несмотря на то что обычно веб-сценарии `worker` получают рабочие задания посредством сообщений, такой подход вовсе не обязателен. Взгляните на данный отличный и компактный способ сделать необходимую работу с помощью веб-сценариев `worker` и HTML. Когда вы разберетесь в том, что делает данный код, опишите это внизу:

```
<!doctype html>
<html lang="en">
  <head>
    <title>Quote</title>
    <meta charset="utf-8">
  </head>
<body>
  <p id="quote"></p>
  <script>
    var worker = new Worker("quote.js");
    worker.onmessage = function(event) {
      document.getElementById("quote").innerHTML = event.data;
    }
  </script>
</body>
</html>
```

quote.html



quote.js



```
var quotes = ["I hope life isn't a joke, because I don't get it.",
  "There is a light at the end of every tunnel...just pray it's not a train!",
  "Do you believe in love at first sight or should I walk by again?"];
var index = Math.floor(Math.random() * quotes.length);

postMessage(quotes[index]);
```

Свое описание приведите здесь:

В HTML у нас имеется скрипт, создающий `worker`, выполнение которого запускается незамедлительно. `worker` произвольно выбирает цитату из массива `quotes` и отправляет ее основному коду с помощью `postMessage`. Основной код извлекает цитату из `event.data` и добавляет ее на страницу в элемент `<p>` с `id` в виде "quote".



HTML5-кроссворд. Решение



Как было бы здорово, если бы это уже был конец книги. Если бы не было больше никаких ключевых моментов, головоломок, JavaScript-листингов или чего-то еще. Об этом можно лишь мечтать...



Поздравляем!
Вы дошли до конца.

Конечно же, еще есть приложение.

И выходные сведения.

А также адрес нашего сайта...

Так просто сбежать у вас не получится — мы серьезно!

Приложение. Оставшиеся темы

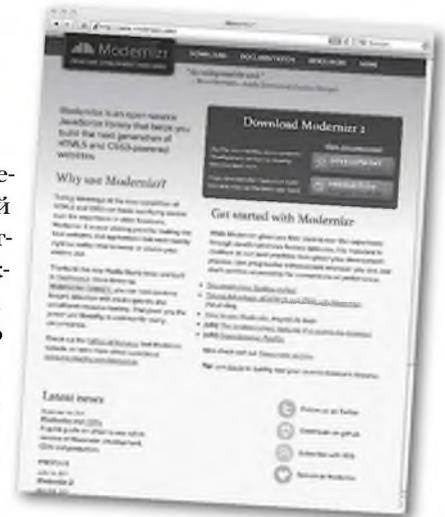
Десять важных тем (которые мы не рассмотрели)



Мы изучили множество различных тем, и вы почти закончили читать эту книгу. Нам грустно с вами расставаться, но прежде, чем мы это сделаем, поведаем еще кое о чем, чтобы подготовить вас к выходу в реальный мир. Вообще-то изначально мы включили в книгу все, что читателям следует знать о HTML5 (не рассмотренное в предыдущих главах), уменьшив размер шрифта до 0,00004. Но такой мелкий текст невозможно было прочитать. Поэтому нам пришлось выбросить большую часть и оставить в приложении только десять самых важных тем.

№ 1. Modernizr

Читая книгу, вы, вероятно, обратили внимание на то, что при необходимости выяснить, поддерживает ли браузер тот или иной API-интерфейс, оказывается, что единый способ сделать это отсутствует; фактически, поддержка почти любого API-интерфейса детектируется по-разному. Например, в случае API-интерфейса Geolocation мы ищем объект geolocation как свойство объекта navigator, в то время как в API-интерфейсе Web Storage мы проверяем, определен ли localStorage в объекте window. Что касается API-интерфейса Video, то в данной ситуации мы проверяем, есть ли у нас возможность создать элемент video в объектной модели документа (DOM) и т. д. Наверняка существует способ получше?



Modernizr – это JavaScript-библиотека с открытым исходным кодом, которая обеспечивает единый интерфейс для детектирования того, что именно поддерживает конкретный браузер. Modernizr охватывает всевозможные детали различных способов детектирования, включая даже факторинг в сложных ситуациях с устаревшими браузерами. Домашняя страница Modernizr располагается по адресу <http://www.modernizr.com/>. Библиотека Modernizr широко поддерживается разработчиками, и вы будете часто сталкиваться с ее применением в Интернете. Мы рекомендуем вам ее.

Включение Modernizr в свою страницу

Для использования Modernizr вам нужно загрузить эту JavaScript-библиотеку в свою страницу. Для этого сначала потребуется зайти на сайт Modernizr (<http://www.modernizr.com/download/>), где вы сможете произвести пользовательское конфигурирование библиотеки, которая в итоге будет содержать только необходимый вам код для детектирования (либо можете включить в нее сразу все, находясь на странице по данному адресу). Сделав это, сохраните библиотеку в файле и загрузите его в свою страницу (носите веб-сайт Modernizr, чтобы найти дополнительные учебные материалы и документацию по оптимальным методикам работы).

Как определяется, что именно поддерживает браузер

После инсталляции Modernizr процесс детектирования HTML5-элементов и API-интерфейсов JavaScript станет намного легче и проще:

Пример определения поддержки API-интерфейсов Geolocation, Web Storage и Video последовательным образом.

Примечание: возможности библиотеки Modernizr простираются значительно дальше простого детектирования поддерживаемых API-интерфейсов; она также позволяет определять поддерживаемые CSS-параметры, видеокодеки и многое другое. Поэтому обязательно ознакомьтесь с ней!



```
if (Modernizr.geolocation) {
    console.log("You have geo!");
}
```



```
if (Modernizr.localstorage) {
    console.log("You have web storage!");
}
if (Modernizr.video) {
    console.log("You have video!");
}
```

№ 2. Элемент audio и API-интерфейс Audio

HTML5 предусматривает стандартный способ воспроизведения аудио на ваших страницах с использованием не плагинов, а элемента `<audio>`:

```
<audio src="song.mp3" id="boombox" controls>
  Sorry but audio is not supported in your browser.
</audio>
```

Выглядит знакомо? Да, `audio` поддерживает аналогичную функциональность, как и `video` (естественно, за исключением возможности воспроизводить видео).

Помимо элемента `<audio>`, существует соответствующий API-интерфейс `Audio`, поддерживающий методы, которые вы и ожидали бы увидеть, например `play`, `pause` и `load`. Если вам все это покажется знакомым — это хорошо, поскольку API-интерфейс `Audio` является отражением (в соответствующих случаях) API-интерфейса `Video`. API-интерфейс `Audio` также поддерживает многие свойства, с которыми вы сталкивались в API-интерфейсе `Video`, например `src`, `currentTime` и `volume`. Ниже приведен небольшой связанный с аудио код, чтобы вы почувствовали, как данный API-интерфейс используется в сочетании с элементом в странице:

```
var audioElement =
  document.getElementById("boombox");
audioElement.volume = .5;
audioElement.play();
```

Извлекаем ссылку на элемент `audio`, после чего уменьшаем громкость звука на 1/2 и запускаем воспроизведение.

Как и в случае с видео, каждый браузер по-своему реализует внешний вид элементов управления проигрывателем (в число которых обычно входит ползунок воспроизведения, кнопки настановки на паузу и регулирования громкости).

Несмотря на простую функциональность, элемент `audio` и API-интерфейс `Audio` предоставляют вам широкий контроль. Подобно тому как мы поступали с элементом `video`, вы сможете создавать интересные веб-приложения, предусматривая скрытие элементов управления из виду и управление воспроизведением аудио в своем коде. А с помощью HTML5 теперь у вас есть возможность делать это без необходимости использовать (и изучать) плагины.

