ИЗУЧАЕМ

Ajax

Научите ваши веб-страницы говорить и слушать одновременно

Выйдите за границы обычного мира при помощи JSON, XML и DOM

Изучайте асинхронные и синхронные приложения за чашечкой кофе

Занимательный путеводитель по миру динамических веб-страниц

Загрузите свои мозги асинхронным программированием

Сделайте неповоротливые веб-приложения такими же легкими и быстрыми, как и десктопные программы

O REILLY

ПИТЕР

УДК 004.738.5
М15

Маклафлин Б.
М15 Изучаем Ajax. — СПб.: Питер, 2008. — 443 с.: ил.

ISBN 978-5-91180-322-3

Книга посвящена технологии веб-программирования Ajax, стоящей на ступень выше базовых DHTML и JavaScript. С помощью Ajax можно создавать интерактивные веб-приложения, отличающиеся быстродействием и высокой производительностью. Эта книга ответит на вопрос, как асинхронные запросы используются в технологии Ajax, и поможет читателю выйти на новый уровень в создании веб-приложений.

Книга будет интересна широкому кругу веб-разработчиков, от начинающих и до профессионалов, желающих перейти на Ajax.

ББК 32.988.02-018
УДК 004.738.5

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.

ISBN 0596102259 (англ.)
ISBN 978-5-91180-322-3

© O'Reilly
© Перевод на русский язык ООО «Питер Пресс», 2008
© Издание на русском языке, оформление ООО «Питер Пресс», 2008

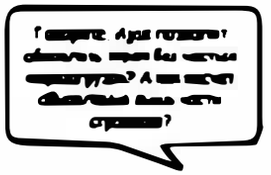
Содержание (сводка)

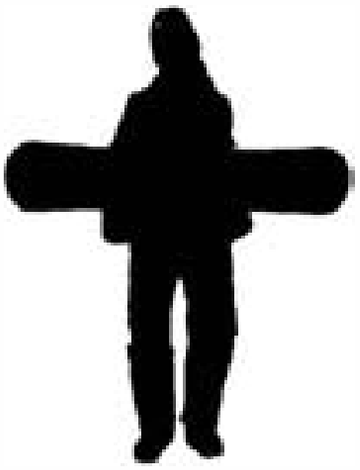
Содержание (настоящее)

1 Веб-приложения для нового поколения

Использование Ajax

Наводим глянец на веб-приложения.
Надоели неуклюжие веб-интерфейсы и долгие перезагрузки страниц? Пора придать веб-приложениям стиль, присущий классическим настольным приложениям. О чем идет речь? О последней новинке в области веб-разработки: Ajax (асинхронный JavaScript и XML) — ваш пропуск в мир полнофункциональных Интернет-приложений, более интерактивных, эластичных и удобных в использовании.



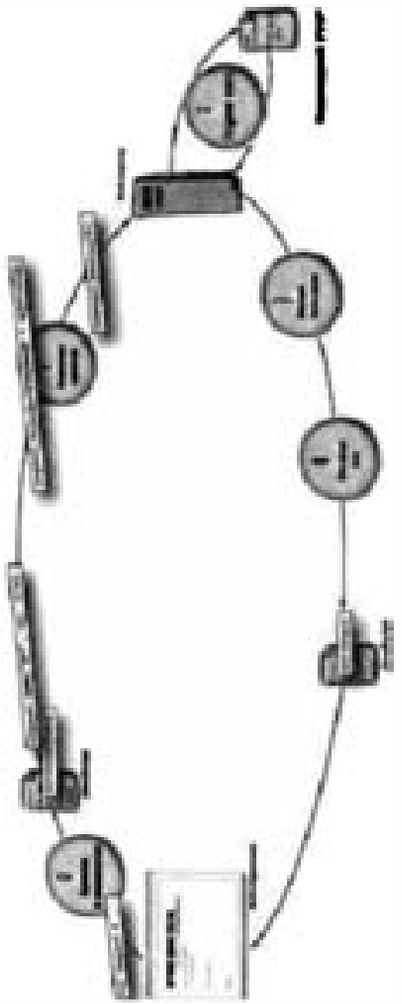


2 Создание запросов Ajax

Учимся говорить на языке Ajax

Учимся асинхронным взаимодействиям

Чтобы написать очередное веб-приложение, необходимо знать Ajax во всех подробностях. В этой главе представлено софтовое знание асинхронного кода JavaScript: вы научитесь отправлять запросы разным браузерам, управлять состояниями готовности и кодами состояния, и даже попутно освоите несколько полезных трюков из области динамического HTML-кода. К концу главы вы будете создавать запросы и обрабатывать ответы как настоящий профессионал... кстати, я упомянул, что вашим пользователям не придется ждать, пока вы будете учиться?



3 Асинхронные приложения

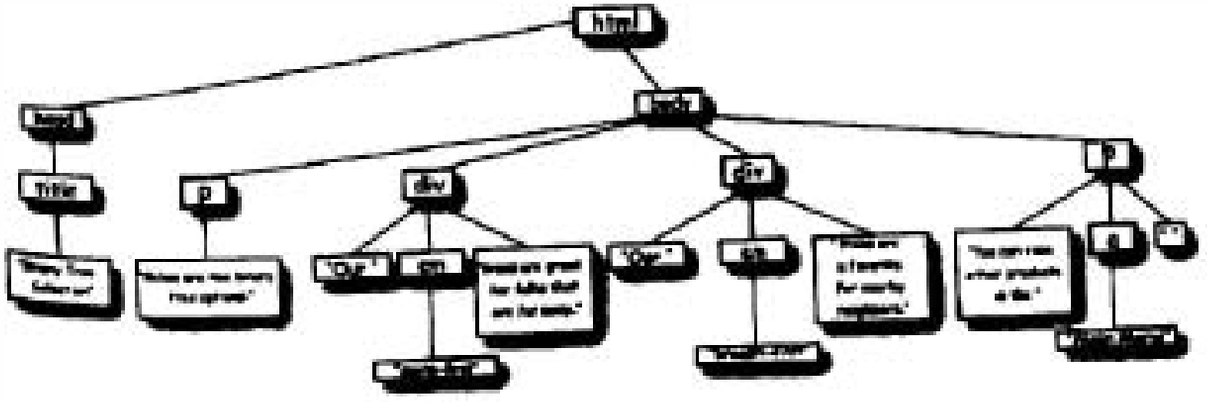
Молниеносная асинхронность

Где можно подождать? Извините, ждать некогда.
Это Веб, а не вокзал, и никто не желает листать старые журналы, пока сервер занимается своими делами. Вы уже знаете, как Ajax избавляет от перезагрузки страниц, но сейчас настало время включить асинхронность в список основных свойств веб-приложений. В этой главе вы узнаете, как отправить запросы пользователей серверу, и как дать пользователю возможность продолжить работу, пока он ждет ответа. А впрочем... забудьте. В этой главе ждать вообще не придется.

4 Модель DOM

О стрижке деревьев

Наша цель: простое и удобное обновление веб-страниц.
Переходим к написанию кода, обновляющего веб-страницы «на ходу».
С использованием модели DOM ваши страницы оживут новой жизнью, будут реагировать на действия пользователя и навсегда избавятся от лишних перезагрузок.
К концу главы вы научитесь добавлять, удалять и обновлять контент практически в любом месте веб-страницы.



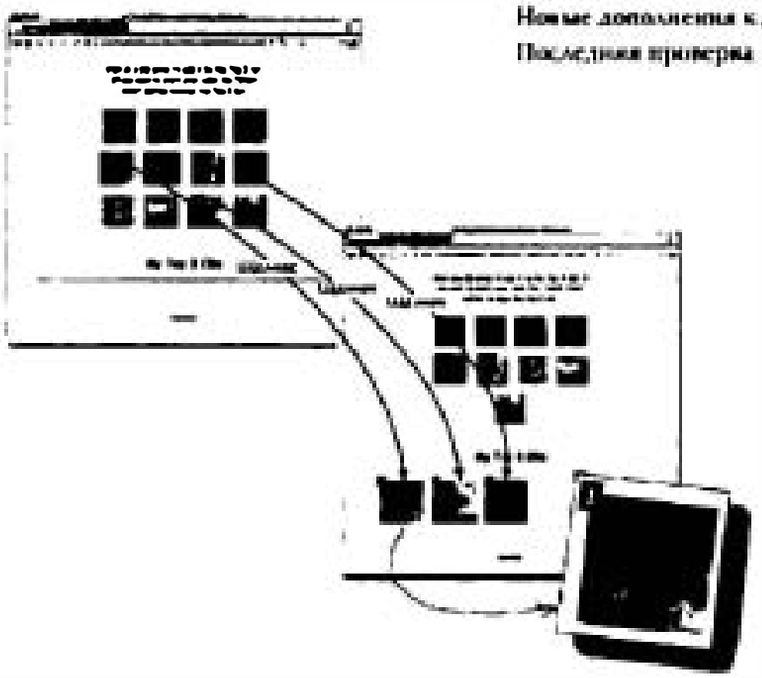
Разработка DOM-приложений

4.5

Вторая порция

Кому добавки DOM?

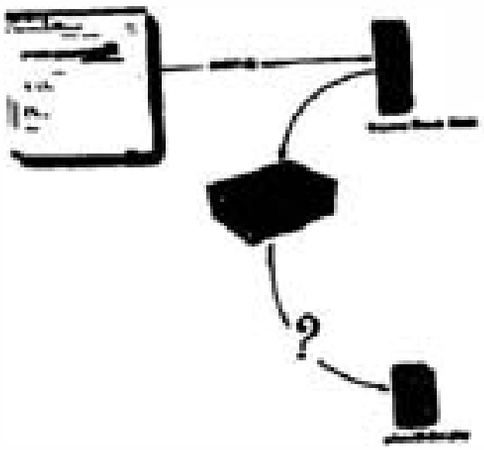
В предыдущей главе был приведен краткий курс по самой интересной технологии обновления веб-страниц: DOM (Document Object Model). Но скорее всего, ваш аппетит еще не удовлетворен, поэтому в этой главе мы используем уже имеющуюся информацию для написания приложения на базе DOM. Заодно мы рассмотрим несколько обработчиков исключений, научимся изменять стили узлов, и создадим дружественное, динамичное приложение. В этой главе ваши навыки работы с DOM поднимутся на совершенно новый уровень.

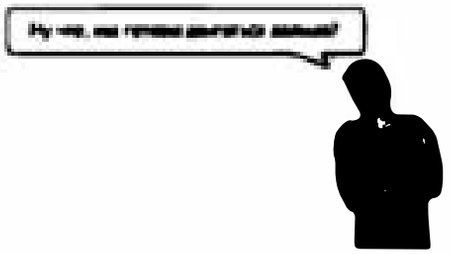


5 Запросы POST

Передача информации в POST

То, чего мы так долго ждали.
Наконец-то мы забудем о вопросах и научимся передавать произвольные данные серверу. Для этого придется немного потрудиться, но к концу этой главы ваши асинхронные запросы к серверу уже не будут ограничиваться категорией «без данных». Так что пристегните ремни — нам предстоит открыть страну POST, страну типов контента и заголовков запросов.





6 Запросы и ответы XML

Больше, чем можно выразить словами

Кажется, будто вас никто не слушает?

Когда вы пытаетесь с кем-то общаться, повседневного языка порой бывает недостаточно. До сих пор все наши запросы и ответы имели текстовый формат, но пришло время выразиться за рамки простого текста. В этой главе мы погрузимся в XML и узнаем, как заставить сервер передавать информацию, не ограничивающуюся простым текстом. А если этого мало, вы научитесь использовать XML и в своих запросах (хотя это далеко не всегда стоит делать). Приготовьтесь... После чтения этой главы *ваши запросы и ответы уже никогда не будут прежними*.

7 JSON или XML

Битва до победного конца

Битва до победного конца.

Помните времена юности, когда все разногласия решались кулаками, кунгфу и убогими подражаниями кун-фу? В этой главе мы вернемся в те дни, оставив позади все дружеские слова и золотое правило этики. XML и JSON, два разных формата отправки и приема данных в асинхронных запросах, должны решить свои проблемы на ринге.

п.1 Приложение 1: Дополнительные материалы

Только для вас: прощальный подарок от Head First Labs.
В этом приложении наши читатели найдут пять специальных подарков. Мы бы охотно остались с вами и рассказали еще много интересного, но сейчас вам пора взять все, что вы узнали, и самостоятельно отправиться в холодный, жестокий мир веб-программирования. И все же бросать вас без дополнительной подготовки не хочется, поэтому в приложении вы найдете пять основных тем, не вошедших в книгу.

п.2 Приложение 2: Инструментарий Ajax и DOM

И еще немного подарков
На этих последних страницах приводится служебный код, слишком сложный для того, чтобы рассматривать его в предыдущих главах. Но сейчас вы уже готовы к знакомству с этими служебными функциями Ajax и DOM.

а у

Как пользоваться этой книгой

Введение

В этом разделе мы ответим на один актуальный вопрос «Почему они поместили ЭТО в книгу по Ajax?»

Для кого написана эта книга?

Если вы ответите «да» на все следующие вопросы...

- Вы знаете HTML, немного CSS и немного JavaScript (экспертом быть не обязательно)?
- Вы хотите изучить, понять и запомнить Ajax, чтобы разрабатывать более динамические веб-приложения?
- Вы предпочитаете легкую застольную беседу скучным, сухим академическим лекциям?

...значит, эта книга для вас.

Кому не стоит читать эту книгу?

Если вы ответите «да» хотя бы на один из следующих вопросов...

- Вы совсем ничего не знаете о HTML, CSS и JavaScript? (Необязательно быть экспертом, но какой-то опыт определенно необходим. Если нет — начните с «Head First HTML and CSS», и только потом беритесь за эту книгу).
- Вы — квалифицированный разработчик Ajax, которому нужен справочник?
- Вы боитесь попробовать что-нибудь новое? Скорее пойдете к зубному врачу, чем наденете полосатое с клетчатым? Полагаете, что техническая книга с оруциллированными браузерами не может быть серьезной?

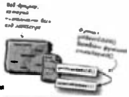
... эта книга не для вас.

[Покупайте эту книгу маркетинга: эта книга найдет любому, у кого есть деньги.]

Мы считаем, что читатель этой книги учится.

Что же необходимо для того, чтобы чему-то *научиться*? Сначала нужно получить информацию, а потом не забыть ее. Дело вовсе не в зубрежке. Согласно новейшим исследованиям в области когнитивистики, нейробиологии и образовательной психологии, для обучения требуется нечто большее, чем текст на странице. Мы знаем, как заставить ваш мозг работать.

Некоторые принципы серии «Head Rush»:



Сделайте материал наглядным. [illegible]

Используйте разговорный стиль. Недавние исследования показали, что студенты сдавали контрольные тесты на 40% лучше, если материал подавался в разговорном стиле (в отличие от формального стиля). [illegible]

Заставьте ученика интенсивно думать. [illegible]

Захватите — и удерживайте — внимание читателя. [illegible]

Сделай сам

Стимулируйте эмоции. [illegible]

Метапознание: учимся учиться

Если вы действительно хотите чему-то научиться, и притом научиться быстрее и глубже, подумайте над тем, как вы думаете. Осознайте суть осознания. Научитесь учиться.

Большинство читателей не посещало курсы метапознания или теории обучения. От нас *хотели*, чтобы мы учились, но редко учили нас учиться.

Но раз уж вы держите в руках эту книгу, предполагается, что вы хотите изучить Ajax. И скорее всего, вам не захочется тратить много времени. А поскольку речь пойдет о разработке приложений, прочитанное придется *запоминать* — но для этого материал нужно сначала понять.

Чтобы извлечь максимум пользы из этой книги (а также любой другой книги или учебного курса), научитесь управлять работой своего мозга

Фокус в том, чтобы ваш мозг воспринял изучаемый материал как нечто Очень Важное. Нечто критичное для вашего существования. Не менее важное, чем тигр. Иначе вам придется постоянно бороться со своим мозгом, который всеми силами постарается, чтобы новая информация не закрепилась.

Как же ЗАСТАВИТЬ ваш мозг отнестись к Ajax как к голодному тигру?

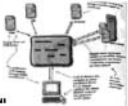
Есть два способа: один медленный и скучный, другой быстрый и эффективный. Медленный способ основан на обычном повторении. Конечно, вы знаете, что даже самую нудную тему можно запомнить, если раз за разом усердно вбивать ее в мозг. После достаточного количества повторений мозг скажет: «Мне это совсем не кажется важным, но раз уж он возвращается к этому снова и снова — наверное, это все-таки важно».

Более быстрый способ основан на выполнении любых операций, стимулирующих мозговую деятельность, и освоении различных типов мозговой деятельности. Все те приемы, о которых говорилось в предыдущей врезке, составляют немалую часть решения; уже доказано, что они помогают заставить мозг работать на вас. Например, исследования показали, что размещение слов внутри картинок, которые ими описываются (в отличие от других мест страницы — сносок, в заголовке или основном тексте), заставляет мозг анализировать связь между словами и рисунком, и как следствие — активизировать больше нейронов. Больше нейронов = выше вероятность того, что нужная информация доберется до мозга и, возможно, будет сохранена.

Разговорный стиль также способствует запоминанию; ведь в разговоре человек обычно следит за нитью обсуждения и готовится ответить. Интересно другое: для вашего мозга неважно, что «разговор» происходит между вами и книгой! С другой стороны, формальный и сухой стиль подачи материала для мозга равносилен лекции в аудитории, заполненной пассивными слушателями. Спите спокойно, дорогие читатели.

Но рисунки и разговорный стиль — всего лишь начало.

Вот что МЫ сделали:



Мы использовали *рисунки*, потому что мозг настроен на прием визуальной информации, а не текста. С точки зрения вашего мозга рисунок действительно стоит 1 024 слов. А если текст и рисунки используются совместно, мы внедряем текст в рисунки — мозг работает эффективнее, когда текст находится *внутри* рисунка, а не вынесен в подпись и не спрятан в тексте.

Мы использовали *повторение*, передавая одну и ту же мысль разными способами и по разным каналам; так повышается вероятность того, что информация будет закодирована в нескольких участках вашего мозга.

Мы использовали *необычные* концепции и рисунки, потому что мозг хорошо улавливает все новое. Рисунки и идеи надеялись хотя бы *некоторым эмоциональным содержанием*, потому что мозг тесно связан с биохимией эмоций. Именно так возникает чувство, что какая-то идея запоминается с большей вероятностью, даже если это чувство — всего лишь *юмор*, *сюрприз* или *интересный факт*.

Мы использовали *личный* разговорный стиль, потому что ваш мозг действует более активно, когда считает, что вы участвуете в беседу, а не являетесь пассивным слушателем. Это происходит даже тогда, когда вы читаете.

Мы включили в книгу свыше 40 *упражнений*, потому что ваш мозг лучше изучает и запоминает, когда вы что-то *делаете*, а не просто *читаете*. Упражнения непростые, но по силам каждому — то, что предпочитает большинство людей.

Мы использовали *разные стили обучения*, потому что вы можете предпочитать пошаговые описания, кто-то другой захочет сначала представить общую картину, а третьему достаточно увидеть пример кода. Однако независимо от вашего личного вкуса, *каждому* будет полезно увидеть один материал, представленный с разных точек зрения.

Обзор на 60 секунд

Мы включили информацию *для обоих полушарий мозга*, потому что чем активнее задействован ваш мозг, тем больше вероятность того, что вы что-то узнаете и запомните и тем дольше сможете концентрироваться на изучаемой теме. Поскольку работа одного полушария мозга часто дает другой стороне возможность отдохнуть, это позволит вам сохранить продуктивность в течение более долгого периода времени.

Мы приводили *истории* и упражнения, представляющие *различные точки зрения*, потому что ваш мозг лучше усваивает материал, когда ему приходится оценивать и делать выводы.

Мы задаем читателю *вопросы*, на которые не всегда существует простой, однозначный ответ, потому что ваш мозг лучше учится и запоминает за работой. Согласитесь, невозможно привести свое тело в спортивную форму, *наблюдая* за занимающимися в спортзале. Но мы постарались, чтобы ваши усилия были приложены в *правильном направлении*. Другими словами, вы не потратите *ни одной лишней дендрит* на неразумительные примеры или разбор сложного, перегруженного техническим жаргоном или слишком краткого текста.

Мы использовали *людей* в своих историях, примерах, рисунках и т. д., потому что... вы — человек. И ваш мозг обращает больше внимания на *людей*, а не на *предметы*.

Мы использовали принцип *80/20*. Предполагается, что если вы намерены серьезно освоить Ajax, одной книгой дело не ограничится. По этой причине мы не пытаемся рассказать *обо всем*. Речь пойдет только о том, что действительно *необходимо*.

А вот что можете сделать ВЫ, чтобы заставить свой мозг работать

Итак, мы свое дело сделали; остальное за вами. Следующие советы — всего лишь основа; прислушайтесь к своему мозгу и определите, что для вас работает, а что — нет.

Вырежьте и повесьте на холодильник

1. **Не торопитесь. Чем больше вы поймете, тем меньше придется запоминать.**
Просто читать недостаточно. Остановитесь и подумайте. Когда книга вас о чем-то спрашивает, не торопитесь переходить к ответу. Представьте, что кто-то *действительно* задал вам вопрос. Чем сильнее ваш мозг поработает над ответом, тем выше вероятность того, что вы запомните и усвоите материал.

2. **Выполняйте упражнения. Делайте заметки.**
Мы приводим упражнения, но не выполняем их за вас — это то же самое, что заниматься зарядкой за другого. И не ограничивайтесь чтением. Используйте карандаш. Доказано, что физическая активность *во время* обучения повышает его качество.

- **Читайте «Часто задаваемые вопросы».**
Все подряд. Время не содержит дополнительную информацию — *они являются частью основного материала!* Не пропускайте их.

- **Не читайте все подряд в одном месте.**
Встаньте, потянитесь, пройдитесь, смените кресло, перейдите в другую комнату. Ваш мозг (и тело) будут что-то ощущать, а ваше обучение не будет привязано к конкретному месту. Помните, что сдавать экзамен в спальне вам никто не разрешит.

- **Не читайте других книг перед сном. Или по крайней мере, других интересных книг.**
Часть обучения (особенно передача информации в долгосрочную память) происходит *после* того, как вы отложите книгу. Вашему мозгу необходимо время для дополнительной обработки информации. Если в это время поступит новая информация, часть того, что вы узнали, будет потеряна.

- **Пейте воду. Да побольше.**
Ваш мозг лучше всего работает при обилии жидкости. Дегидратация (которая может наступить даже до того, как вы почувствуете жажду) подавляет когнитивные функции. Пиво или что покрепче допускается только после сдачи экзамена.

- **Говорите о прочитанном. Вслух.**
Речь активизирует другую часть мозга. Если вы стремитесь что-то понять или запомнить на будущее, проговорите вслух. Или еще лучше — попытайтесь изложить на словах кому-то другому. Ваше учение пойдет быстрее, и вы обнаружите новые идеи, о которых даже не подозревали во время чтения.

- **Прислушивайтесь к своему мозгу.**
Следите за тем, чтобы мозг не перегружался. Если вы начинаете «скользить» глазами по страницам или забываете только что прочитанное, сделайте перерыв. С некоторого момента вам не удастся повысить скорость обучения, пытаясь забить голову новой информацией. Более того, это даже может повредить процессу.

- **Чувствуйте!**
Ваш мозг должен знать, что изучаемая информация *важна*. Вживайтесь в истории. Придумывайте собственные подписи к фотографиям. Посмеяться над неудачной шуткой лучше, чем вообще ничего не почувствовать.

Read Me

Это учебник, а не справочник. Мы намеренно убрали все, что может помешать усвоению материала. И при первом чтении необходимо начать с самой первой страницы, потому что в книге используются некоторые предположения относительно того, что вы уже видели и знаете.

Предполагается, что вы знакомы с HTML и CSS

Для изложения основ HTML и CSS потребовалась бы отдельная книга. Мы решили сосредоточиться на Ajax-программировании и обойтись без пересказа азов разметки и стилей, о которых можно узнать в других источниках.

Предполагается, что вы хотя бы видели код JavaScript

Для изложения основ... а впрочем, мы уже об этом говорили. Серьезно, JavaScript — нечто большее, чем простой сценарный язык, и мы не будем описывать всевозможные варианты его применения в книге. Вы узнаете, как язык JavaScript связан с Ajax-программированием, и научитесь использовать JavaScript для создания интерактивных веб-страниц и передачи запросов серверу.

Если вы еще не написали ни единой строчки на JavaScript, совершенно не разбираетесь в функциях и фигурных скобках, а то и вовсе никогда не программировали, найдите хорошую книгу по JavaScript и ознакомьтесь с темой. Впрочем, если вам все же захочется взяться за эту книгу, не стесняйтесь — но учтите, что основы будут излагаться довольно кратко.

Программирование на стороне сервера не рассматривается

Сейчас часто встречаются программы, работающие на стороне сервера и написанные на Java, PHP, Ruby, Python, Perl, Ruby on Rails, C# и других языках. Ajax-программирование работает во всех перечисленных языках, поэтому мы постарались представить некоторые из них в примерах книги.

Чтобы не отвлекать читателя от изучения Ajax, мы не стали тратить много времени на объяснение принципов использования серверных программ; мы приведем примеры серверного кода с несколькими примечаниями, но этим и ограничимся. Мы считаем, что приложения Ajax должны быть написаны так, чтобы они работали с любыми серверными программами; а еще мы считаем, что читатель достаточно сообразителен и сможет перенести уроки, усвоенные на примере PHP, на код Ruby on Rails или сервлеты Java.

В книге используются разные браузеры

Как ни противно, разные браузеры обрабатывают HTML-код, таблицы CSS и JavaScript совершенно разными способами. Если вы хотите стать квалифицированным Ajax-программистом, всегда тестируйте асинхронные приложения в разных современных браузерах. Все примеры в книге были протестированы в последних версиях Firefox, Opera, Safari, Internet Explorer и Mozilla. Если обнаружите какие-нибудь проблемы, сообщите нам... честное слово, это случайность.

Мы часто используем теги вместо имен элементов

Вместо того чтобы говорить об «элементе a», мы приводим имя тега — скажем, «элемент <a>». Строго говоря, такая терминология неточна (<a> — открывающий тег, а не полноценный элемент), но зато текст лучше читается.

Упражнения ОБЯЗАТЕЛЬНЫ для выполнения

Упражнения — это не дополнения для самых усердных, а часть основного материала книги. Одни из них способствуют запоминанию, другие улучшают понимание, а третьи помогают применять полученные знания на практике. ***Не пропускайте упражнения.***

Повторение — мать учения

У книг серии «Head First» есть одна отличительная особенность: мы хотим, чтобы вы *действительно* усвоили материал. А еще мы хотим, чтобы после чтения книги вы запомнили то, то узнали. Большинство справочников не ставит своей целью повторение и запоминание, но эта книга — учебник, поэтому многие концепции встречаются в тексте более одного раза.

Примеры сделаны как можно более компактными

Никому не захочется просматривать 200 строк листинга, чтобы найти две нужные строки. Практически все примеры в книге приводятся с минимальным контекстом, чтобы их содержательная часть была понятной и простой. Не ждите, что все приме ы будут защищены от ошибок или хотя бы логически завершены — они написаны специально для обучения и не всегда являются полнофункциональными программами.

Код примеров можно загрузить с сайта `http://www.headfirstlabs.com/books/hfajax/`.

У упражнений «Шевелим мозгами» нет ответов

У некоторых упражнений этой категории правильного ответа вообще не существует, а в других случаях вы должны сами решить, правилен ли ваш ответ, и при каких условиях (и это является частью процесса обучения). В некоторых упражнениях «Шевелим мозгами» присутствуют подсказки, указывающие нужное направление.

Технические рецензенты

Мне невероятно повезло с командой рецензентов. **Йоханнес де Йонг** (Johannes de Jong) считает, что он сделал для книги не так уж [illegible] — и он очень, очень сильно ошибается. Йоханнес начал процесс рецензирования, поддерживал его «на плаву» и выдавал шутки о виски в нужные моменты. **Полин Макнамара** (Pauline McNamara) работала над всеми книгами серии «Head First» (а теперь и «Head Rush»), потому что является рецензентом экстра-класса. Ее правка всегда была технически верной, оставаясь «крутой» во всех смыслах. И как ей это удается? **Валентин Кретаз** (Valentin Cretaz) выявил все мои грамматические ошибки и даже проследил за технической правильностью диаграмм. Потрясающее сочетание. **Кристин Стромберг** (Kristin Stromberg) подключилась к проекту на поздней стадии, и буквально проглотила все главы. Ее замечания были предельно уместными, и помогли сделать книгу более занятной и увлекательной. Поблагодарите ее, читатель... Я серьезно.

Эндрю Монкхаус (Andrew Monkhouse) помог мне в области Ajax-программирования. Отличная работа, Эндрю! И отдельное спасибо — **«Медведю» Бибо** (Bear Bibault). Жизнь вмешалась в наши планы, но я все равно ценю твое участие!

Надеюсь, эта книга вам понравится — она настолько же ваша, насколько и моя.

Эрик и Бет Фримен

При работе над такими книгами дружба играет не менее важную роль, чем знания и обучение. Если бы не **Бет Фримен** (Beth Freeman) и **Эрик Фримен** (Eric Freeman), моя жизнь была бы гораздо беднее. Они привели меня в семью «Head First», доверились мне и многому научили. Бет написала главу 9, а Эрик работал над обложкой и многими новыми элементами «Head Rush». Ребята, я ценю вашу дружбу даже больше, чем вашу помощь. До скорой встречи.

И еще многим другим...*

Издательство O'Reilly:

Майк Хендриксон (Mike Hendrickson) направлял работу над проектом, обходя все возможные препятствия. Майк, без тебя эта книга никогда бы не была написана. Ты помогал мне в трудные моменты, и я уверен, что не в последний раз. Когда мне приходится туго, приятно знать, что ты здесь.

Майк Лукидес

Иногда мне кажется, что глава почти закончена, но в ней чего-то не хватает. В таких случаях я обращаюсь к своему редактору **Майку Лукилесу** (Mike Loukides); он всегда знает, что именно нужно добавить. Это наша четвертая совместная книга, и мы уже планируем пятую и шестую.

Я сердечно благодарен своим коллегам из O'Reilly: **Грег Коррин** (Greg Corrin) руководил маркетингом, и особенно постарался в последнюю неделю. **Элли Фолькхаузен** (Ellie Volkhausen) создала первый вариант обложки, а **Майк Конке** (Mike Kohnke) и **Карен Монтгомери** (Karen Montgomery) переделали обложку для серии «Head Rush». **Колин Горман** (Colleen Gormann), спасибо за очередную великолепную редактуру — ты продолжаешь работать над книгами «Head First», и мне это нравится. **Сью Виллинг** (Sue Willing), **Рон Билодо** (Ron Bilodeau) и **Марлоу Шеффер** (Marlowe Shaeffer) заботились о том, чтобы принтер был доволен жизнью, а эта задача не из легких.

Грег Коррин

Имя «O'Reilly» стоит на обложке не случайно. **Тим O'Рейли** (Tim O'Reilly) рискнул взяться за эту серию, и как выяснилось — не напрасно. Тим, ты не представляешь, какая для меня честь работать над этой серией.

Кэтти Сьерра

Берт Бэйтс

Кэтти Сьерра и Берт Бэйтс:

Написать одну из книг «Head First» невозможно без продолжительного разговора с **Кэтти Сьерра** (Kathy Sierra) и **Бертом Бэйтсом** (Bert Bates) — гениальными создателями серии. Берт и Кэтти трудились до последней недели проекта, засиживались допоздна и беседовали со мной о том, как немного улучшить каждую страницу. Без вас эта книга была бы совершенно другой, но ваша *дружба для меня еще дороже!*

* Обычно благодарности обозначаются просто: мы признательны такому-то, будто каждый из упомянутых сыграл по крайней мере одну значимую роль. А может и больше, для разнообразия и значимости. Если вы хотите, чтобы вас поблагодарили в следующей книге, и у вас большая скидка — напишите нам.