Стандарт в сфере аудиоформатов

Прискорбно, но, как и для видео, стандартного формата для аудио нет. Популярны три формата: MP3, WAV и Ogg Vorbis. Вы увидите, что поддержка данных форматов варьируется среди браузеров (на момент написания книги Chrome был единственным браузером, поддерживавшим все три формата).



№ 3. jQuery

Не забывайте, что Ajax — это всего лишь название шаблона использования XMLHttpRequest для извлечения данных, как отмечалось в главе 6.

jQuery представляет собой JavaScript-библиотеку, призванную уменьшить количество, а также упростить большую часть JavaScript-кода и синтаксиса, которые необходимы для работы с объектной моделью документа (DOM), использования Ajax и добавления визуальных эффектов к вашим страницам. jQuery является весьма популярной, широко используемой библиотекой и поддерживает возможность расширения посредством своей плагиновой модели.

jQuery не позволяет сделать ничего такого, чтобы было бы невозможно с помощью JavaScript (как мы уже говорили, jQuery является всего лишь JavaScript-библиотекой), однако она дает возможность сократить количество кода, которое вам придется написать.

Популярность jQuery говорит сама за себя, хотя может потребоваться некоторое время, чтобы вы привыкли к ней. Посмотрим, что можно сделать с помощью библиотеки jQuery. Исследуйте ее более пристально, если решите, что она будет вам полезна.

Для начала вопрос: помните ли вы все те функции `window.onload`, которые мы писали по ходу книги? Например:

```
window.onload = function() {
    alert("the page is loaded!");
}
```

А вот то же самое, но с использованием jQuery:

```
$(document).ready(function() {
    alert("the page is loaded!");
});
```

Как и в нашей версии: когда загрузка документа будет закончена, вызвать мою функцию.

Можно сократить данный код еще больше, до:

```
$(function() {
    alert("the page is loaded!");
});
```

Это классно, но чтобы привыкнуть, потребуется немного времени. Не беспокойтесь — все это быстро станет для вас привычным делом.

А что с извлечением элементов из объектной модели документа (DOM)? Именно здесь jQuery блистает. Допустим, в вашей странице имеется якорь с `id` в виде "buynow" и вы хотите назначить обработчик событий, инициируемых при щелчке кнопкой мыши, для события `click` данного элемента (как мы уже делали ранее несколько раз). Вот как это можно сделать:

```
$(function() {
    $("#buynow").click(function() {
        alert("I want to buy now!");
    });
});
```

Так что же здесь происходит? Сначала мы задаем функцию, которая будет вызываться по завершении загрузки страницы.

Далее мы извлекаем якорь с `id` в виде "buynow" (следует отметить, что jQuery использует CSS-синтаксис для выборки элементов).

Затем мы вызываем jQuery-метод `click` в отношении результата для задания обработчика `onclick`.

На самом деле это лишь начало; мы можем так же легко задать обработчик событий `click` для каждого якоря на странице:

```
$(function() {
  $("a").click(function() {
    alert("I want to buy now!");
  });
});
```

← Для этого нам потребуется использовать лишь соответствующее имя тега.

↪ Сравните показанное здесь с кодом, который вам пришлось бы написать, используя JavaScript без jQuery.

Мы также можем делать более сложные вещи:

```
$(function() {
  $("#playlist > li").addClass("favorite");
});
```

↪ Например, мы можем найти все элементы ``, которые являются дочерними по отношению к элементу с `id` в виде `"playlist"`.

↪ А затем добавить их в класс `"favorite"`.

↪ Вообще-то здесь jQuery лишь «разогревается»; данная библиотека позволяет делать вещи, намного сложнее этих.

У jQuery есть совершенно другая сторона, которая позволяет осуществлять любопытные интерфейсные трансформации элементов, как показано далее:

```
$(function() {
  $("#specialoffer").toggle(function() {
    $(this).animate({ backgroundColor: "yellow" }, 800);
  },function() {
    $(this).animate({ backgroundColor: "white" }, 300);
  });
});
```

↪ Этот код переключает элемент с `id` в виде `"specialoffer"` из состояния, в котором он желтый и имеет ширину 800 пикселей, в состояние, в котором он белый с шириной 300 пикселей, при этом переход между двумя данными состояниями анимируется.

Как вы можете видеть, с помощью jQuery возможно многое, при этом мы даже еще не начинали разговор об использовании данной библиотеки для общения с веб-службами и обо всех плагинах, которые работают с jQuery. Если вы заинтересовались данной темой, введите в браузере адрес <http://jquery.com/> и изучите имеющиеся там учебные материалы и документацию.

↪ А также загляните в книгу «Изучаем работу с jQuery» (Head First jQuery)!

№ 4. XHTML мертв, да здравствует XHTML

Мы довольно жестко новели себя с XHTML в этой книге, сначала во время дискуссии «XHTML мертв», а затем позднее, когда вели речь о «JSON против XML». Вся правда заключается в том, что когда дело доходит до XHTML, то здесь имеется в виду версия XHTML 2 и выше, которая умерла, при этом вы, фактически, можете писать свою HTML5-разметку, используя XHTML-стиль, если захотите. Почему вы можете этого захотеть? Что ж, вам может потребоваться осуществлять валидацию или трансформацию своих документов как XML либо обеспечить поддержку XML-технологий вроде SVG (см. тему № 5), которые работают с HTML.

Давайте взглянем на простой XHTML-документ, а затем пройдемся по основным аспектам (для нас не представлялось возможным поведать обо всем, что вам потребуется знать по данной теме, как и обо всех вещах, связанных с XML, поскольку вы быстро запутались бы).

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>You Rock!</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <p>I'm kinda liking this XHTML!</p>
    <svg xmlns="http://www.w3.org/2000/svg">
      <rect stroke="black" fill="blue" x="45px" y="45px"
        width="200px" height="100px" stroke-width="2" />
    </svg>
  </body>
</html>
```

То же самое определение doctype, что и раньше!

Это XML, нам нужно пространство имен!

Все элементы должны быть правильными; обратите здесь внимание на идущие в конце строки символы />, используемые для закрытия данного пустого элемента.

Мы используем SVG для рисования прямоугольника на нашей странице. Изучите тему №5 (на следующей странице), чтобы узнать больше о SVG.

Мы можем вложить XML прямо в страницу! И это очень здорово.

А теперь давайте взглянем на ряд вещей, которые вам потребуется принимать во внимание в случае с вашими XHTML-страницами:

- Ваша страница должна представлять собой правильный XML.
- Ваша страница должна загружаться с использованием типа MIME application/xhtml+xml, для чего вам потребуется удостовериться в том, что ваш сервер обеспечивает данный тип (проверив это самостоятельно либо связавшись с администратором вашего сервера).
- Обеспечьте и включите пространство имен XHTML в свой элемент <html> (что мы уже сделали в коде чуть выше).

Как мы уже отмечали ранее, относительно XML существует масса дополнительных вещей, о которых можно узнать, а также множество вещей, с которыми следует быть осторожными. И, как и всегда в случае с XML, да пребудет с вами Сила...

№ 5. SVG

Scalable Vector Graphics («Масштабируемая векторная графика»), или SVG, представляет собой еще один способ — помимо canvas — включения графики нативно в ваши веб-страницы. SVG существует уже долгое время (с 1999 года или около того) и в настоящий момент поддерживается во всех текущих версиях основных браузеров, включая Internet Explorer 9 и выше.

В отличие от canvas, который, как вы знаете, представляет собой элемент, позволяющий рисовать пиксели на поверхности для растрового рисования с помощью JavaScript, SVG-графика определяется использованием XML. XML, вы говорите? Да, XML! Вы будете создавать элементы, которые представляют графику, а затем объединять эти элементы друг с другом комплексными путями для создания графических сцен. Давайте взглянем на очень простой SVG-пример:

```

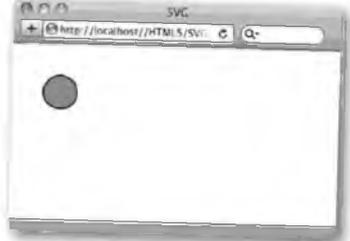
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>SVG</title>
  <meta charset="utf-8" />
</head>
<body>
  <div id="svg">
    <svg xmlns="http://www.w3.org/2000/svg">
      <circle id="circle"
        cx="50" cy="50" r="20"
        stroke="#373737" stroke-width="2"
        fill="#7d7d7d" />
    </svg>
  </div>
</body>
</html>

```

Мы используем HTML5 в XHTML-стиле потому, что применяем SVG, что основывается на XML.

Мы используем элемент `<svg>` прямо в нашей HTML-разметке!

Наш SVG-пример прост: он содержит лишь круг, который расположен по координатам $x=50$, $y=50$ и обладает радиусом 20 пикселей...
...снабжен линией обводки шириной 2 пиксела, имеющей темно-серый цвет...
...и заливкой с использованием серого цвета со средним значением.



Вы можете извлечь данный элемент `circle`, как и любой другой элемент из объектной модели документа DOM, и сделать с ним то, что вам необходимо... например, добавить обработчик событий `click` и изменять значение атрибута `fill` элемента `circle` на "red", когда пользователь щелкнет на данном элементе мышью.

SVG определяет ряд базовых фигур, таких как круги, прямоугольники, многоугольники, линии и т. д. Если вам потребуется нарисовать более сложные фигуры, то вы также сможете определить контуры с помощью SVG — естественно, в этот момент вещи начнут становиться более сложными (как вы уже могли убедиться в случае с контурами в canvas). Однако существуют графические редакторы, которые позволяют вам нарисовать сцену или экспортировать ее как SVG, избавляя вас от головной боли, которую вам доставила бы необходимость самостоятельно разбираться со всеми этими контурами! Вы сможете масштабировать свою графику, увеличивая или уменьшая ее по своему усмотрению, и она при этом не подвергнется нисселизации, которая имеет место при масштабировании изображений в формате jpeg или png. Это облегчает повторное использование графики в различных ситуациях. А поскольку SVG определяется с использованием текста, в случае с SVG-файлами можно осуществлять поиск, индексировать их, применять к ним сценарии, а также сжимать их. Исследуйте данную технологию подробнее, если она вас заинтересовала.

№ 6. Автономные веб-приложения

Если у вас имеется смартфон или планшетный компьютер, то вы, вероятно, выходите в Интернет, находясь в пути, и благодаря Wi-Fi и сетям сотовой связи почти все время подключены к Всемирной паутине. А как насчет того времени, когда вы не подключены к ней? Разве не будет здорово, если вы сможете продолжать использовать все эти веб-приложения HTML5, которые создаете для себя?

Что ж, теперь у вас есть такая возможность. Автономные веб-приложения поддерживаются всеми настольными и мобильными браузерами (с одним исключением: Internet Explorer).

Как же сделать свои веб-приложения доступными в автономном режиме? Нужно создать файл *манифеста кэша*, который будет содержать список всех файлов, необходимых вашему приложению для работы, и браузер загрузит все эти файлы и неключится на локальные файлы, когда ваше устройство перейдет в автономный режим. Чтобы сообщить своей веб-странице о том, что у нее имеется файл манифеста, нужно просто добавить его файловое имя в тег `<html>`, как показано далее:

```
<html manifest="notetosef.manifest">
```

Вот что содержит файл `notetosef.manifest`:

```
CACHE MANIFEST
CACHE:
notetosef.html
notetosef.css
notetosef.js
```

← С этого должен начинаться каждый файл манифеста кэша.

← В секции `CACHE` нужно указать все файлы, которые вы хотите кэшировать: файлы с HTML-разметкой, с CSS и JavaScript-кодом, файлы изображений и т. п.

Данный файл «говорит»: при заходе на веб-страницу, указывающую на данный файл, следует загрузить все файлы, указанные в секции `CACHE` этого файла. Вы также можете добавить две дополнительные секции в данный файл — `FALLBACK` и `NETWORK`. `FALLBACK` определяет то, какой файл будет использоваться, если вы попытаетесь получить доступ к файлу, который не был кэширован, а `NETWORK` определяет файлы, которые никогда не должны кэшироваться (например, в случае с ресурсами, отслеживающими уровень носимости).

А теперь, прежде чем решите поиграть со всем этим, вам необходимо узнать о двух вещах: во-первых, вам будет нужно убедиться в том, что ваш веб-сервер обеспечивает корректный тип MIME в случае с файлами манифеста кэша (точно так же, как нам это было нужно сделать, когда речь шла о видеофайлах в главе 8). Например, если вы используете сервер Apache, то добавьте приведенную далее строку в файл `.htaccess` на верхнем уровне вашего веб-каталога:

```
AddType text/cache-manifest .manifest
```

Второе, что вам будет нужно знать, заключается в том, что тестирование автономных веб-приложений осуществляется мудреным способом! Мы рекомендуем вам изучить соответствующие справочные материалы на эту тему, а также спецификацию автономных веб-приложений HTML5.

Как только у вас заработает базовое кэширование, вы сможете использовать JavaScript для получения уведомлений о связанных с кэшем событиях, которые инициируются, например, когда файл манифеста кэша подвергается обновлению, а также о состоянии кэша. Для получения уведомлений о событиях необходимо добавить обработчики событий в объект `window.applicationCache`, как показано далее:

```
window.applicationCache.addEventListener("error", errorHandler, false);
```

← Реализуйте `errorHandler` для получения уведомлений, если в случае с кэшем произойдет ошибка.



Благодаря такой вещи, как автономные веб-приложения, у вас есть возможность пользоваться своими любимыми веб-приложениями, когда вы не подключены к Интернету!

№ 7. API-интерфейс Web Sockets

В этой книге мы рассмотрели два способа коммуникации: XMLHttpRequest и JSONP. В обоих случаях мы использовали модель «запрос/ответ» на основе HTTP. То есть мы применяли браузер для совершения запроса исходной веб-страницы, CSS и JavaScript, и каждый раз, когда нам требовалось что-то другое, мы совершали новый запрос с использованием XMLHttpRequest или JSONP. Мы даже совершали запросы, когда для нас не было никаких новых данных, что иногда случалось в примере с Mighty Gumball.

Web Sockets — это новый API-интерфейс, который позволяет поддерживать открытое подключение к веб-службе, чтобы каждый раз, когда становятся доступными новые данные, эта служба могла присылать их вам (и ваш код мог получать соответствующие уведомления). Считайте все это открытой телефонной линией между вами и службой.

Вот высокоуровневый обзор того, как следует использовать данный API-интерфейс: сначала для создания веб-сокета вам потребуется применить конструктор WebSocket:

```
var socket = new WebSocket("ws://yourdomain/yourservice");
```

Обратите внимание: в случае с данным URL-адресом используется протокол WS, а не протокол HTTP.

И помните, что вам либо кому-то другому придется написать серверный код, чтобы у вас было с чем «общаться».

Вы сможете получить уведомление, как только сокет будет открыт посредством события open, для которого можно назначить обработчик:

```
socket.onopen = function(){
    alert("Your socket is now open with the web service");
}
```

Здесь мы предусмотрели обработчик, который станет вызываться, когда сокет будет полностью открыт и готов к коммуникации.

Вы сможете отправить веб-службе сообщение с помощью метода postMessage:

```
socket.postMessage("player moved right");
```

Здесь мы отправляем серверу строку; двоичные данные наступают, но пока еще не поддерживаются широко.

А для получения сообщений необходимо зарегистрировать другой обработчик, как показано далее:

```
socket.onmessage = function(event) {
    alert("From socket: " + event.data);
};
```

Зарегистрировав обработчик, мы можем получать все сообщения, содержащиеся в свойстве event.data.

Конечно, помимо всего этого есть немного еще кое-чего, и вам потребуется изучить учебные материалы, имеющиеся в Интернете, однако относительно API-интерфейса больше особо сказать нечего. Данный API-интерфейс отстает в развитии от некоторых других API-интерфейсов HTML5, поэтому читайте самые свежие руководства, где содержится информация о совместимости с браузерами, прежде чем браться за какой-либо крупный проект.



№ 8. Дополнительно об API-интерфейсе Canvas

Мы уже провели весело время с canvas в главе 7, создавая наш старпан TweetShirt. Однако есть еще много интересных вещей, связанных с canvas, которые можно сделать, и мы хотим рассказать здесь о некоторых из них.

Мы очень кратко упомянули, что вы сможете сохранять и восстанавливать context элемента canvas с помощью, соответственно, методов save и restore. Зачем вам может потребоваться сделать это? Допустим, вы задали значения для ряда свойств context, например fillStyle, strokeStyle, lineWidth и т. д. А затем у вас возникла необходимость временно изменить их значения для того, чтобы кое-что сделать, например нарисовать фигуру, однако вы при этом не хотите заниматься восстановлением всех этих значений, чтобы вернуться к тем значениям свойств, что были у вас вначале. Для этого вы можете воспользоваться методами save и restore:

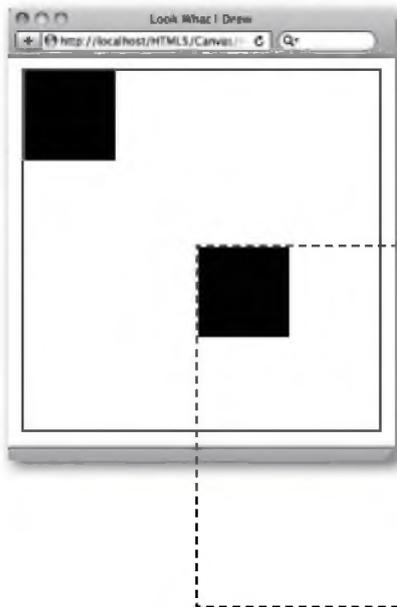
```
context.fillStyle = "lightblue";  
...  
context.save();  
context.fillStyle = "rgba(50, 50, 50, .5)";  
context.fillRect(0, 0, 100, 100);  
context.restore();  
...
```

Мы задаем значения для группы свойств в context и осуществляем рисование.

Теперь мы сохраняем context. Значения всех этих свойств будут надежно сохранены. Мы сможем изменить их...

...а затем вернуть всех их обратно, сделав их такими, какими они были прежде, когда мы сохраняли эти значения, для чего просто вызовем метод restore! В этот момент все наши свойства приобретут значения, которые они имели перед сохранением.

Данные методы особенно пригодятся кстати, когда вам потребуется транслировать или повернуть canvas для того, чтобы что-то нарисовать, а затем вернуть его в положение по умолчанию. Что делают методы translate и rotate? Давайте посмотрим...

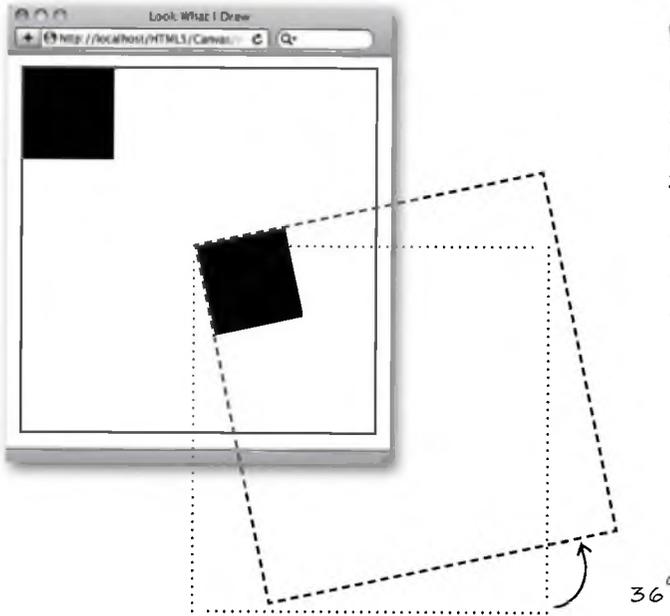


① У нас в странице имеется canvas размером 400x400 пикселей. Если мы нарисуем прямоугольник с координатами $x=0$, $y=0$, то он появится в верхнем левом углу, как мы и ожидали бы.

```
context.fillRect(0, 0, 100, 100);
```

② Теперь мы берем canvas и перемещаем его на 200 пикселей вправо и на 200 пикселей вниз. Если мы нарисуем еще один прямоугольник с координатами $x=0$, $y=0$, то он появится на 200 пикселей правее и на 200 пикселей ниже первого нашего прямоугольника. Мы только что транслировали canvas.

```
context.translate(200, 200);  
context.fillRect(0, 0, 100, 100);
```



③ А что, если мы повернем canvas до того, как нарисуем прямоугольник? canvas вращается вокруг своего верхнего левого угла (по умолчанию), и поскольку мы только что переместили верхний левый угол в позицию 200, 200, там и будет вращаться canvas.

```
context.translate(200, 200);
context.rotate(degreesToRadians(36));
context.fillRect(0, 0, 100, 100);
```

↑ Когда вы транслируете или поворачиваете canvas, он перемещается по координатной сетке, которая позиционируется относительно верхнего левого угла окна браузера. Если вы примените CSS для позиционирования canvas, то данные значения будут приниматься в расчет. Попробуйте!

А теперь давайте соединим все это! Вы можете использовать методы translate и rotate сообща для создания интересных эффектов.

```
var canvas = document.getElementById("canvas");
var context = canvas.getContext("2d");
var degrees = 36;
```

Здесь мы сохраняем context, чтобы его можно было легко восстановить, вернув в нормальное положение на координатной сетке после того, как мы закончим.

```
context.save();
context.translate(200, 200);
context.fillStyle = "rgba(50, 50, 50, .5)";
```

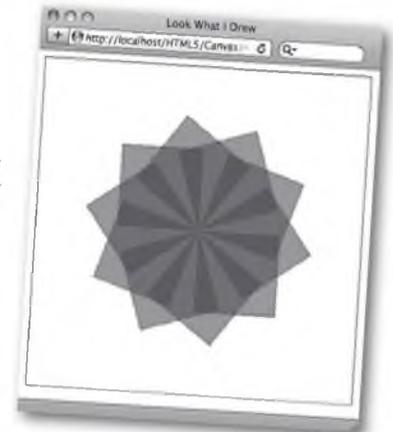
Мы транслируем наш canvas, задав при этом значения 200, 200.

А вот итоговый результат. Интересно выглядит!