1 Веб-приложения для нового поколения

Использование Ajax

Наводим глянец на веб-приложения. Надоели неуклюжие веб-интерфейсы и долгие перезагрузки страниц? Пора придать веб-приложениям стиль, присущий классическим настольным приложениям. О чем идет речь? О последней новинке в области веб-разработки Ajax (асинхронный JavaScript и XML) — ваш пропуск в мир [illegible], более *интерактивных*, *восприимчивых* и *удобных* в использовании. Итак, беритесь за пробную версию Ajax, прилагаемую к каждому экземпляру книги: мы займемся чистой веб-[illegible]

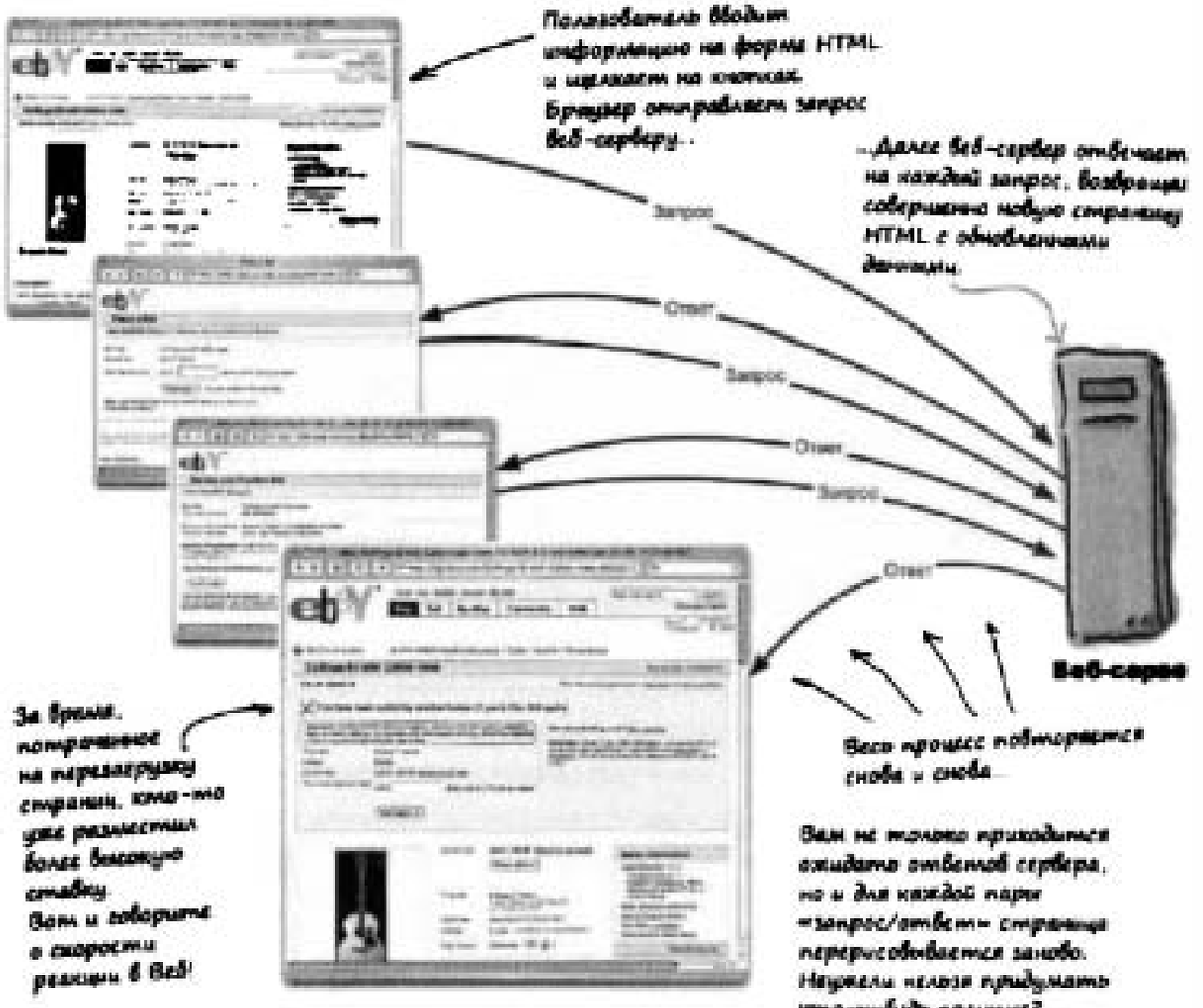
Веб: Перезагрузка

Вообще-то в этой главе мы избавимся от перезагрузки страниц...

Вашим клиентам надоело подолгу ждать при размещении заказов на сайте? Они жалуются на то, что при каждом нажатии кнопки происходит обновление страницы? Значит, настало время взяться за программирование и подняться на новый уровень. Добро пожаловать в следующее поколение веб-приложений — мир, где благодаря JavaScript, динамическому HTML и толике XML ваши приложения станут похожи на динамические, восприимчивые настольные приложения

Давайте посмотрим, к каким приложениям привыкли вы (и ваши клиенты):

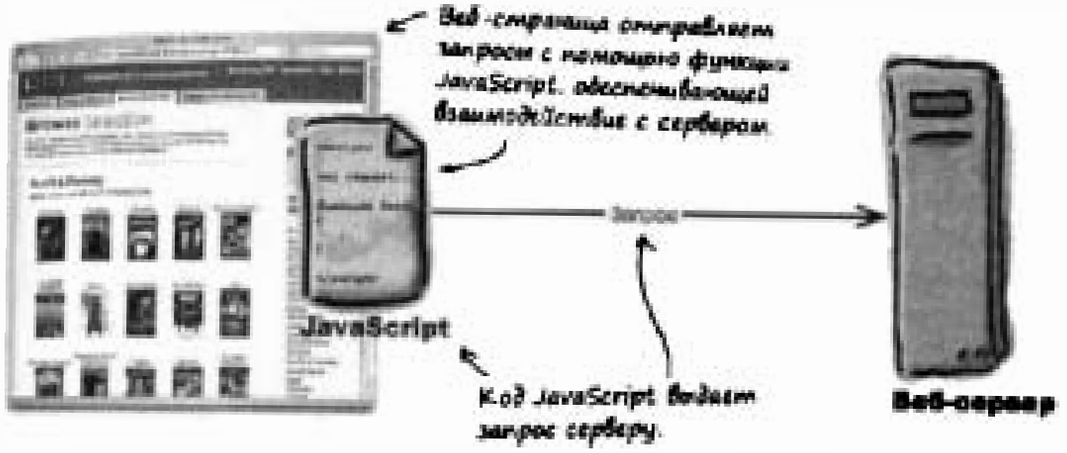
Старый подход (образца 1999 г.)



Добро пожаловать в новое тысячелетие!

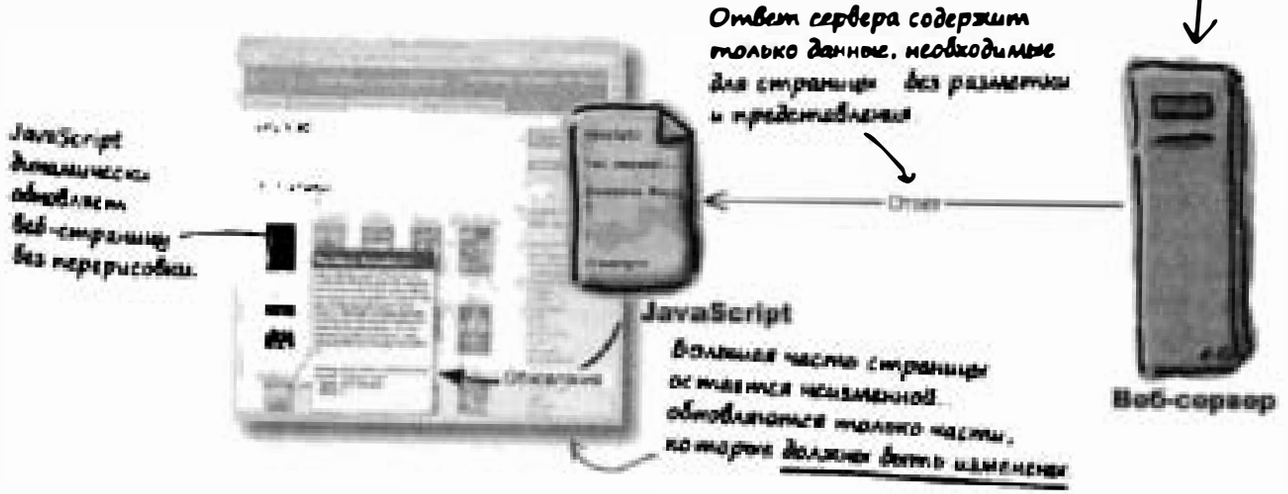
Программировать в старой модели «запрос/ответ» может кто угодно. Но если вам нужны **более быстрые приложения**, похожие на приложения настольных систем, потребуется нечто новое — **Ajax**, принципиально новый подход к веб-программированию.

Ждать больше не придется...



Для веб-сервера ничего не изменилось: он по-прежнему отвечает на каждый запрос, как и прежде

...если вы используете технологию Ajax



Не так быстро, приятель... ты говоришь, что новые приложения начинают быстрее реагировать на действия пользователя. Но ведь нам все равно придется дождаться, пока завершится выполнение кода JavaScript, верно?

Приложения Ajax к тому же являются асинхронными.

Вспомните, о чем мы говорили — о применении Ajax для построения супер-веб-приложений. Пока что мы узнали, что Ajax-приложения могут общаться с веб-сервером без чистой перезагрузки браузера и перерисовки целой страницы. Однако Ajax содержит множество других элементов, улучшающих пользовательский интерфейс.

Помимо предотвращения раздражающей перезагрузки страниц, JavaScript в Ajax-приложениях взаимодействует с веб-сервером *асинхронно*. Иначе говоря, JavaScript отправляет запрос серверу, а вы в это время можете продолжать вводить данные в веб-формах и даже щелкать на кнопках — все это время веб-сервер будет работать в фоновом режиме. Когда его работа завершится, ваш код обновляет только изменившуюся часть страницы — но вам при этом ждать не придется. В этом и заключается мощь асинхронных запросов! Объедините ее с обновлением страниц без перезагрузки, и вы поймете, на что способны Ajax-приложения.

Не огорчайтесь, если вы что-то не поняли; тема асинхронного программирования гораздо подробнее рассматривается в следующих главах.

Часто задаваемые вопросы

В: Короче, я совсем запутался. Выходит, теперь мы не используем модель «запрос/ответ»?

О: Страницы, как и прежде, выдают запросы и получают ответы. Просто мы используем несколько иной метод выдачи запросов и обработки ответов — запрос создается кодом JavaScript, а не отправкой веб-формы.

В: А почему нельзя просто отправить форму? Что нам реально дает Ajax?

О: Код JavaScript в Ajax-приложениях отправляет запросы, которые должны быть обработаны сервером, но не дожидается ответа. Что еще лучше, JavaScript также может работать с ответом сервера вместо того, чтобы перезагружать целую страницу, когда сервер завершит обработку ответа.

В: Как же сервер получает обратно ответ?

О: Именно в этом проявляется асинхронная природа Ajax. Получив от сервера ответ, JavaScript может обновить страницу новыми данными, изменить изображение и даже перенаправить пользователя на новую страницу. И пока все это происходит, пользователю совершенно не придется ждать.

В: Значит, Ajax стоит использовать для любых запросов?

О: Во многих ситуациях лучше ограничиться традиционным веб-программированием. Например, если на форме заведены все данные, можно разрешить пользователю щелкнуть на кнопке Submit и отправить всю форму на сервер без использования Ajax.

Однако для большинства задач динамической обработки страниц (таких, как смена графики, обновление полей и реакция на действия пользователя) стоит применять Ajax. Если требуется обновить только часть страницы, Ajax позволяет это сделать.

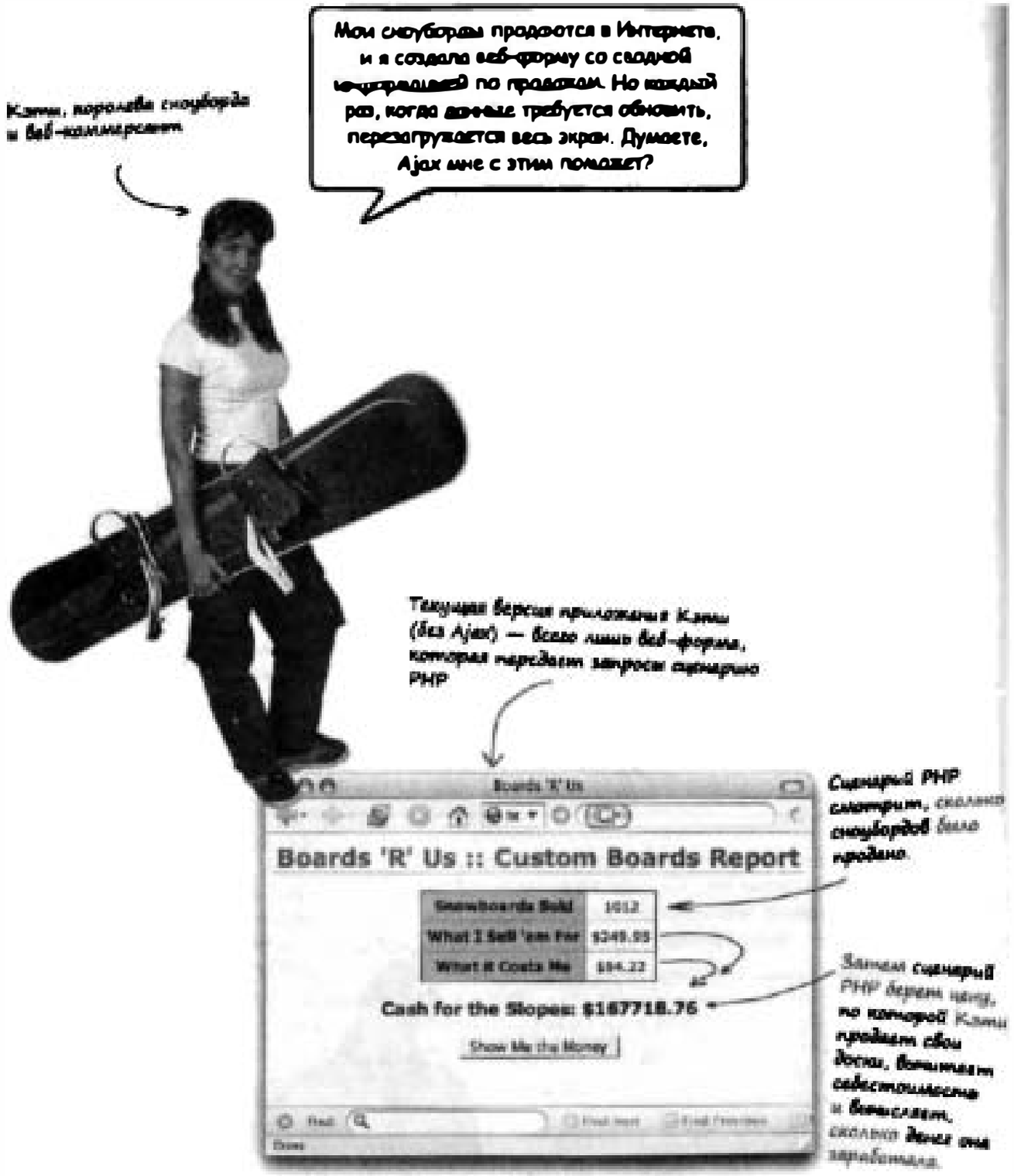
В: Вы что-то сказали про XML?

О: Иногда JavaScript обменивается данными с серверами в формате XML. Впрочем, использовать XML в запросах нужно не всегда. В следующих главах мы потратим достаточно много времени, выясняя, когда и как уместно использовать XML.

В: AJAX — это сокращение от «Asynchronous JavaScript and XML» («асинхронный JavaScript и XML»)?

О: Ajax — название, а не сокращение. Хотя очень похоже... И буквы подходят... хм-м. В общем, не спрашивайте, это не мы придумали!