```
for (var i = 0; i < 360/degrees; i++) {
    context.fillRect(0, 0, 100, 100);
    context.rotate(degreesToRadians(degrees));
}
context.restore();
```

Теперь наш canvas возвращается в свое исходное положение!

Мы рисуем 10 прямоугольников путем поворота canvas на 36° перед тем, как рисовать прямоугольник в позиции 0, 0, при каждом выполнении цикла.



Комбинируйте эти простые трансформации с другими даже еще более мощными (и сложными!) методиками вроде композиции и преобразований, и возможности по созданию графических изображений с использованием canvas станут безграничными.

№ 9. API-интерфейс Selectors

Вы уже знаете как производить выборку элементов из объектной модели документа DOM с помощью `document.getElementById`; мы использовали данный метод как способ сделать так, чтобы HTML и JavaScript работали сообща. Вы также увидели, как следует использовать `document.getElementsByTagName` (данный метод возвращает массив всех элементов, соответствующих заданному тегу) и даже метод `document.getElementsByClassName` (возвращающий, как вы уже догадались, все элементы в определенном классе). Благодаря HTML5 сейчас у нас имеется новый способ выборки элементов из объектной модели документа DOM, основанный на jQuery. Теперь вы можете задействовать те же селекторы, которые применяете в CSS с целью выборки элементов для стилизации в своем JavaScript, для выборки элементов из объектной модели документа DOM с использованием метода `document.querySelector`. Допустим, у нас имеется приведенная далее простая HTML-разметка:

```
<!doctype html>
<html lang="en">
<head>
  <title>Query selectors</title>
  <meta charset="utf-8">
</head>
<body>
  <div class="content">
    <p id="avatar" class="level5">Gorilla</p>
    <p id="color">Purple</p>
  </div>
</body>
</html>
```

Приспально взгляните на структуру данной HTML-разметки. Мы собираемся использовать API-интерфейс Selectors для выборки элементов из страницы.

У нас есть элемент `<div>` с классом "content" и два элемента `<p>`, каждый из которых обладает собственным идентификатором, при этом у одного из них имеется класс "level5".

Теперь давайте воспользуемся API-интерфейсом Selectors для выборки элемента `<p>` с `id` в виде "avatar":

```
document.querySelector("#avatar"); ←
```

По сути, это то же самое, что и `document.getElementById("avatar")`. Теперь давайте используем класс элемента для его выборки:

```
document.querySelector("p.level5"); ← Сейчас мы используем имя тега и класс для его выборки.
```

Мы также можем произвести выборку элемента `<p>`, который является дочерним по отношению к элементу `<div>`, следующим образом:

```
document.querySelector("div>p"); ← Здесь мы используем селектор дочерних элементов child для выборки элемента <p>, который является дочерним по отношению к элементу <div>. По умолчанию он выбирает первый элемент.
```

А если нам понадобятся все элементы `<p>` в `<div>`, то мы сможем воспользоваться другим методом в API-интерфейсе Selectors с именем `querySelectorAll`:

```
document.querySelectorAll("div>p"); ← Теперь мы извлекаем все дочерние элементы <p> элемента <div>!
```

`querySelectorAll` возвращает массив элементов точно так же, как и метод `getElementsByTagName`. Вот и всё! В API-интерфейсе Selectors имеется только два метода. Данный API-интерфейс является небольшим, однако он привносит новую мощную функциональность для выборки элементов.

№ 10. Однако есть даже еще кое-что!

Мы действительно хотели ограничиться десятью темами, которые не рассмотрели в данной книге, но, похоже, что нам есть еще куда двигаться, и вместо того чтобы стоять между вами и чтением алфавитного указателя, мы поведаем вам о нескольких дополнительных темах на одной странице. Вот они (имейте в виду, что некоторые из этих областей все еще развиваются, однако мы понимаем, что вам потребуются знания о них на будущее):

API-интерфейс Indexed Database и Web SQL

Если вы ищете нечто более индустриальное, чем API-интерфейс Web Storage, для сохранения своих данных локально, следите за сферой баз данных, функционирующих в Интернете. В настоящее время там присутствуют два конкурирующих решения: Web SQL и IndexedDB. По иронии судьбы, технология Web SQL обрела более широкую поддержку по сравнению с IndexedDB, однако недавно комитет стандартов выступил против нее (то есть он не рекомендует переписать ее как стандарт, и вам, вероятно, не следует основывать на пей свой следующий стартап). С другой стороны, технология IndexedDB пока не реализована широко, однако поддерживается со стороны Google и Firefox. IndexedDB обеспечивает быстрый доступ к обширной коллекции индексированных данных, в том время как Web SQL является компактным SQL-движком, работающим в браузере. Следите за тем, куда движутся эти технологии; они быстро меняются!



API-интерфейс Drag and Drop

Веб-разработчики уже долгое время обеспечивают возможность перетаскивать и помещать элементы с помощью мыши, используя для этого jQuery, а теперь данная функциональность пассивно поддерживается в HTML5. Используя API-интерфейс Drag and Drop, вы можете определить что-либо для перетаскивания, а также место, куда перетаскиваемый элемент можно будет поместить, и JavaScript-обработчики, которые будут уведомляться о различных событиях, инициируемых при перетаскивании и помещении элементов. Чтобы сделать элемент поддерживающим возможность перетаскивания с помощью мыши, нужно лишь задать для атрибута `draggable` значение `true`. Перетаскивать можно почти любые элементы: изображения, списки, параграфы и т. д. Вы можете конфигурировать поведение, связанное с перетаскиванием, путем прослушивания событий, например, `dragstart` и `dragend`, и даже изменять стиль элемента, чтобы во время перетаскивания он приобретал такой внешний вид, который вы пожелаете. Вы можете отправлять небольшое количество данных наряду с перетаскиваемым элементом, используя свойство `dataTransfer`; обращаться к нему можно посредством объекта `event`, чтобы узнать, например, перемещается ли определенный элемент или же копируется. Как вы можете видеть, благодаря такому API-интерфейсу HTML5, как Drag and Drop, перед вами открывается масса прекрасных возможностей по обеспечению новых взаимодействий с интерфейсом пользователя.

API-интерфейс Cross-document Messaging

В главе 6 мы использовали шаблон передачи данных, известный как JSONP, чтобы избежать междоменных коммуникационных проблем, возникающих в случае с `XMLHttpRequest`. Существует еще один способ коммуникации между документами — даже документами, находящимися в разных доменах. API-интерфейс Cross-document Messaging определяет, что вы сможете отправить сообщение документу, который загрузили, используя элемент `iframe`. Данный документ даже может располагаться в другом домене! Теперь вам не потребуется загружать *просто любой* документ в свой `iframe`; вам будет нужно удостовериться в том, что он исходит из домена, которому вы доверяете, и настроить его на прием ваших сообщений. В результате вы получаете способ отправлять сообщения туда-сюда между двумя HTML-документами.

И мы могли бы продолжать и дальше...

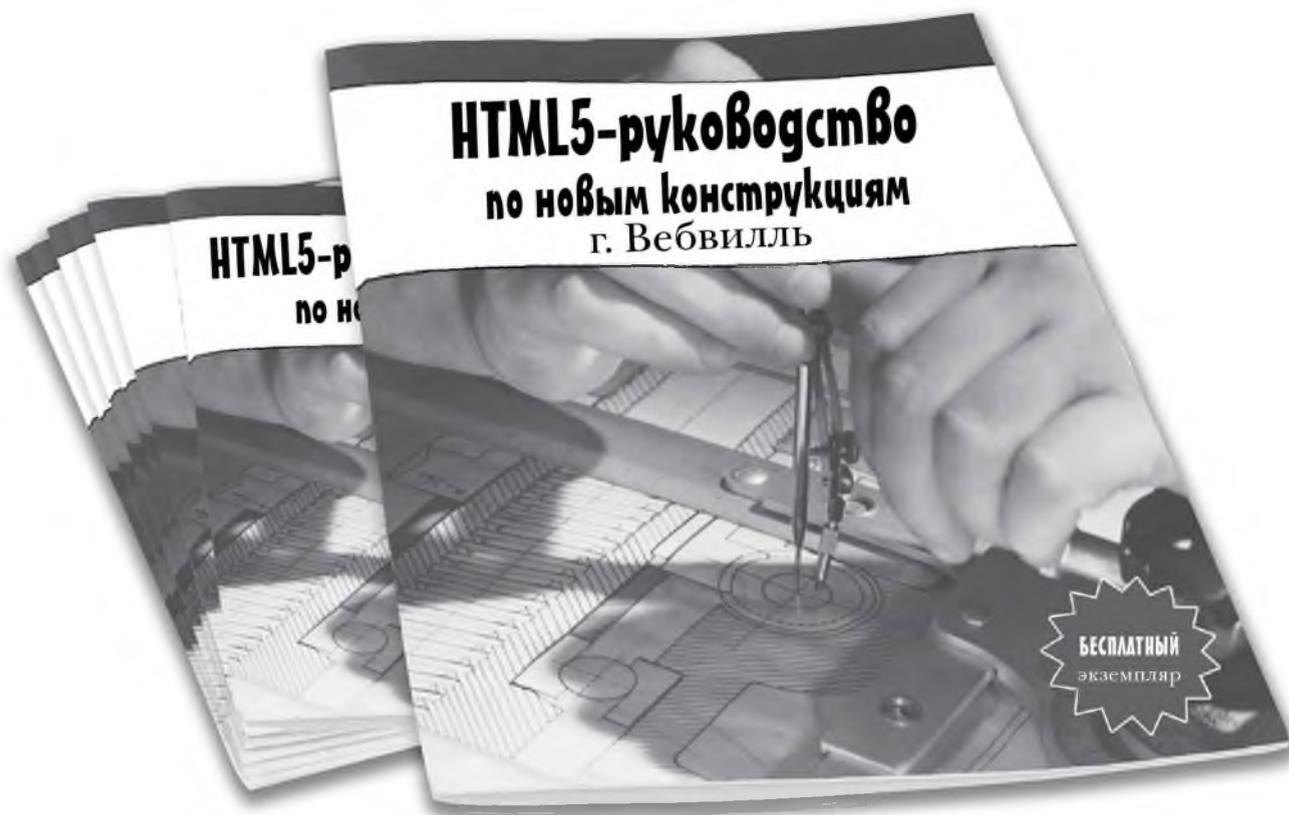
Захватывающая вещь относительно HTML5 заключается в том, что большое количество новых функциональных возможностей разрабатывается довольно быстрыми темпами; на этой странице мы могли бы привести еще больше материала, однако для этого просто не осталось места. Поэтому заходите к нам на сайт в Интернете по адресу <http://wickedlysmart.com>, чтобы узнавать обо всех новейших разработках в области HTML5!



Не верится, что книга почти закончилась. Прежде чем мы с вами расстанемся, у нас есть для вас прощальный подарок от города Вебвилль; это руководство по HTML5-элементам (а также по новинкам в CSS3), которое мы вам обещали. Замечательный город этот Вебвилль, не правда ли?!

HTML5-руководство по новым конструкциям

Здесь, в Вебвилле, мы недавно внесли ряд дополнений в наши «строительные» коды и подготовили удобное руководство по всем новым конструкциям, которые могут представлять для вас интерес. Так, в частности, мы добавили группу новых семантических элементов, которые обеспечат для вас даже еще более широкие возможности по созданию страниц. Однако наше руководство не является исчерпывающим; вернее, наша цель здесь заключалась в том, чтобы дать вам, опытного разработчику, достаточно материала для того, чтобы вы были хорошо знакомы с новыми HTML5-элементами и CSS3-свойствами и смогли использовать их в веб-приложениях, создание которых изучаете в данной книге, когда будете к этому готовы. Поэтому если вам требуется краткое учебное пособие по семантическим нововведениям в HTML5, то можете взять один экземпляр — они **БЕСПЛАТНЫ** (но лишь в течение ограниченного времени).



Вебвилльское руководство по семантическим элементам HTML5

Здесь, в Вебвилле, мы внесли ряд свежих изменений в наш «строительный» код и подготовили удобное руководство по всем новым конструкциям. Если вы используете элементы `<div>` вместо обычных конструкций вроде `<header>`, `<nav>`, `<footer>` и `<article>` для блоговых статей, то у нас есть для вас новые строительные блоки. Итак, давайте взглянем на них.



`<section>`

`<section>` — это «общий документ». Вы могли бы использовать `<section>` для разметки, например, «Руководства по HTML». Или чтобы заключить в него HTML. `<section>` не является общим контейнером — это работа `<div>`. И помните, что нужно использовать `<div>`, если вы просто группируете элементы в целях стилизации.



`<article>`

`<article>` — это отдельный блок содержимого, которым вы можете захотеть поделиться с другой страницей или веб-сайтом (или даже со своим псом). Идеально подходит для блог-постов и новых статей.



`<header>`

`<header>` предназначен для верхушек элементов вроде `<section>` и `<article>`. `<header>` также может использоваться в верхней части `<body>` для создания основного заголовка страницы.



`<footer>`

`<footer>` предназначен для нижней части элементов. Таких элементов, как `<section>`, `<article>` и `<div>`. Вы могли подумать, что допускается только один `<footer>` в странице; на самом деле вы можете использовать его всякий раз, когда вам будет необходимо разместить содержимое нижнего колонтитула в `<section>` вашей страницы (например, биографические данные или ссылки на некую статью).



`<hgroup>`

Это мудреный элемент. В отличие от `<header>`, который может содержать любые элементы, связанные с заголовком, `<hgroup>` специально предназначен для группирования заголовков (`<h1>...<h6>`) внутри `<header>`. Хорошо подходит для структур.

Вебвильское руководство по семантическим элементам HTML5



`<nav>`

`<nav>` — это навигация, и данный элемент, конечно же, предназначен для ссылок. Но не для любых ссылок: используйте `<nav>`, когда у вас имеется группа ссылок, например навигация для вашего сайта или блога. Не используйте его для одиночных ссылок в параграфах.



`<aside>`

`<aside>` удобен для размещения всевозможных вещей, которые представляют собой блоки содержимого, которые находятся вне основного потока вашей страницы, например сайдбары, выносные цитаты или мысли, пришедшие в голову позднее.



`<time>`

Ну наконец-то! Здесь речь идет о дате/времени. Вы можете использовать `<time>` для определения даты/времени. Не нужно спешить; найдите время и сделайте все правильно — изучите допустимые форматы для `<time>`.



`<progress>`

Почти закончили? Да, мы с успехом продвигаемся вперед по этим HTML5-элементам... `<progress>` представляет то, насколько далеко вы продвинулись в завершении выполнения задачи. Используйте немного CSS и JavaScript для создания красивых эффектов.



`<abbr>`

Эй, мистер, обязательно используйте аббревиатуру вместо этого длинного слова! Это значительно облегчит поиск, поскольку поисковые движки не всегда настолько сообразительны относительно аббревиатур, как мы.



`<mark>`

Используйте `<mark>` для пометки слов, например, с целью выделения или редактирования. Данный элемент хорошо подходит для использования в сочетании с результатами, выдаваемыми поисковыми движками.

Добавление стиля в ваши новые конструкции с помощью CSS3

Вебвилльское руководство по CSS3-свойствам

Теперь, когда ваши новые строительные блоки заняли свое место, пора задуматься о дизайне интерьера. Вам ведь захочется, чтобы все ваши новые конструкции выглядели красиво, не так ли?

Новые свойства

В CSS3 имеется порядочное количество новых свойств, многие из которых делают то, что создатели веб-страниц претворяли в жизнь довольно долго посредством различных витиеватых манипуляций с HTML, изображениями и JavaScript. Примеры:



```
opacity: 0.5;
```

```
border-radius: 6px;
```

```
box-shadow: 5px 5px 10px #373737;
```

← Делает элемент непрозрачным на 50%.

← Создает эффект закругления с изгибом величиной в 6 пикселей в случае с каждым углом.

← Тень длиной 5 пикселей, высотой 5 пикселей, размывкой с радиусом 10 пикселей, имеющая темно-серый цвет.

Новые макеты

Появилась пара новых мощных способов создания макетов страниц с помощью CSS, которые выходят за рамки обычного позиционирования и намного более просты в применении. Примеры:

```
display: table;
```

```
display: table-cell;
```

```
display: flexbox;
```

```
flex-order: 1;
```

} Это даст вам табличный макет без HTML-таблиц.

← Благодаря flexbox вы получаете больший контроль над тем, как браузер будет подходить к размещению блоков, например элементов div на странице.

Новые анимации

Благодаря анимациям у вас есть возможность анимировать переход между значениями свойств. Например, вы можете сделать так, чтобы что-нибудь исчезало из виду путем перехода из непрозрачного состояния в полупрозрачное:

```
transition: opacity 0.5s ease-in-out;
```

```
opacity: 0;
```

← Задавая для opacity значение 0, например, в случае наступления события, инициируемого при наведении курсора мыши на элемент, мы можем создать анимацию его исчезновения/появления снова.

← Свойство transition определяет свойство для перехода в конкретное состояние и выхода из него (в данном случае речь идет о непрозрачности), время, которое будет занимать переход, и функцию замедления, чтобы он был постепенным.

Новые селекторы

Появилась целая масса новых селекторов, включая nth-child, который позволяет нацеливаться на специфические дочерние элементы, заключенные в том или ином элементе. Наконец-то у вас появилась возможность задавать цвет фона чередующихся строк в списке, не прилагая при этом сумасшедших усилий.