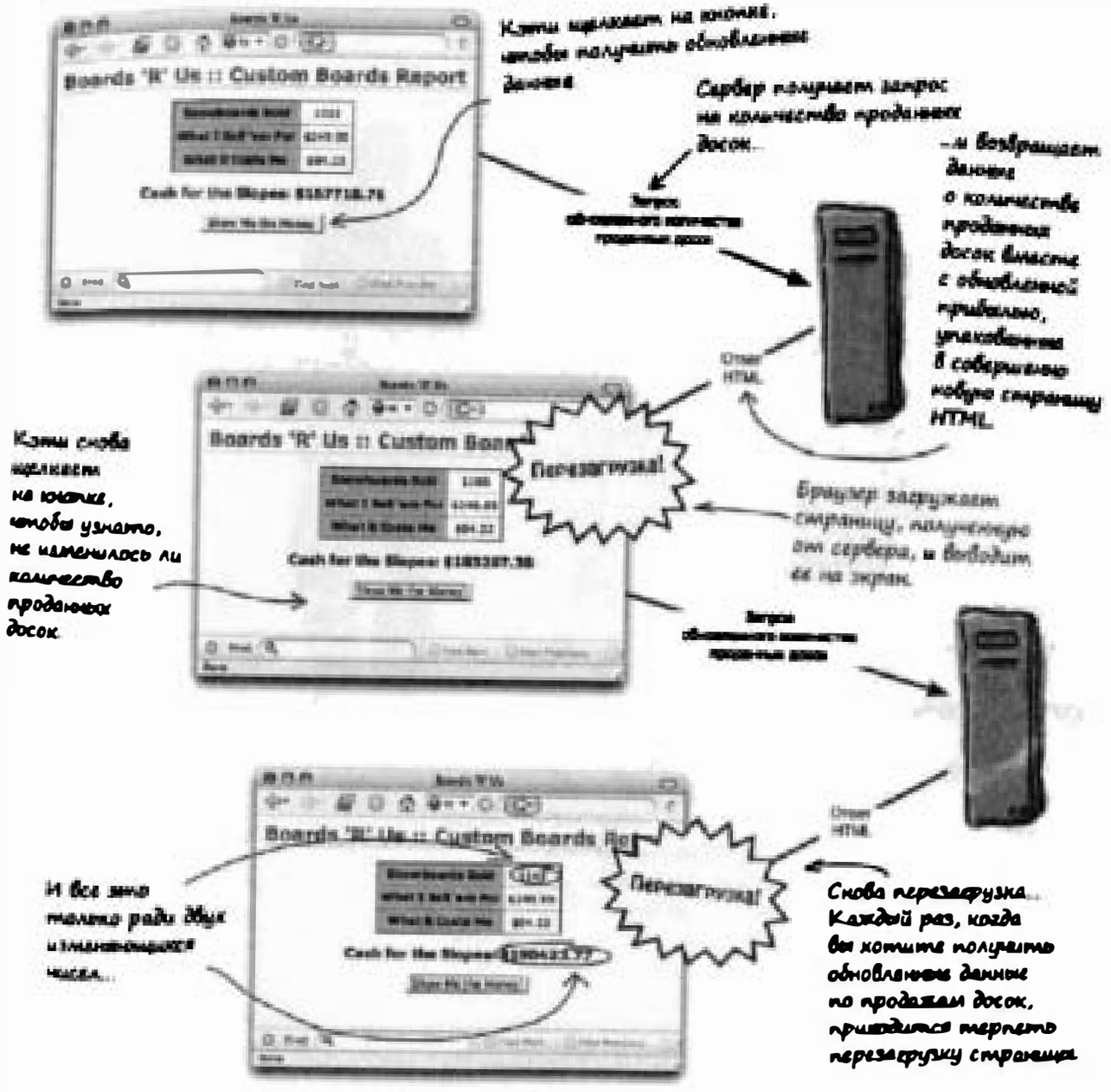
Джесси Джеймс Гарретт (Jesse James Garrett), предложивший термин «Ajax», утверждает, что это не сокращение. Пойди разберись!


Мои сноуборды продаются в Интернете, и я создала веб-форму со сводной информацией по продажам. Но каждый раз, когда данные требуется обновить, перезагружается весь экран. Думаете, Ajax мне с этим поможет?
Кэти, королева сноубордов и веб-коммерции
Текущая версия приложения Кэти (без Ajax) — всего лишь веб-форма, которая передает запросы сценарию PHP
Boards 'R' Us
Boards 'R' Us :: Custom Boards Report
Snowboards Sold
1012
What I Sell 'em For
$349.99
What it Costs Me
$94.22
Cash for the Slopes: $187718.76
Show Me the Money
Сценарий PHP смотрит, сколько сноубордов было продано
Затем сценарий PHP берет цену, по которой Кэти продает свои доски, вычитает себестоимость и вычисляет, сколько денег она заработала.

«Перезагрузки? Нам эта гадость не нужна».

Кэти не раздражает так, как перерисовка всей страницы при каждом нажатии кнопки или вводе значения. В отчете Кэти изменяются лишь несколько цифр, но перерисовывается вся страница.

Для начала попробуем разобраться, откуда берутся эти перезагрузки...



Ajax приходит на помощь

Теперь видите, в чем корень проблемы? Каждый раз, когда Кэти хочет узнать количество проданных досок, весь экран перерисовывается, и у нее начинает рябить в глазах.

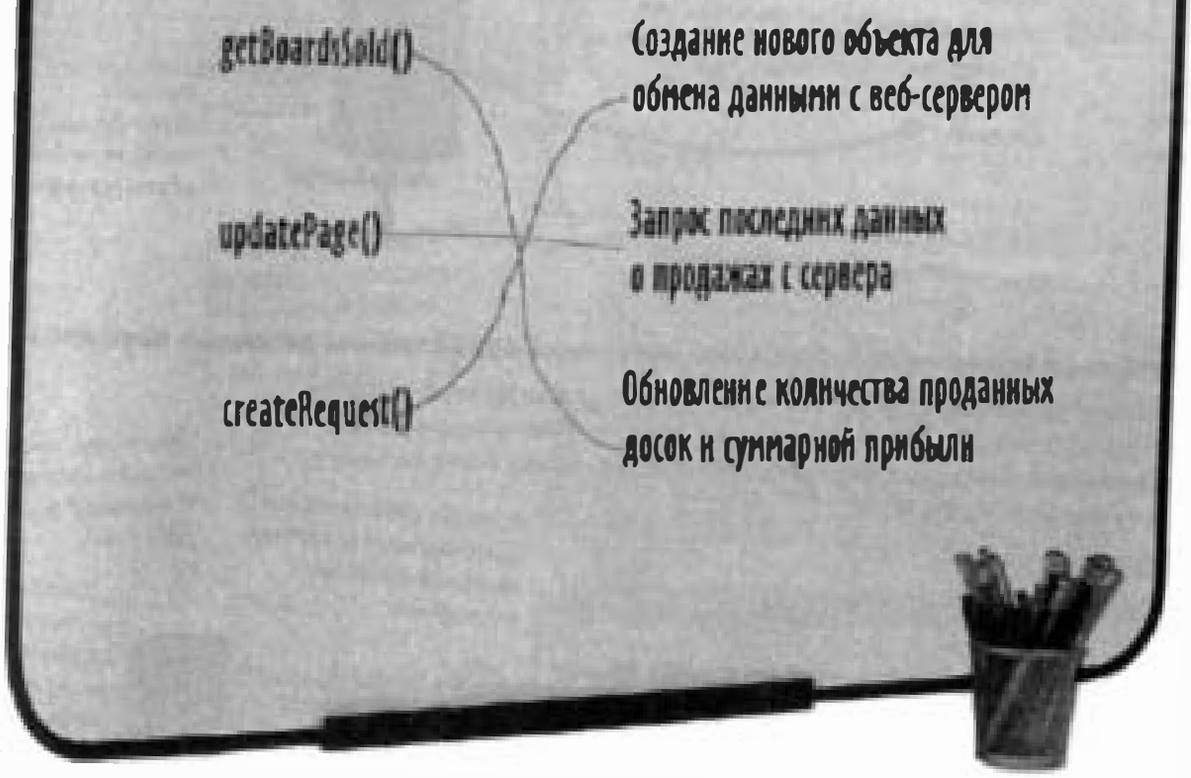
Исправляем веб-отчет

Давайте изменим отчет Кэти так, чтобы Ajax запрашивал обновленное количество проданных досок. Далее мы будем получать ответ с сервера и обновлять веб-страницу с использованием JavaScript и динамического кода HTML. Частые перезагрузки уходят в прошлое, а Кэти снова счастлива.

Может, нам даже достанется бесплатная доска.

Сделайте это...

Для превращения отчета Кэти в Ajax-приложение нам потребуется пара функций JavaScript. Ниже приведены имена трех функций JavaScript. Соедините каждую функцию с описанием того, что она, по вашему мнению, должна делать в итоговой версии приложения Boards.



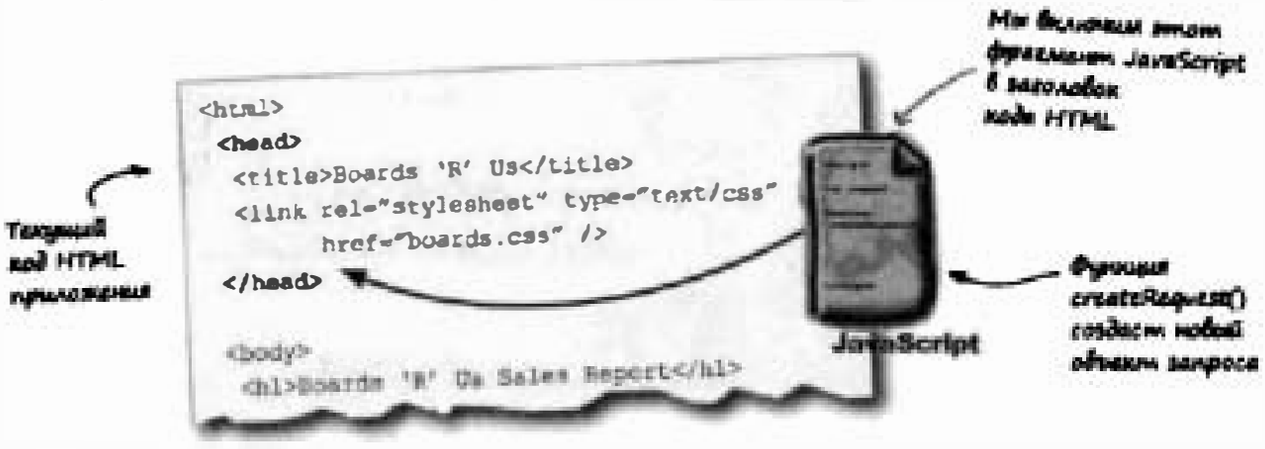
→ Ответ на 63 с.

Переработка отчета

Давайте переделаем отчет Кэти на базе Ajax. Ajax поможет избавиться от перезагрузки страниц и сократить объем данных, передаваемых с сервера. Вот что мы собираемся сделать в оставшейся части этой главы:

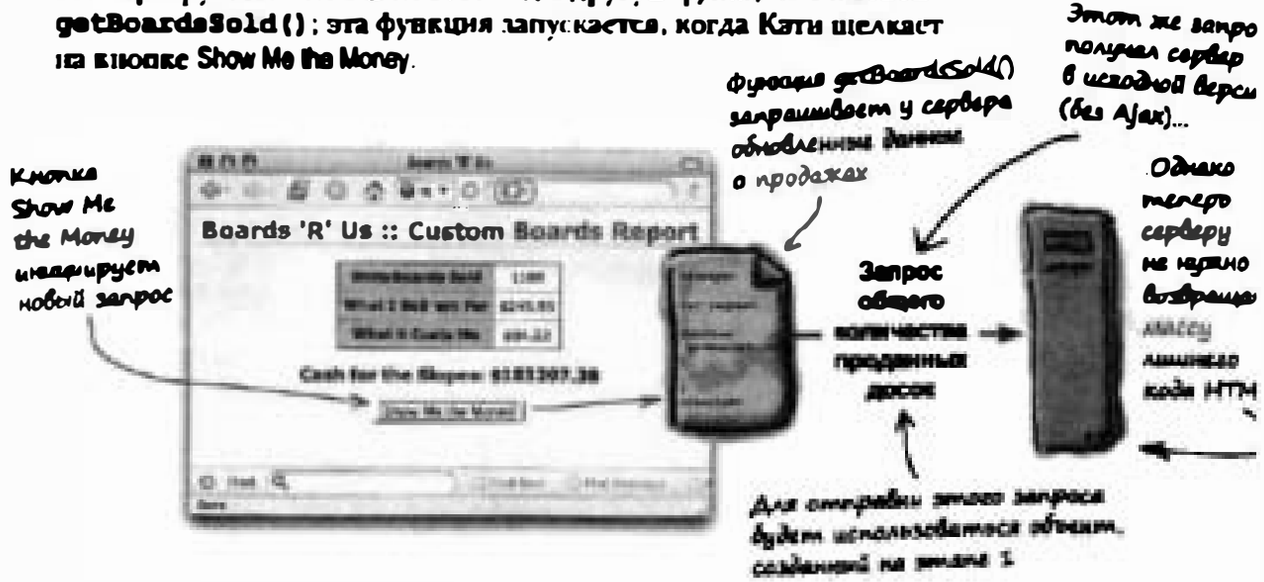
1 Создание нового объекта для отправки запросов серверу

Для начала нам понадобится функция JavaScript для создания объекта, предназначенного для передачи запросов серверу и приема ответов. Назовем эту функцию **createRequest()**.



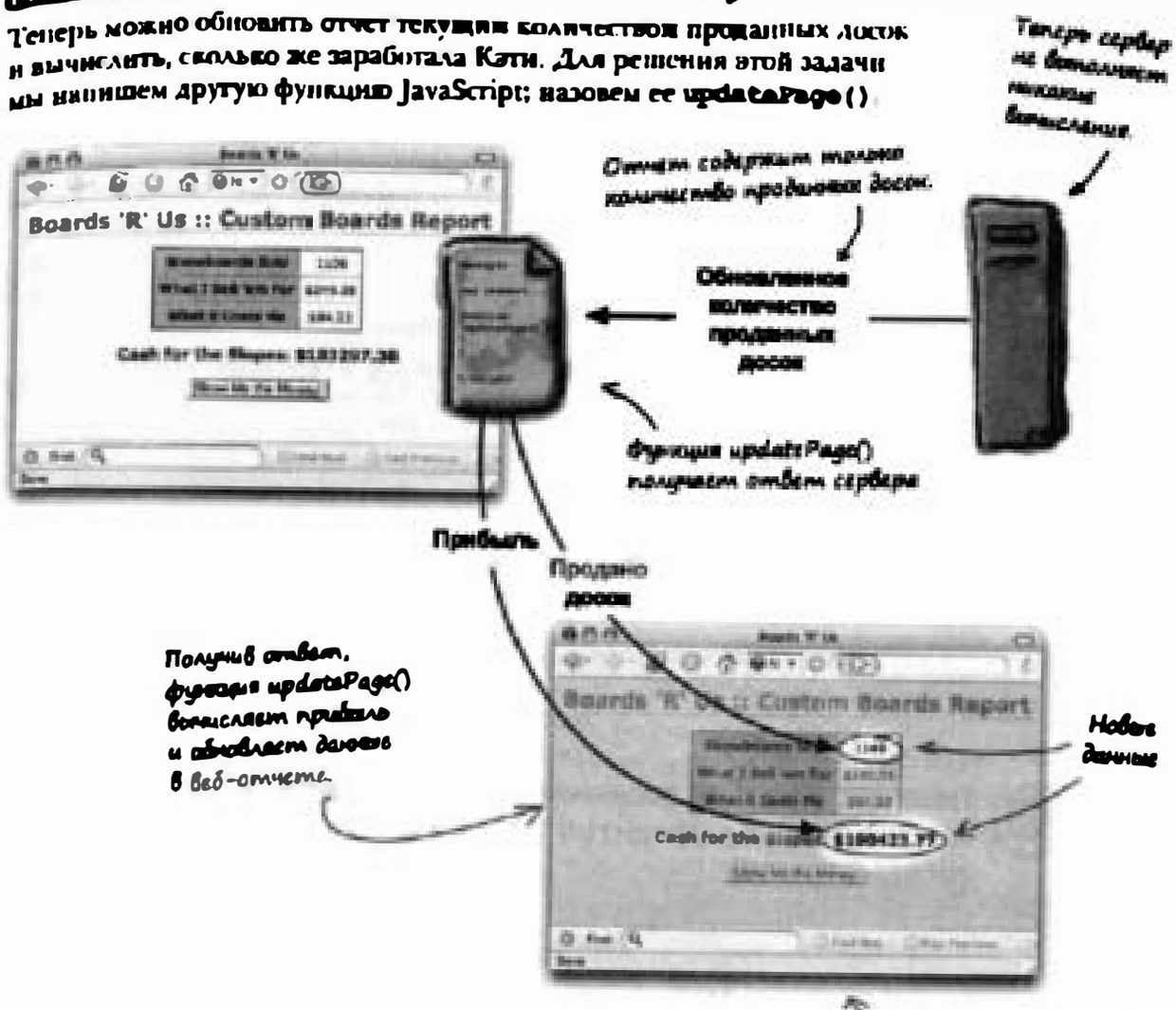
2 Написание функции JavaScript для получения обновленных данных о продажах

Объект, созданный на этапе 1, используется для отправки запроса веб-серверу. Мы помещаем этот код в другую функцию с именем **getBoardsSold()**; эта функция запускается, когда Кэти щелкает на кнопке Show Me the Money.



3 [redacted] отчета Кэти с использованием JavaScript

Теперь можно обновить отчет текущим количеством проданных досок и вычислить, сколько же заработала Кэти. Для решения этой задачи мы напишем другую функцию JavaScript; назовем ее **updatePage()**.



Что-что?

Не беспокойтесь, если вы не до конца поняли, что здесь происходит... особенно в том, что касается запроса. Все это будет подробно рассмотрено в других местах. Просто попытайтесь в общих чертах понять, как работают приложения Ajax. Обратите особое внимание на то, что веб-страница использует JavaScript для выдачи запроса, а сервер возвращает одно-единственное число вместо кода HTML.