```
ul li:nth-child(2n) { color: gray; }
```

← Это означает: выбрать каждый второй элемент списка и задать серый цвет фона.

✧ **Выходные сведения** ✧



Дизайн всех внутренних макетов был выполнен Эриком Фрименом и Элизабет Робсон. Кэти Сиерра и Берт Бэйтс придумали стиль оформления книг из серии «Изучаем...». При производстве данной книги были использованы программы Adobe InDesign CS и Adobe Photoshop CS.

К числу мест, где осуществлялось написание данной книги, относятся следующие: Бэйнбридж Айленд, штат Вашингтон; Портленд, штат Орегон; Лас-Вегас, штат Невада; Порт-оф-Несс, Шотландия; Сисайд, штат Флорида; Лексингтон, штат Кентукки; Туксон, штат Аризона и Анахейм, штат Калифорния. На протяжении долгих дней, пока мы писали эту книгу, нам придавал силы кофеин в чае «Honest Tea», «GT's Kombucha», а также музыка таких исполнителей, как Sia, Sigur Ros, Том Вейтс, OMD, Филип Глас, Muse, Ено, Кришна Дас, Майк Олдфилд, Одра Мэй, Devo, Стив Роач, Veuman Brothers, Pogo и всех людей на turntable.fm, а также музыка огромной массы исполнителей 1980-х годов, которые вас, возможно, не заинтересуют.

А вы знаете о нашем веб-сайте?
Там приводятся ответы на некоторые вопросы из этой книги, а также руководства, из которых можно научиться дополнительным вещам, и ежедневные обновления в блоге от авторов!

Мы не прощаемся с Вами!

Заходите на сайт
wickedlysmart.com



Э. Фримен, Э. Робсон

Изучаем программирование на HTML5

Серия «Head First O'Reilly»

Перевел с английского В. Черник

Заведующий редакцией
Руководитель проекта
Ведущий редактор
Литературный редактор
Художник
Верстка

*К. Галицкая
Д. Виноцкий
М. Моисеева
М. Моисеева
Л. Адуевская
Е. Лея*

ООО «Мир книг», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.

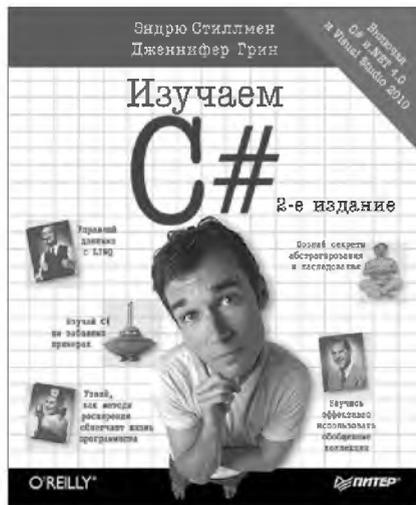
Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Подписано в печать 10.09.12. Формат 84х108/16. Усл. п. л. 67,200. Тираж 2000. Заказ 0000.

Отпечатано с готовых диапозитивов в типографии «Вятка». 610033, Киров, ул. Московская, 122.

Изучаем С#. 2-е изд. *Head First C#, 2nd Ed.*

Э. Стиллмен, Дж. Грин



ISBN:978-5-459-00422-9

Объем: 696 с.

Дата выхода: сентябрь 2011

В отличие от большинства книг по программированию, построенных на основе скучного изложения спецификаций и примеров, с этой книгой читатель сможет сразу приступить к написанию собственного кода на языке программирования С# с самого начала. Вы освоите минимальный набор инструментов, а далее примете участие в забавных и интересных программных проектах: от разработки карточной игры до создания серьезного бизнес-приложения. Второе издание книги включает последние версии С# .NET 4.0 и Visual Studio 2010 и будет интересно всем изучающим язык программирования С#. Особенностью данного издания является уникальный способ подачи материала, выделяющий серию «Head First» издательства O'Reilly в ряду множества скучных книг, посвященных программированию.

Изучаем HTML, XHTML и CSS

Head First HTML with CSS & XHTML

Э. Фримен, Э. Фримен



ISBN: 978-5-49807-113-8

Объем: 656 с.

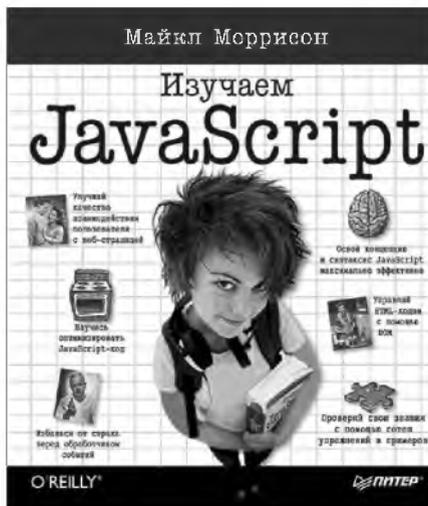
Дата выхода: февраль 2010

Устали от чтения таких книг по HTML, которые понятны только специалистам в этой области? Тогда самое время взять в руки наше издание. Хотите изучить HTML так, чтобы уметь создавать веб-страницы, о которых вы всегда мечтали? Так, чтобы более эффективно общаться с друзьями, семьей и привередливыми клиентами? Хотите действительно обслуживать и улучшать HTML-страницы по прошествии времени, чтобы они работали во всех браузерах и мобильных устройствах? Тогда эта книга для вас. Прочитав ее, вы узнаете все секреты создания веб-страниц. Благодаря ей вам больше не придется думать, какие цвета нужно использовать, чтобы они сочетались между собой, как правильно применять шрифты, чтобы они не «плавали» по экрану и верно отображались в различных браузерах. Вы узнаете, как работают профессионалы, чтобы получить визуально привлекательный дизайн, и как максимально эффективно использовать HTML, CSS и XHTML, чтобы создавать такие веб-страницы, мимо которых не пройдет ни один пользователь.

Изучаем JavaScript

Head First JavaScript

М. Моррисон



ISBN: 978-5-459-00322-2

Объем: 592 с.

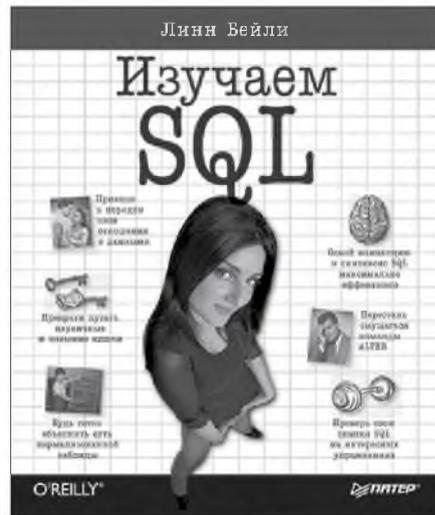
Дата выхода: октябрь 2011

Вы готовы сделать шаг вперед в своей практике веб-программирования и перейти от верстки в HTML и CSS к созданию полноценных динамических страниц? Тогда пришло время познакомиться с самым «горячим» языком программирования — JavaScript! С помощью этой книги вы узнаете все о языке JavaScript: от переменных до циклов. Вы поймете, почему разные браузеры по-разному реагируют на код и как написать универсальный код, поддерживаемый всеми браузерами. Вам станет ясно, почему с кодом JavaScript никогда не придется беспокоиться о перегруженности страниц и ошибках передачи данных. Не пугайтесь, даже если ранее вы не написали ни одной строчки кода, — благодаря уникальному формату подачи материала эта книга с легкостью проведет вас по всему пути обучения: от написания простейшего java-скрипта до создания сложных веб-проектов, которые будут работать во всех современных браузерах. Особенностью данного издания является уникальный способ подачи материала, выделяющий серию «Head First» издательства O'Reilly в ряду множества скучных книг, посвященных программированию.

Изучаем SQL

Head First SQL

Л. Бейхли



ISBN: 978-5-459-00421-2

Объем: 592 с.

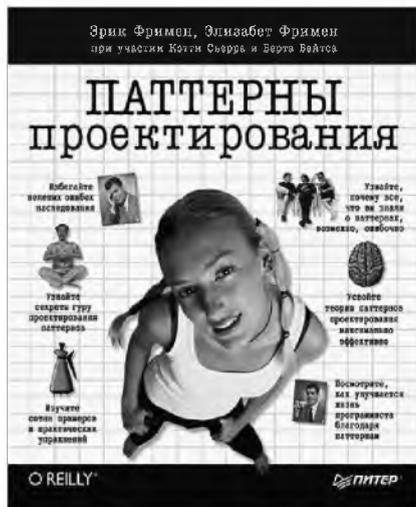
Дата выхода: сентябрь 2011

В современном мире наивысшую ценность имеет информация, но не менее важно уметь этой информацией управлять. Эта книга посвящена языку запросов SQL и управлению базами данных. Материал излагается, начиная с описания базовых запросов и заканчивая сложными манипуляциями с помощью объединений, подзапросов и транзакций. Если вы пытаетесь разобраться в организации и управлении базами данных — эта книга будет отличным практическим пособием и предоставит вам все необходимые инструменты. Особенностью данного издания является уникальный способ подачи материала, выделяющий серию «Head First» издательства O'Reilly в ряду множества скучных книг, посвященных программированию.

Паттерны проектирования

Head First Design Patterns

Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс



ISBN: 978-5-459-00435-9

Объем: 656 с.

Дата выхода: март 2011

В мире постоянно кто-то сталкивается с такими же проблемами программирования, которые возникают и у вас. Многие разработчики решают совершенно идентичные задачи и находят похожие решения. Если вы не хотите изобретать велосипед, используйте готовые шаблоны (паттерны) проектирования, работе с которыми посвящена эта книга. Паттерны появились, потому что многие разработчики искали пути повышения гибкости и степени повторного использования своих программ. Найденные решения воплощены в краткой и легко применимой на практике форме. Особенностью данного издания является уникальный способ подачи материала, выделяющий серию «Head First» издательства O'Reilly в ряду множества скучных книг, посвященных программированию. Книга будет интересна широкому кругу веб-разработчиков, от начинающих до профессионалов, желающих освоить работу с паттернами проектирования.

Программируем для iPhone и iPad

Head First iPhone and iPad Development, 2nd Ed.

Д. Пилон, Т. Пилон



ISBN: 978-5-459-00375-8

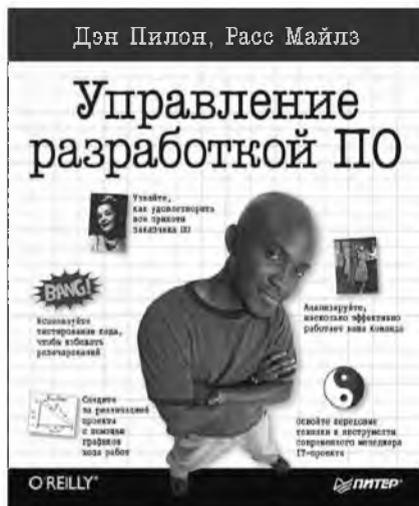
Объем: 592 с.

Книга предназначена для профессиональных программистов, желающих создавать приложения для iPhone и iPod с помощью Apple SDK и языка программирования Objective-C. В отличие от большинства книг по программированию, построенных на основе скучного изложения спецификаций и примеров, с этой книгой читатель сможет приступить к написанию собственного кода с самого начала. Вы освоите минимальный набор инструментов и примете участие в забавных и интересных программных проектах: от разработки игр до создания серьезных бизнес-приложений. Нестандартный способ подачи материала облегчает понимание и значительно увеличивает скорость восприятия, заставляя вас думать головой. Не пугайтесь, даже если ранее вы не написали ни одной строчки кода — благодаря своему уникальному визуальному формату подачи материала эта книга с легкостью проведет вас по всему пути создания качественного приложения для iPhone и iPad.

Управление разработкой ПО

Head First Software Development

Д. Пилон, Р. Майлз



ISBN: 978-5-459-00522-6

Объем: 464 с.

Дата выхода: март 2011

Даже опытные разработчики программного обеспечения постоянно сталкиваются с трудностями при реализации программных проектов: например, из-за смены требований заказчика ПО или непонимания конечными пользователями логики работы с новой программой. Если вы не собираетесь пасовать перед этими и другими распространенными проблемами управления IT-проектами, изучите с помощью этой уникальной книги передовые методы и практики, наработанные в области разработки программного обеспечения. Здесь вы получите всю необходимую информацию о каждом этапе жизненного цикла разработки программного обеспечения: переговоры с заказчиком и формализация клиентских требований, планирование процесса разработки ПО и подготовка технического задания, постановка задач и написание инструкций, тестирование ПО и устранение «багов». Особенностью этой книги является уникальный способ подачи материала, выделяющий серию «Head First» издательства O'Reilly в ряду множества скучных книг, посвященных программированию и разработке ПО.