...через несколько страниц. А пока немного отвлечемся...

Основные моменты: глава 1

Вероятно, вы подумали: «Что эта сводка делает в начале главы? Непорядок». Вы правы... но ведь это особая книга, вы помните? Прежде чем двигаться дальше, задержитесь и прочитайте каждый основной момент вслух.

Затем перейдите на следующую страницу и приготовьтесь загрузить эти концепции в свой мозг. Мы подробно рассмотрим каждую из них на оставшихся страницах главы.

- Для отправки запросов в асинхронных приложениях используются объекты JavaScript, а не отправка формы.
- Запросы и ответы обрабатываются браузером, а не кодом JavaScript.
- Получив ответ на асинхронный запрос, браузер «вызывает» код JavaScript и передает ему ответ сервера.

Мы не шутим... Не переходите на следующую страницу, пока не прочитаете эту сводку ВСЛУХ. Уверяем, никто не подумает, что вы с приветом (ну может, кто-нибудь и подумает, но наверняка порадуется вашему уму!)

Освежимся HTML

Позабыли все эти и ? На следующей странице мы займемся кодом HTML, поэтому я кратко напомню два самых интересных элемента HTML, с которыми вы столкнетесь.

 — контейнерный элемент, содержащий взаимосвязанные элементы. Стиль всех вложенных элементов определяется одним правилом CSS.

```
<div id="menu">
 <a href="home.html">home</a>
 <a href="books.html">writing</a>
 <a href="links.html">resources</a>
 <a href="lib.html">library</a>
</div>
```

Включай ссылку на атрибут id элемента в CSS-файл, вы одновременно определяете стили всех вложенных элементов.

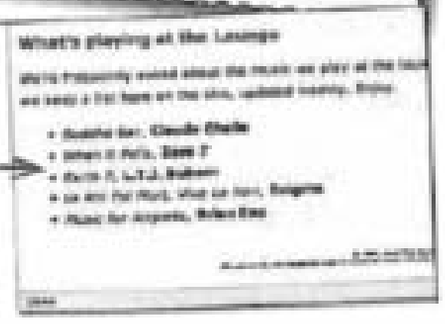
Используйте для группировки похожих элементов, имеющих сходное назначение.

Элемент позволяет отделить фрагмент встроенного текста от окружающих элементов. Стиль элементов определяется в CSS, что позволяет легко обеспечить визуальное выделение текста.

Элементы определяют фрагменты текста, но не создают новых абзацев или блоков.

```
<ul>
 <li><span class="cd">Buddha Bar</span>,
   <span class="artist">Claude Challe</span></li>
 <li><span class="cd">When It Falls</span>,
  <span class="artist">Zero 7</span></li>
...
```

Элементы не создают новых блоков текста, однако их стили также определяются в CSS.



HTML-код приложения

Начнем с самого существенного. У Кэти уже есть веб-страница. Давайте посмотрим, как она выглядит, а уже потом начнем добавлять в нее код JavaScript, о котором мы говорили.

```
<html>
 <head>
  <title>Boards 'R' Us</title>
  <link rel="stylesheet" type="text/css" href="boards.css"/>
 </head>

 <body>
  <h1>Boards 'R' Us Sales Report</h1>
  <div id="boards">
   <table>
    <tr><th>Snowboards Sold</th>
     <td><span id="boards-sold">1012</span></td></tr>
    <tr><th>What I Sell 'em For</th>
     <td>$<span id="price">249.95</span></td></tr>
    <tr><th>What it Costs Me</th>
     <td>$<span id="cost">84.22</span></td></tr>
   </table>
   <h2>Cash for the Slopes:
    $<span id="cash">167718.76</span></h2>
   <form method="GET" action="getUpdatedBoardSales.php">
    <input value="Show Me the Money" type="submit" />
   </form>
  </div>
 </body>
</html>
```

Кэти прочитала книгу «Head First HTML with CSS and XHTML», и поэтому прекрасно разбирается в новейших веяниях...

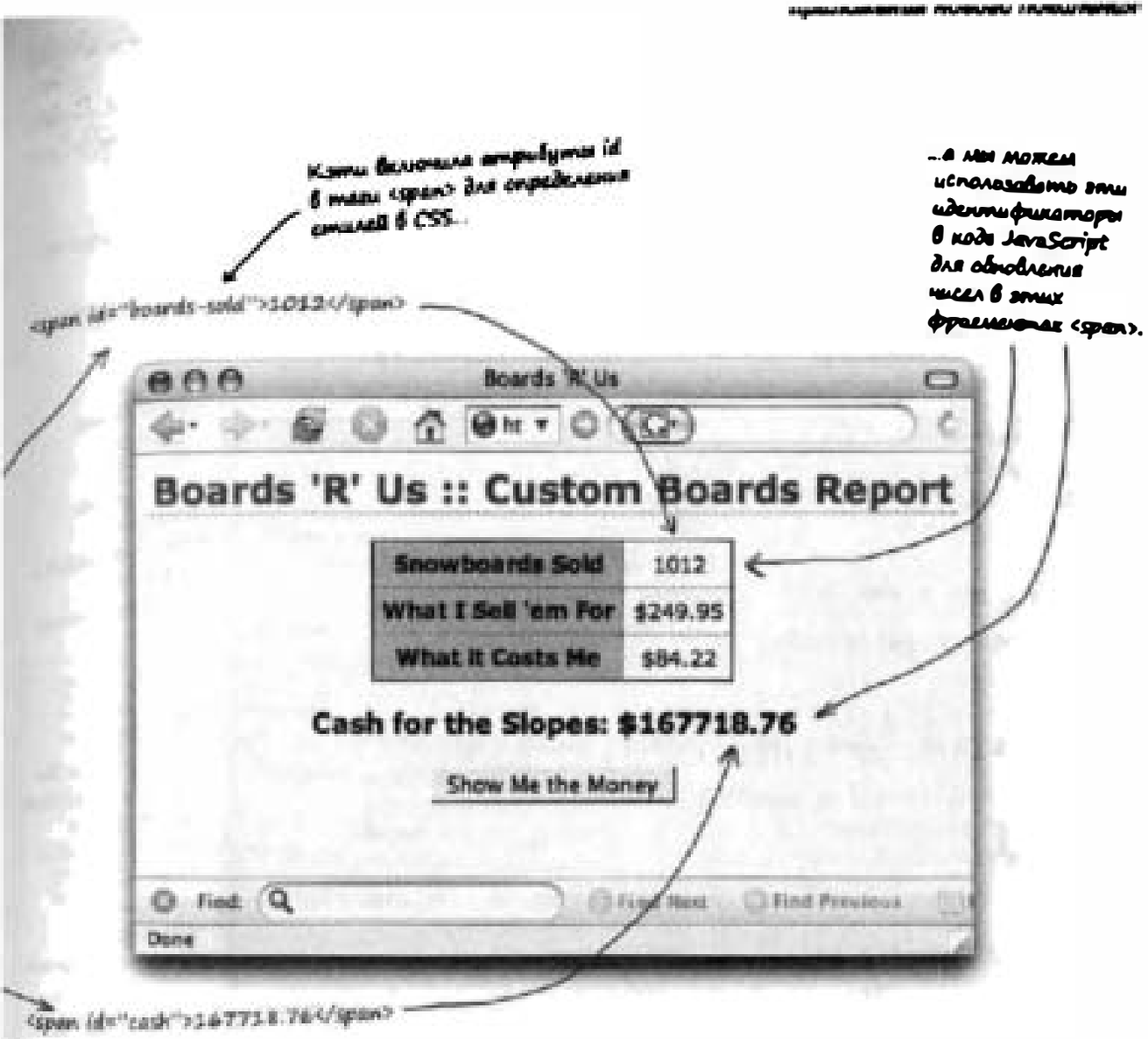
Сюда надо будет вставить теги <script>...

Кэти использует внешнюю таблицу стилей CSS.

Вот количество проданных досок, которое нам нужно обновить

...а теперь можно рассчитать новую прибыль

Кнопка, на которой Кэти щелкнет для обновления данных. Пока кнопка отправляет форму серверу, но это скоро изменится.



Просто сделайте это

Пора заняться делом. Загрузите примеры книги по адресу **http://www.headfirstlabs.com** и найдите папку **chapter01/boards/**. Откройте файл **boards.html** в браузере. Он должен выглядеть точно так же, как приведенная ранее веб-страница Кэти.

Шаг 1: Создание объекта запроса

Вернемся к обновлению веб-чата Кэти. Для начала нам понадобится функция, которая создает новый объект для передачи запросов серверу. Задача усложняется тем, что в разных браузерах этот объект создается разными способами. Чтобы упростить вашу работу, мы написали для вас «готовый» код JavaScript. Каждый раз, когда вы встречаете значок , знайте, что вам придется принять код «на веру» — как, например, код создания объекта для создания запросов к серверу. Поверьтесь нам, этот код будет подробно расписан в следующих главах. А пока просто введите его и посмотрите, что он делает.

```
var request = null;                          // Переменная для хранения объекта запроса
function createRequest() {
  try {
    request = new XMLHttpRequest();          // В этой строке мы пытаемся создать объект запроса
  } catch (trymicrosoft) {                   // Тип объекта запроса
    try {
      request = new ActiveXObject("Msxml2.XMLHTTP");   // Эти две строки тоже пытаются создать объект запроса,
    } catch (othermicrosoft) {                         // но с применением способа, работающего в Internet Explorer
      try {
        request = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (failed) {
        request = null;                      // Если возникнут проблемы, эта строка гарантирует, что переменная запроса останется равной null
      }
    }
  }
  // Теперь можно проверить, остается ли переменная равной null. Если остается, значит, в программе что-то пошло не так...
  if (request == null)
    alert("Error creating request object!");   // ...и мы можем выдать пользователю сообщение об ошибке при помощи функции JavaScript alert()
}
```

Готовый код JavaScript

Если вы не хотите набирать этот код, найдите его в файле create-request.txt из папки chapter01/boards

Не будет лучше, если вы все же наберете его вручную. Это поможет вашему мозгу привыкнуть к написанию Ajax-приложений

Включите этот фрагмент в элемент <head> кода HTML в файле boards.html. Не забудьте ввести теги <script> и </script>.

```
<head>
 <title>Boards 'R' Us</title>
 <link rel="stylesheet" type="text/css" href="boards.css" />
 <script language="javascript" type="text/javascript">

 </script>
</head>
```

Здесь должен находиться наш код JavaScript

Эта строка сообщает браузеру, что далее следуют сценарный код, написанный на языке JavaScript

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

В: Мне положено понимать, что именно делает этот код?

О: Нет, понимать все до мелочей не обязательно. Пока достаточно иметь общее представление о том, как выглядит этот код; мы подробно рассмотрим его в главе 2.

В: Что такое null? Я увидел это слово в готовом коде JavaScript, но плохо понимаю, что оно означает.

О: Специальное ключевое слово null означает *неопределенное значение*, или несуществующую ссылку. Другими словами, это значение, которого нет. Не путайте null с «0» или «false» — оба эти значения являются определенными и отличными от null.

В: Как называется объект запроса — «XMLHttpRequest» или «ActiveXObject»?

О: Возможен любой вариант. Эта тема будет подробно рассмотрена в следующей главе, а пока в двух словах скажу, что в разных типах браузеров приходится использовать разные имена объектов.

В: Значит, для работы с Ajax-приложениями у пользователя должен быть установлен определенный браузер?

О: Нет — код должен работать в любом браузере с включенной поддержкой JavaScript. Так что Ajax-приложения нормально работают в Firefox, Mozilla, Internet Explorer, Safari, Netscape и Opera.

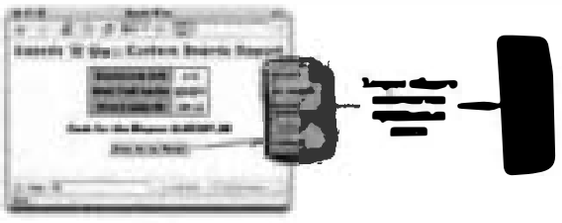
В: А если пользователь отключит JavaScript в своем браузере?

О: К сожалению, для работы Ajax-приложений обязательна поддержка JavaScript. Следовательно, пользователи, отключившие JavaScript, не смогут работать с вашими Ajax-приложениями.

Обычно по умолчанию поддержка JavaScript в браузере включена. Вероятно, отключивший ее знает, что делает, и может включить ее снова для запуска вашего Ajax-приложения.

Шаг 2: Запрос обновленных данных

После того как объект запроса будет создан функцией **createRequest()**, можно переходить к следующему шагу: написанию функции JavaScript **getBoardsSold()**. Эта функция использует созданный объект для запроса общего количества проданных досок с сервера. Давайте разберемся, что должна делать эта функция, а затем вернемся к программированию приложения Кэти. Помните нашу диаграмму? Вот шаг, над которым мы сейчас работаем:



А вот что необходимо сделать, чтобы функция **getBoardsSold()** выполнила свою работу:

Для решения этой задачи можно воспользоваться готовым кодом JavaScript на предыдущих страницах

ⓐ **Создать новый объект запроса вызовом функции createRequest().**

ⓑ **Определить, к какому URL-адресу необходимо подключиться для получения обновленной информации о количестве проданных досок.**

Кэти уже закодировала URL в своей форме, так что здесь все просто

ⓒ **Настроить объект запроса для установления связи.**

А здесь будет использоваться объект запроса, созданный функцией createRequest()

ⓓ **Запросить обновленное количество проданных досок.**

Полученное число будет использоваться позднее — на шаге 3, при обновлении страницы (об этом чуть позже)

Просто сделайте это

Откройте файл boards.html и введите новую функцию JavaScript с именем **getBoardsSold()** сразу же после **createRequest()**. Посмотрите, удастся ли вам включить в **getBoardsSold()** строку JavaScript для создания объекта запроса (*шаг а* в нашей схеме). Если возникнут затруднения, обратитесь к ответам в конце главы (с. 61).

Обязательно внесите эти изменения в свой экземпляр boards.html перед тем, как переходить к следующей странице.

Создание функции getBoardsSold()

Если вы выполнили упражнение с предыдущей страницы, то boards.html будет содержать код следующего вида:

Полностью, весь код JavaScript размещается между тегами <script> и </script>

```
<script language="javascript" type="text/javascript">
  var request = null;

  function createRequest() {
    // pre-assembled JavaScript
  }

  function getBoardsSold() {
    createRequest();
  }
</script>
```

Переменная для объекта запроса; после вызова createRequest() указывает на созданный объект

Функция createRequest(), которую вы должны были добавить несколько страниц назад

Новая функция JavaScript...

...а здесь создается объект запроса

Отправка запроса по нужному URL-адресу

Итак, что теперь? У нас появился объект, который может использоваться для запроса к серверу на получение количества проданных досок, но как передать этот запрос? Сначала нужно сообщить объекту, куда — а вернее, какой программе на сервере Кэти — направляется этот запрос. Итак, мы должны указать URL сценария, работающего на веб-сервере.

Где взять URL? В веб-форме Кэти:

URL сценария PHP, к которому обращен запрос

```
<form method="GET" action="getUpdatedBoardSales.php">
  <input value="Show Me the Money" type="submit" />
</form>
```

Элемент <form> в коде HTML веб-страницы Кэти

ничего страшного... просмотрите и продолжайте читать. Вам не обязательно понимать этот сценарий, чтобы изучать Ajax или следить за приводимыми примерами

PHP ...с первого взгляда

Давайте пробежимся по PHP-коду, который Кэти использует для своего приложения Boards. Мы не будем подробно разбираться, как работает сценарий... но вот что происходит при выдаче запроса на обновленное количество проданных досок.

```
<?php

// Connect to database
$conn = @mysql_connect("mysql.headfirstlabs.com",
                        "secret", "really-secret");
if (!$conn)
  die("Error connecting to MySQL: " . mysql_error());

if (!mysql_select_db("headfirst", $conn))
  die("Error selecting Head First database: " . mysql_error());

$select = 'SELECT boardsSold';
$from   = ' FROM boardsrus';
$queryResult = @mysql_query($select . $from);
if (!$queryResult)
  die('Error retrieving total boards sold from database.');

while ($row = mysql_fetch_array($queryResult)) {
  $totalSold = $row['boardsSold'];
}
$price = 249.95;
$cost = 84.22;
$cashPerBoard = $price - $cost;
$cash = $totalSold * $cashPerBoard;

mysql_close($conn);

?>

<html>
 <head>  <title>Boards 'R' Us</title>
  <link rel="stylesheet" type="text/css" href="boards.css" />
 </head>

 <body>
  <h1>Boards 'R' Us :: Custom Boards Report</h1>
```

Начальная часть сценария подключается к базе данных

Этот фрагмент сценария обеспечивает выборку обновленного количества проданных досок из базы данных

При желании эту информацию тоже можно сохранить в базе данных

Здесь вычисляется прибыль для одной доски

Код HTML, который мы уже рассматривали

```
  <div id="boards">
   <table>
    <tr><th>Snowboards Sold</th>
     <td><span id="boards-sold">
<?php
  print $totalSold;
?>
    </span></td></tr>
    <tr><th>What I Sell 'em For</th>
     <td>$<span id="price">
<?php
  print $price;
?>
    </span></td></tr>
    <tr><th>What it Costs Me</th>
     <td>$<span id="cost">
<?php
  print $cost;
?>
     </span></td></tr>
   </table>
   <h2>Cash for the Slopes:
    $<span id="cash">
<?php
  print $cash;
?>
    </span></h2>
   <form method="GET" action="getUpdatedBoardSales.php">
    <input value="Show Me the Money" type="submit" />
   </form>
  </div>
 </body>
</html>
```

Большой объем кода HTML, необходимый сценарию PHP.

Сценарий PHP выводит информацию, прочитанную из базы данных, вместе с HTML-кодом отчета

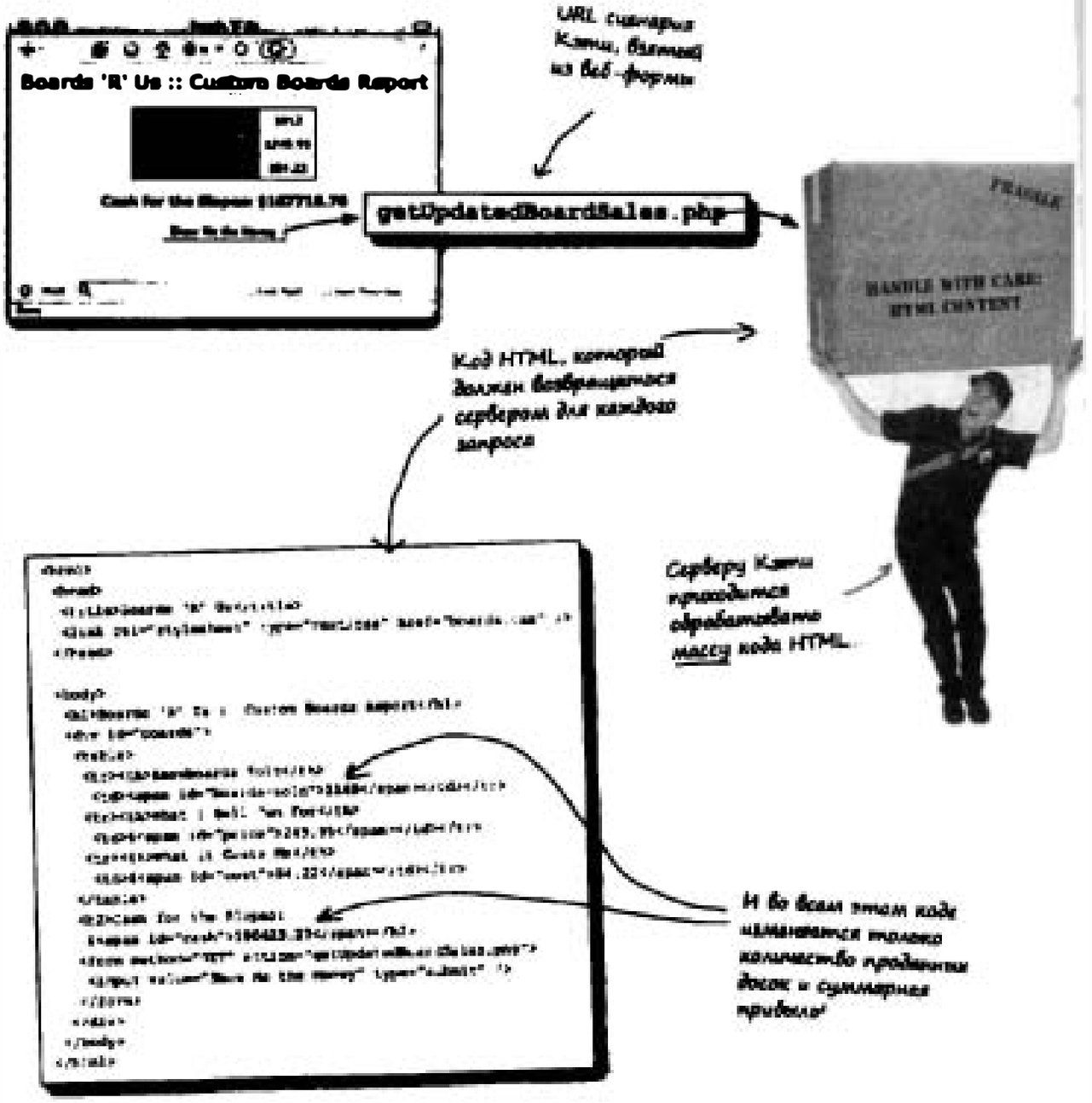
Что происходит?

Фактически этот код всего лишь определяет, сколько досок было продано, вычисляет сумму, заработанную Кэти, и возвращает форму HTML с обновленными данными (количеством проданных досок и прибылью).

Даже если вы не знаете PHP или никогда не работали с базами данных — ничего страшного. Через несколько страниц я приведу версию сценария, работающего без базы данных, чтобы вы могли запустить веб-отчет Кэти на своем компьютере.

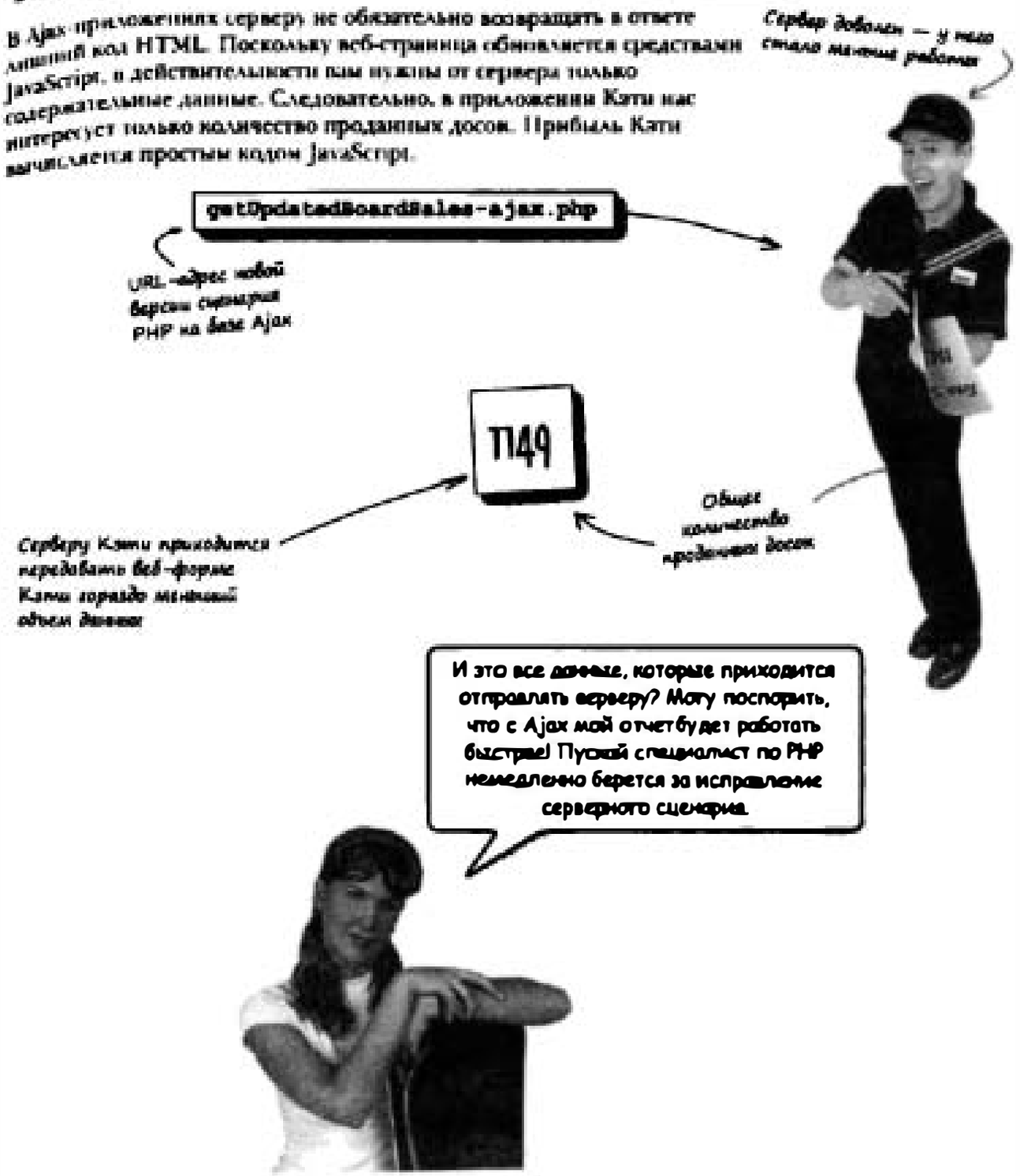
Что делал сервер ранее...

Помните, как работает старая версия отчета Кэти (без Ajax)? Каждый раз, когда к сценарию PHP поступает запрос, сценарий возвращает количество проданных досок вместе с полным кодом страницы HTML. Вернитесь на пару страниц назад, и вы увидите, что сценарий PHP более чем наполовину состоит из HTML! Взгляните на происходящее еще раз:



Что должен делать сервер сейчас

В Ajax-приложениях серверу не обязательно возвращать в ответе лишний код HTML. Поскольку веб-страница обновляется средствами JavaScript, в действительности вам нужны от сервера только содержательные данные. Следовательно, в приложении Кэти нас интересует только количество проданных досок. Прибыль Кэти вычисляется простым кодом JavaScript.



PHP ...со второго взгляда

Сценарий PHP значительно упрощается, когда вся его работа сводится к возврату количества проданных досок. Убедитесь сами:

Первая часть сценария почти не изменилась по сравнению с предыдущей версией

```
<?php

// Connect to database
$conn = @mysql_connect("mysql.headfirstlabs.com",
                       "secret", "really-secret");
if (!$conn)
  die("Error connecting to MySQL: " . mysql_error());

if (!mysql_select_db("headfirst", $conn))
  die("Error selecting Head First database: " . mysql_error());

$select = 'SELECT boardsSold';
$from   = '  FROM boardsrus';
$queryResult = @mysql_query($select . $from);
if (!$queryResult)
  die('Error retrieving total boards sold from database.');

while ($row = mysql_fetch_array($queryResult)) {
  $totalSold = $row['boardsSold'];
}

echo $totalSold;

mysql_close($conn);

?>
```

Вместо кода HTML новый сценарий возвращает всего одно число: количество проданных досок

* Обратите внимание: код, вычисляющий прибыль Кэти, исчез. Эти вычисления будут выполняться в нашем коде JavaScript.

Просто сделайте это

Пора обновить сценарий PHP и перевести его на Ajax. Откройте архив примеров и найдите файл getUpdatedBoardSales-ajax.php в каталоге chapter01/boards. Он представляет собой разновидность getUpdatedBoardSales.php, которая возвращает только количество проданных досок без кода HTML. Кроме того, сценарий работает без базы данных. Проследите за тем, чтобы файл находился в одном каталоге с файлом boards.html; сейчас мы воспользуемся новой версией этого сценария.

URL нового сценария

Итак, у нас появился сценарий PHP, который возвращает только количество проданных досок вместо полного кода HTML; остается лишь обратиться к сценарию с запросом.

Сохраним URL нового сценария PHP в переменной JavaScript:

Инициализация подключения

У нас есть объект запроса и переменная с URL-адресом для подключения. Вот как это делается:

```
function getBoardsSold() {
  createRequest();
  var url = "getUpdatedBoardSales-ajax.php";
  [                                        ]
}
```

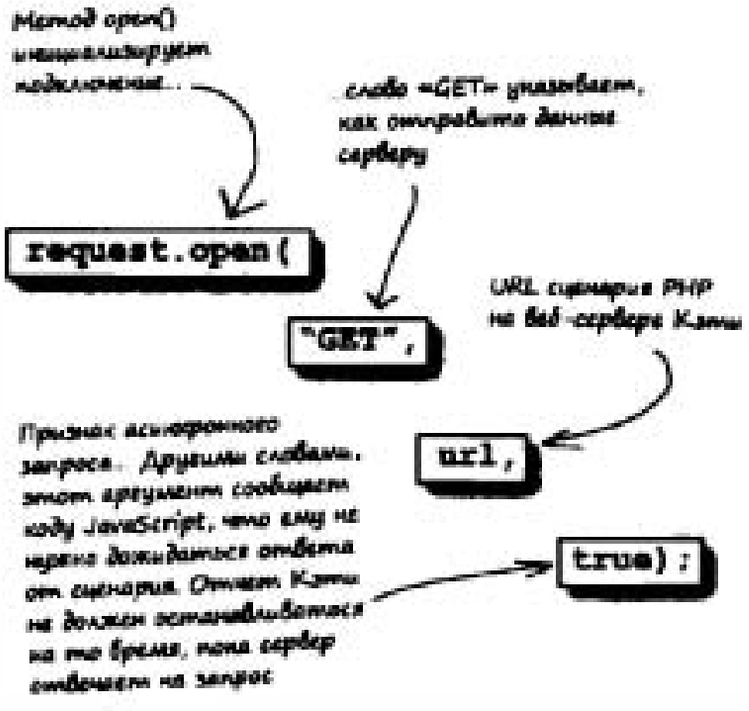
Эта строка инициализирует подключение и сообщает объекту запроса, как подключиться к серверу

Здесь мы начинаем использовать объект запроса, созданный функцией createRequest()

Довольно сложный материал. Постарайтесь себя «натаскать», если вам удастся ответить хотя бы на один вопрос самостоятельно

НЕ

Немного подробнее...



ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

В: Я полагал, что для выдачи запросов лучше использовать метод POST?

О: Запросы POST обычно используются при отправке серверу больших объемов данных, при передаче форм с коммерческой информацией и при размещении заказов. Поскольку мы не отправляем данные сценарию PHP и не оформляем заказы, в данном случае проще воспользоваться запросом GET; запросы POST будут рассмотрены ниже.

В: Придется ли мне когда-либо использовать open() для создания синхронных запросов и задавать последний аргумент равным «false»?

О: Несомненно. Иногда бывает нежелательно, чтобы пользователь продолжал работу над страницей, пока сервер отвечает на запрос. Например, если пользователь размещает заказ, обычно все его дальнейшие действия приостанавливаются до момента подтверждения заказа

Идеальная пара

Фрагменты кода связываются нерушимыми узами с комментариями. Этот союз обеспечивает любовь, дружеское общение и самодокументирование кода, что в конечном счете экономит программисту время и усилия. Ниже приведено несколько строк кода JavaScript, относящегося к Ajax. Свяжите каждую строку кода с комментарием, описывающим ее назначение.

Код

```
request.open("POST", url, true);
```

```
var request = new XMLHttpRequest();
```

```
var url = "/servlet/GetWikiMessageServlet";
```

```
request.open("GET", url, false);
```

Комментарии

```
/* Создание нового объекта для HTTP-взаимодействия с веб-сервером */
```

```
/* Создание новой переменной и присваивание ей URL сервлета Java */
```

```
/* Инициализация синхронного подключения методом GET */
```

```
/* Инициализация асинхронного подключения методом POST */
```

Помните наш контрольный список для getBoardsSold()?

Мы довольно далеко продвинулись в работе над кодом. Почему бы не вычеркнуть из списка то, что уже сделано?

- [ ] Создать новый объект запроса вызовом функции createRequest().
- [ ] Определить, к какому URL-адресу необходимо подключиться для получения обновленной информации о количестве проданных досок.
- [ ] Настроить объект запроса для установления
- [ ] Запросить обновленное количество проданных

Идеальная пара

Вам удалось составить счастливые пары? Проверьте ответы. Если вы где-то ошиблись, не жалейте времени и непременно разберитесь, что вы упустили и почему.

Встречаем счастливые пары...

Не работает в Internet Explorer

```
/* Создание нового объекта для HTTP-взаимодействия с веб-сервером */
var request = new XMLHttpRequest();

/* Создание новой переменной и присваивание ей URL сервлета Java */
var url = "/servlet/GetMileageServlet";

/* Инициализация синхронного подключения методом GET */
request.open("GET", url, false);

/* Инициализация асинхронного подключения методом POST */
request.open("POST", url, true);
```

Простая переменная JavaScript

Запросы отправляются методом GET или POST

Если аргумент равен false, запрос будет синхронным...

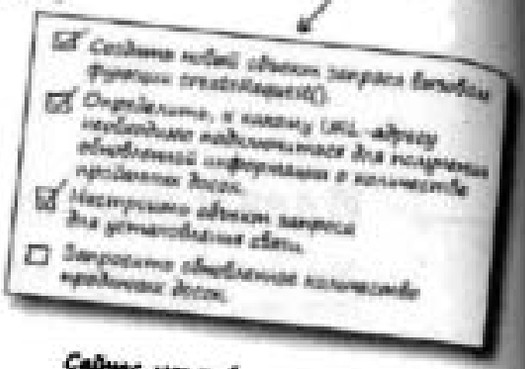
...а если true — запрос асинхронный

Подключение к веб-серверу

Отлично! У нас есть объект запроса, URL, подключение инициализировано и готово к использованию. Остается лишь создать подключение и запросить обновленное количество проданных досок. Для этого достаточно вызвать для объекта **request** функцию **send()**:

```
function getBoardsSold() {
  createRequest();
  var url = "getUpdatedBoardSales-ajax.php";
  request.open("GET", url, true);
  [redacted]
}
```

А вот наш список с 2.0 с. Остался всего один пункт



Сейчас мы работаем над последним пунктом

Здесь мы отправляем запрос...

...а это означает, что запрос не содержит данных. Сценарию никакие данные не нужны, он просто возвращает количество проданных досок

Вы шутите? Мы проделали столько работы, а теперь выясняется, что я отправляю серверу null? И это называется программирование нового поколения?

Слишком поверхностно голову (или страницу) и не упустить этот важный момент

Серверу не нужны никакие данные.

В этом конкретном приложении вам не обязательно предоставлять серверу какую-либо информацию. В каждом запросе к сценарию PHP нас интересует единственное число — общее количество досок, проданных Кэти. Следовательно, в запросе нам не нужно передавать серверу ничего, кроме **null**.

Помните: null означает неопределенную величину, а проще говоря, «ничего» — именно это мы и хотим передать серверу

Для выдачи запросов в асинхронных приложениях используются объекты JavaScript, а не отправка форм.

Подводим итоги

Нам еще предстоит довольно много сделать, но давайте ненадолго отвлечемся и посмотрим, что уже сделано. Возьмите чашку кофе, расслабьтесь и просмотрите описания двух шагов по превращению отчета Кэти в Ajax-приложение.

Задача, которую нужно было решить

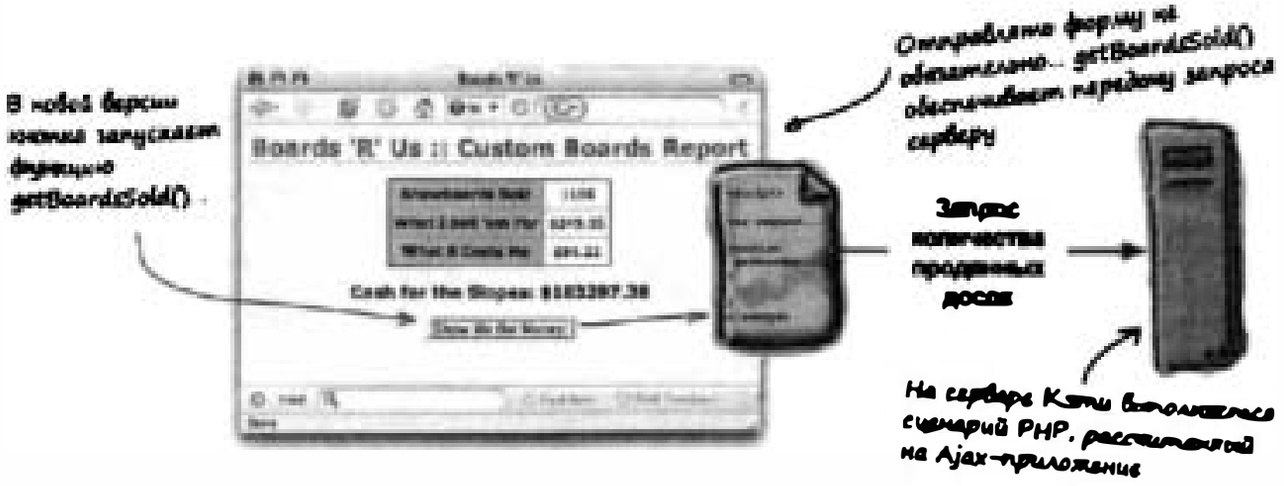
1 Создание объекта для отправки запроса серверу

...а вот что мы для этого сделали

Мы добавили новую функцию с именем **createRequest()**, которая создает объект запроса. Код даже работает в разных браузерах.

2 Написание функции JavaScript для запроса обновленного количества проданных досок

Мы написали новую функцию с именем **getBoardsSold()**. Эта функция использует **createRequest()** для создания нового объекта запроса. Затем мы создали переменную для хранения ссылки на Ajax-версию сценария PHP на сервере Кэти, инициализировали подключение и отправили запрос серверу.



Постойте... Но ведь кнопка по-прежнему отправляет форму старой версии сценария PHP? И как запускается функция getBoardsSold()? Кажется, мы еще не закончили с шагом 2...

Кое-что еще не сделано

Мы должны позаботиться о том, чтобы веб-форма Кэти не отправлялась обновленной версии сценария PHP — иначе дело кончится новыми перезагрузками страницы, а наша нелегкая работа пропадет даром!

Но это еще не все: необходимо обеспечить вызов нашей функции `getBoardsSold()` из веб-формы. Похоже, мы еще решительно не готовы к обновлению страницы... но что делать дальше?

Осталось решить две крупные задачи. Напишите внизу, что, по вашему мнению, необходимо сделать для завершения шага 2.

1.

2.

3.

СТОП! Не переходите к следующей странице, пока что-нибудь не напишете здесь.

Возвращаемся к HTML

Видите, что нужно сделать? Когда Кати щелкает на кнопке, веб-форма отправляет сценарию PHP всю форму. Но делать этого не нужно, потому что мы используем Ajax для обработки запросов к серверу.

Вернемся к коду HTML отчета:

```
<body>
 <h1>Boards 'R' Us :: Customer Boards Report</h1>
 <div id="boards">
  <table>
   <tr><th>Snowboards Sold</th>
    <td><span id="boards-sold">1012</span></td></tr>
   <tr><th>What I Sell 'em For</th>
    <td>$<span id="price">249.95</span></td></tr>
   <tr><th>What it Costs Me</th>
    <td>$<span id="cost">84.22</span></td></tr>
  </table>
  <h2>Cash for the Slopes:
   $<span id="cash">167718.76</span></h2>
  <form method="GET" action="getUpdatedBoardSales.php">
   <input value="Show Me the Money" type="submit" />
  </form>
 </div>
</body>
```

Код HTML из тега <body>... Остаток кода не приводится

В новой версии кнопка не должна отправлять форму...

```
<input value="Show Me the Money" type="button" />
```

...поэтому ее можно заменить обычной кнопкой ввода

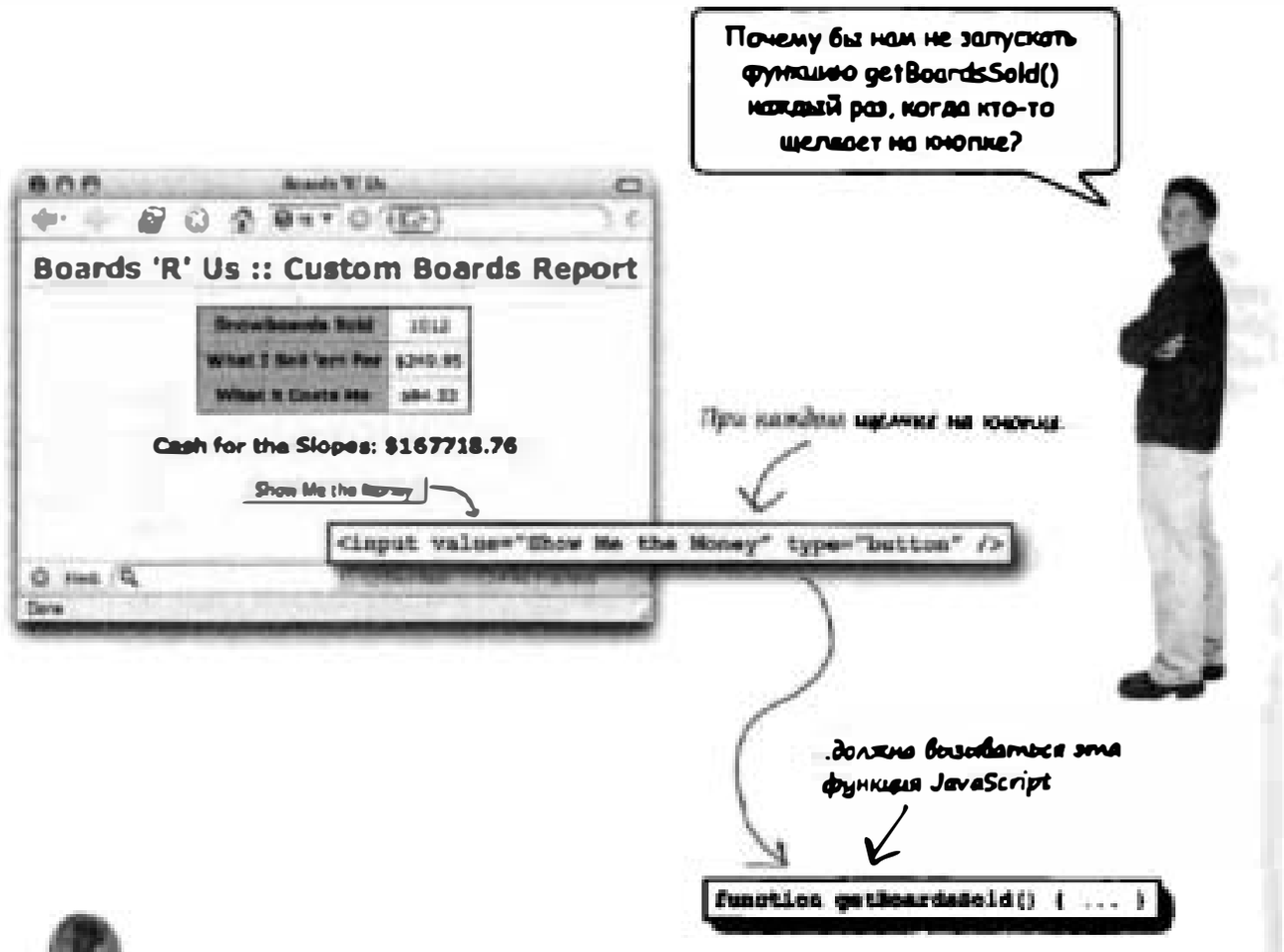
Часто задаваемые вопросы

В: А нельзя вместо изменения типа кнопки просто исключить атрибут action из тега <form>?

О: Хорошая мысль... В дополнение к смене типа кнопки. Если ограничиться простым удалением action, кнопка все равно попытается отправить форму, но при отсутствии action в теге <form> дело кончится простой перезагрузкой формы. Так что обязательно смените тип кнопки submit на button.

Запуск getBoardsSold() из веб-формы

Код JavaScript завершен, а кнопка уже не пытается отправить форму... но и функция **getBoardsSold()** пока что не запускается. Впрочем, проблема решается легко:



Вопрос для JavaScript

JavaScript, ты хотел запустить функцию из кода HTML. Можешь помочь нам в этом?

«Конечно бы — ведь я невероятно гибок в том, что касается вызова функций из веб-страниц. Просто включите в HTML один из моих обработчиков событий. Обработчик **onBlur()** используется при выходе из поля, **onClick()** — при щелчках... как насчет **onChange()** при изменении значения... а еще есть **onFocus()**... постойте, у меня их еще много... что значит — некогда? »

Добавление обработчика события

Каждый раз, когда Кэти щелкает на кнопке, должна запускаться функция `getBoardsSold()`. Для решения этой задачи можно воспользоваться обработчиком события JavaScript. Обработчик события связывает фрагмент кода JavaScript (такой, как функция **`getBoardsSold()`**) с некоторым событием — в нашем примере со щелчком на кнопке, находящейся на веб-странице.

Давайте воспользуемся обработчиком события, чтобы связать кнопку Show Me the Money с функцией **`getBoardsSold()`**. Функция связывается со щелчком на кнопке, поэтому выбираем обработчик **`onClick`**:

```
<form method="GET" action="getUpdatedBoardSales.php">
  <input value="Show Me the Money" type="button"
                                                  />
</form>
```

Так как в Ajax-версии отсюда эта форма никогда не выступает в роли подчиненной, вы можете смело удалить атрибут «action»

«onClick» означает: каждый раз, когда Кэти щелкает на кнопке...

...должна запускаться эта функция

Вспоминаем основные моменты

Вот почему мы исправили кнопку «Show Me the Money» так, чтобы она вызывала функцию JavaScript вместо отправки веб-формы Кэти

Еще не забыли нашу сводку основных моментов? Успевали следить за тем, что было сделано, в что еще предстоит сделать? Давайте вкратце разберемся, что мы узнали в этой главе:

Объекты запроса создаются функцией `createRequest()`

Для отправки запросов в асинхронных приложениях используются объекты JavaScript, а не отправка формы.

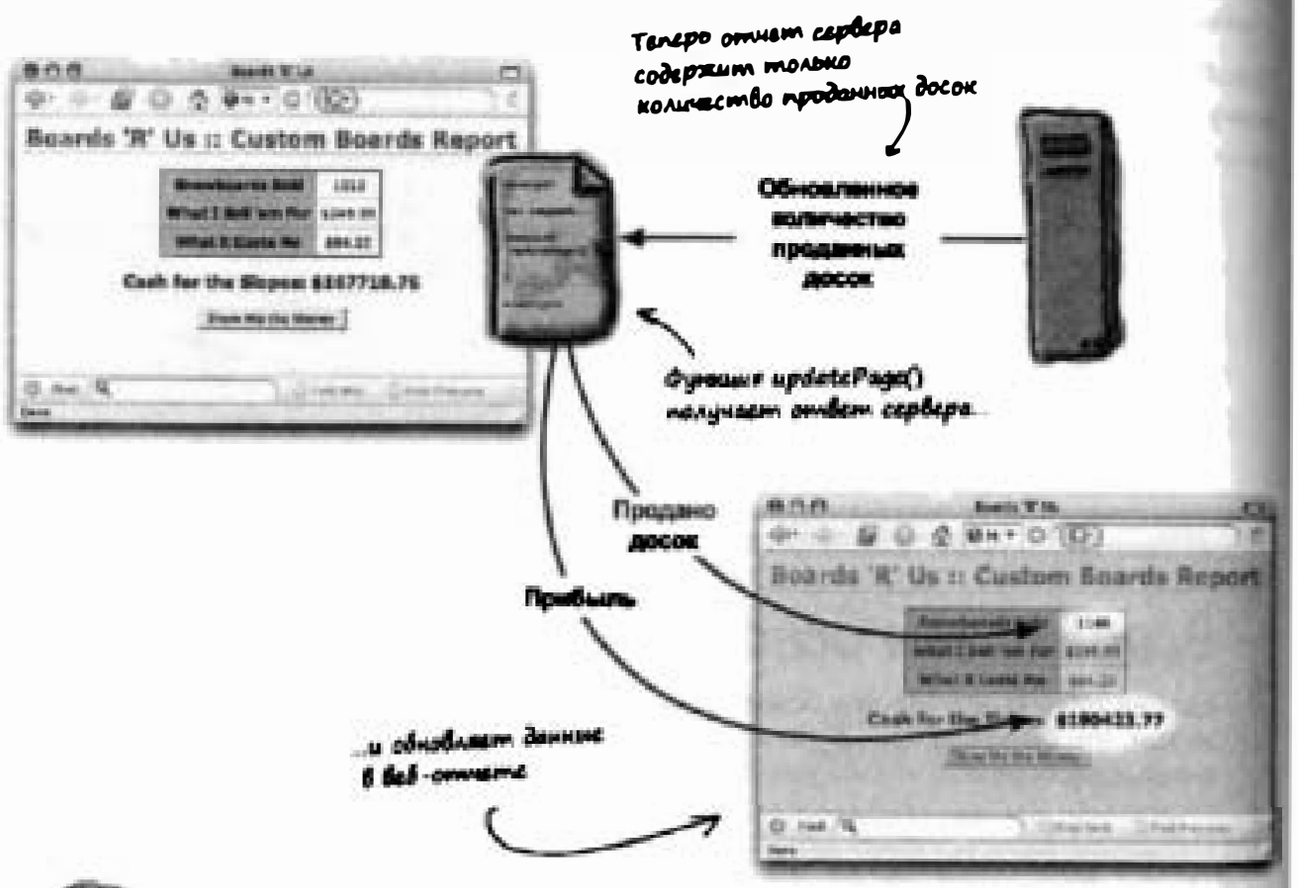
Ах эти мои комментарии! Мы еще не добрались...

Запросы и ответы обрабатываются браузером, а не кодом JavaScript.

Получив ответ на асинхронный запрос, браузер передает управление функции обратного вызова.

Шаг 3: Кодирование updatePage()

Итак, взаимодействие с сервером мы обеспечили; теперь можно принять от сервера ответ и обновить отчет новыми числами. Пора писать функцию `updatePage()`. Помните, что должна делать эта функция?



ШЕВЕЛИМ МОЗГАМИ

Как функция updatePage() получает ответ от сценария PHP? Помните, запрос к сценарию выдала функция getBoardsSold()... в наше приложение должно быть встроено *асинхронное*. Это означает, что Кэти нельзя заставлять ждать, пока сервер вернет новые данные.

Так как же функция updatePage() получает ответ с сервера? Подумайте пару минут, а затем переверните страницу.

Наверное, функцию updatePage() нужно вызвать в конце getBoardsSold(). Тогда updatePage() и получит ответ от сервера, верно?

```
function getBoardsSold() {
  createRequest();
  var url =
    "getUpdatedBoardSales-ajax.php";
  request.open("GET", url, true);
  request.send(null);
  updatePage();
}
```

Не выйдет... Функция send() не ждет ответа от сервера, поэтому updatePage() получит управление до того, как сервер завершит обработку запроса

Ajax — асинхронный JavaScript

Вспомните, что функция **getBoardsSold()** выдает асинхронный запрос. А это означает, что JavaScript не ожидает получения ответа от сервера.

Скорее всего, функция **getBoardsSold()** завершит работу еще до того, как сценарий **getUpdatedBoardSales-ajax.php** получит возможность обработать ваш запрос.

А если getBoardsSold() отработает до получения ответа от сервера, куда попадет ответ? И как организовать запуск updatePage(), когда сервер завершит обработку запроса?

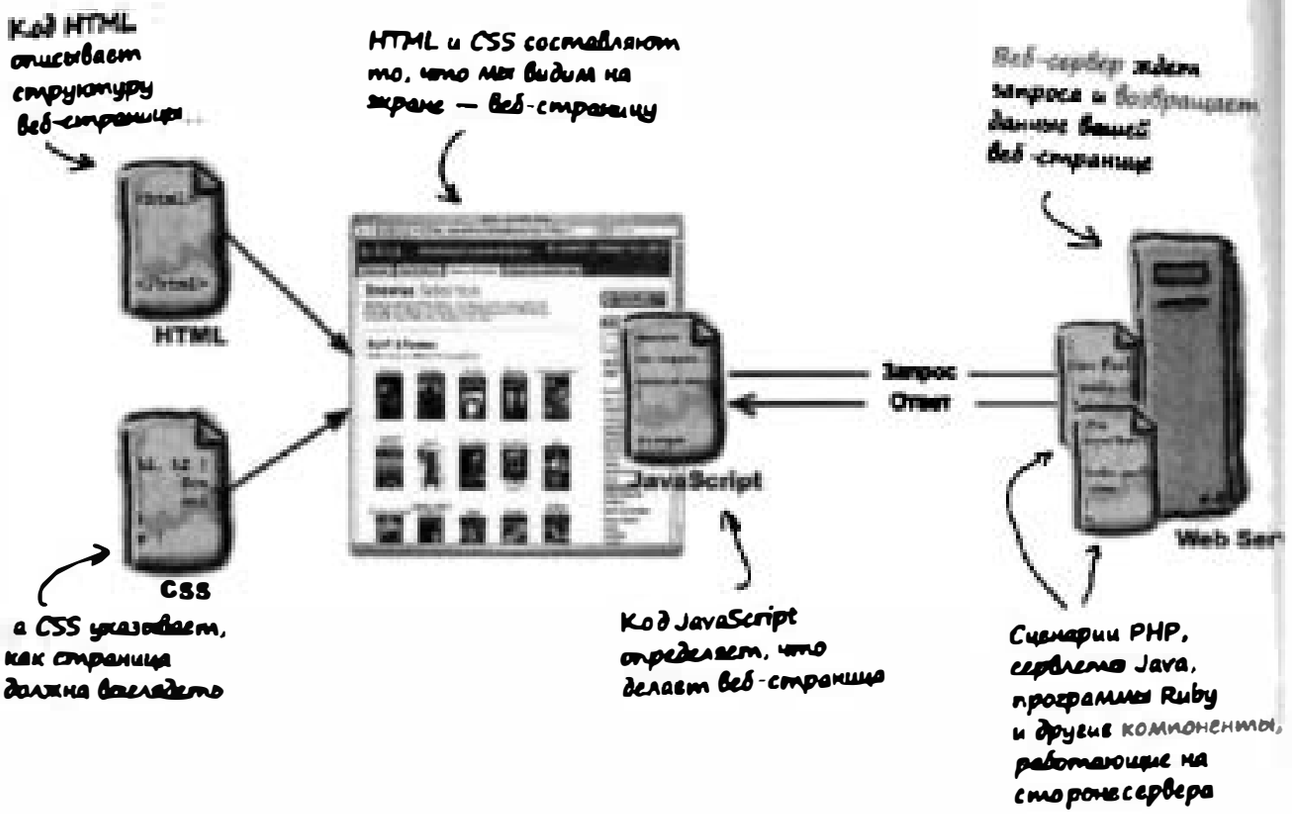
Куда попадет ответ?

Мы столкнулись с самым сложным аспектом асинхронного веб-программирования: если асинхронный код не ожидает ответа с сервера, как *использовать* этот ответ? И вообще, куда попадет этот ответ?

Вы уже знаете, что веб-страницы могут вызывать функции JavaScript из обработчиков событий. И видели, как асинхронный JavaScript выдает запросы серверу. Но чтобы картина асинхронности стала полной, не хватает еще одной важной детали...

Как мы представляем веб-приложения...

Чтобы разобраться в происходящем, необходимо сделать шаг назад и представить себе схему веб-приложения в целом. Как мы обычно представляем веб-приложение? Страница HTML, немного кода JavaScript, немного CSS для определения стиля... еще веб-сервер со сценариями или сервлетом. Фрагменты складываются в следующую картину:



Но это еще не все...

КТО Я?

Настало время проверить ваши способности к аналитическому мышлению. В пьесе «Аjax» имеется один важный персонаж. Мы еще с ним не встречались, но к торжественному появлению все готово. Посмотрим, удастся ли вам переиграть нового персонажа, заранее вычислив его личность до того, как вы перевернете страницу. Для упрощения вашей задачи мы приведем несколько высказываний таинственного персонажа.

➡ «Лелик? Многие этого не сознают, но я получил классическое образование. И еще я люблю Вагнера».

➡ «Приведите ко мне усталые, бедные, стесненные угловые скобки, желающие получить свой стиль...»

— Да, наш загадочный персонаж иногда склонен к излишней театральности

➡ «Кажется, я люблю водный спорт. Более того, я был в массовке в фильме «На гребне волны» и чуть не получил главную роль в «Голубой волне»».

➡ «Я по-настоящему крут... За последние годы я даже поучаствовал в нескольких войнах».

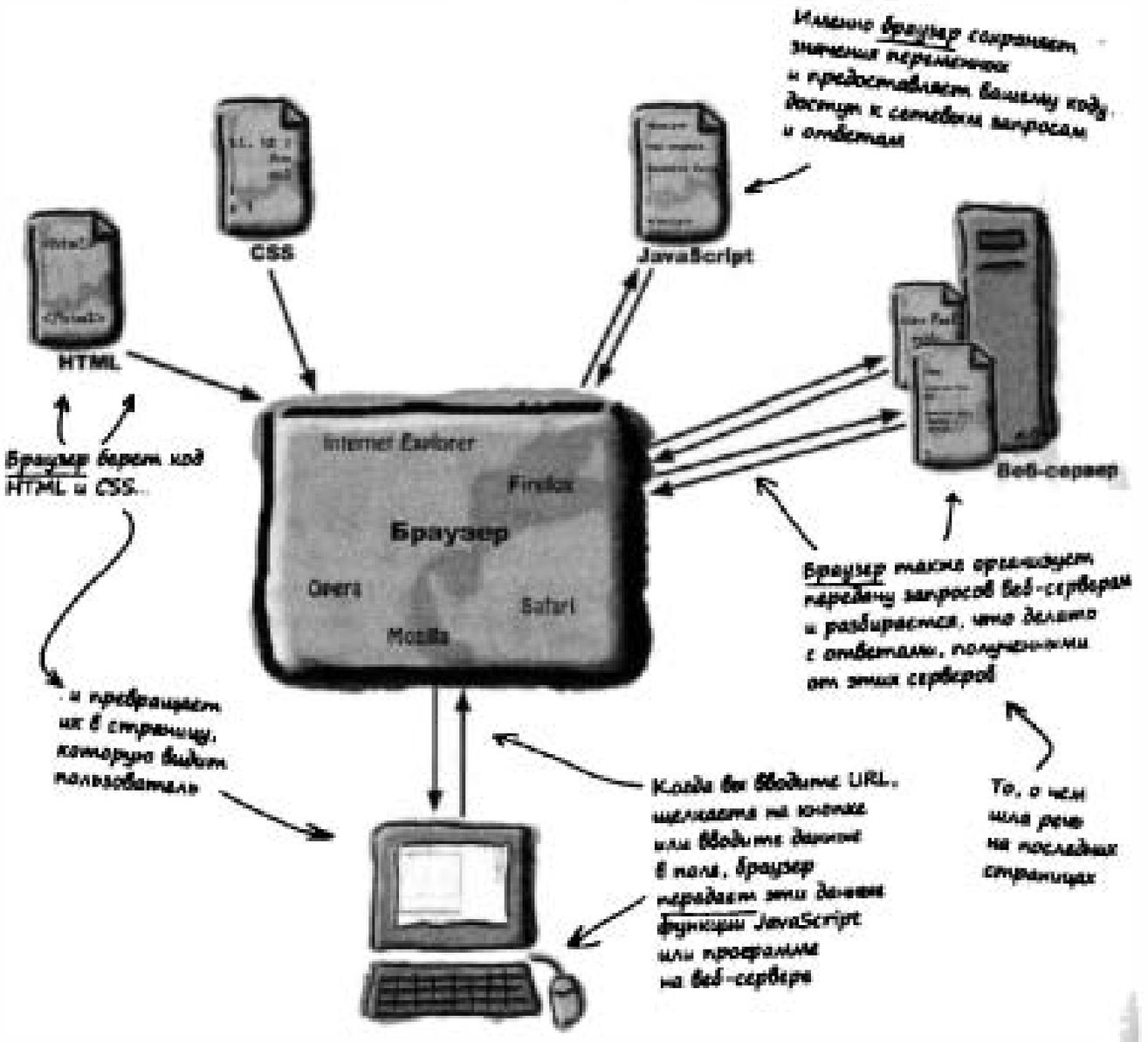
КТО Я? _____

Напишите, кем, по вашему мнению, является таинственный персонаж, а потом переверните страницу и проверьте свою догадку

На сцену выходит браузер

Кто-то должен связать все компоненты воедино, и эту задачу берет на себя браузер. Он берет HTML, CSS, и преобразует угловые скобки и точки с запятой в страницу с графикой, кнопками и текстом.

Кроме того, именно браузер выполняет код JavaScript... незаметно для вас он производит такие важные операции, как сохранение значений переменных, создание новых типов и обработка сетевых запросов, которые могли быть выданы вашим кодом.



Браузер просто помогает

Браузер таких сложных вещей не делает. Когда возникает необходимость в отправке запроса из JavaScript, вы пишете код следующего вида:

```
function getBoardsSold() {
  createRequest();
  var url = "getUpdatedBoardSales-ajax.php";
  request.open("GET", url, true);
  request.send(null);
}
```

Браузер всего лишь организует низкоуровневые сетевые операции, обеспечивающие работу этого кода. Работа сетевых подключений зависит от операционной системы (Linux, Windows, Mac OS X и т. д.), поэтому браузер берет на себя специфику конкретных систем. Благодаря этому код JavaScript работает в любой системе — а браузер обеспечивает преобразование кода в форму, понятную для данного компьютера.

Вернитесь к с. 40 и обратите внимание на то, что именно *браузер* занимается отправкой запросов и получением ответов от сервера. Ваш код указывает браузеру, **что делать**, а браузер решает, **как делать**.

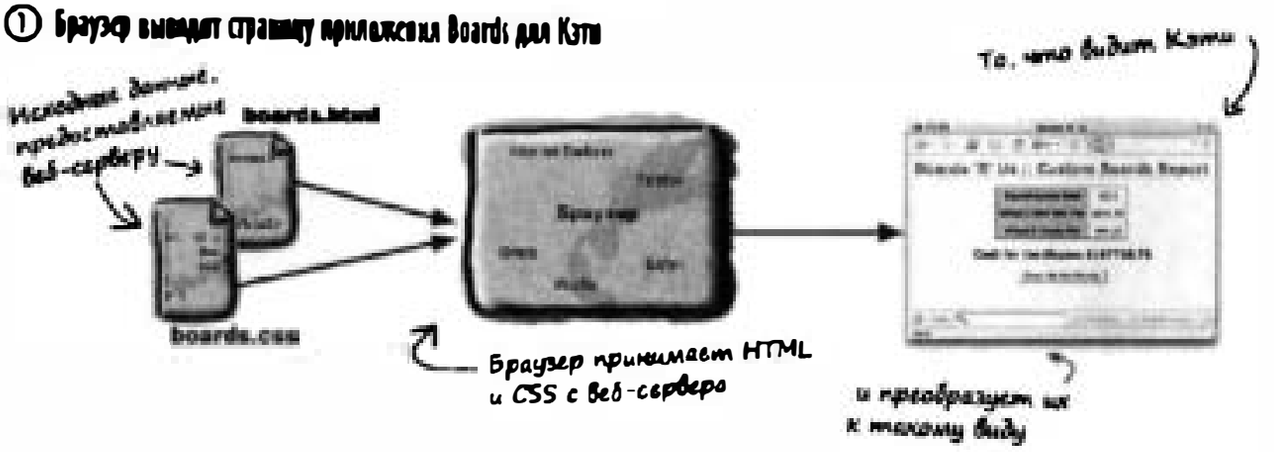
Запросы и ответы обрабатываются браузером, а не кодом JavaScript.

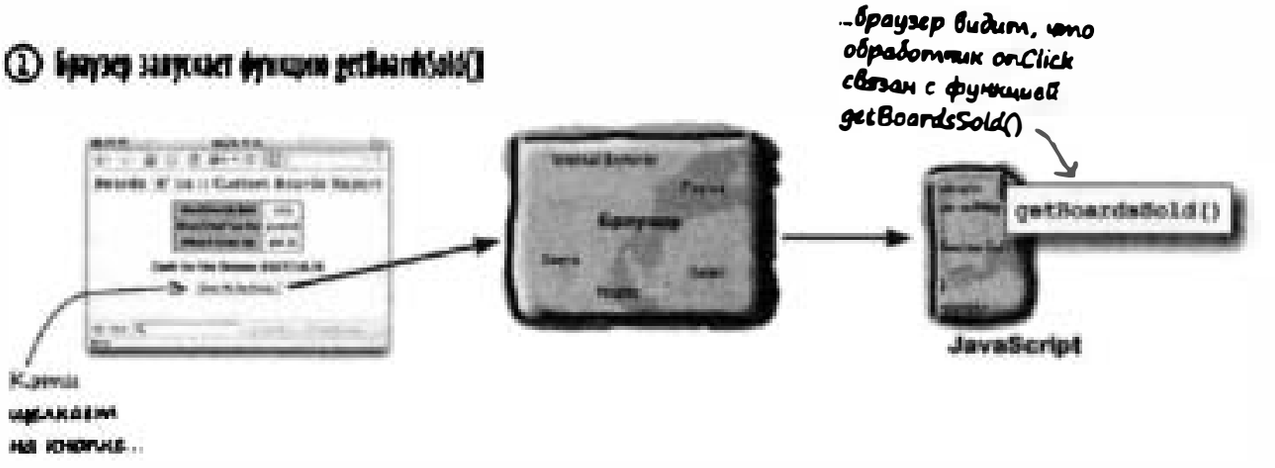
ШЕВЕЛИМ МОЗГАМИ

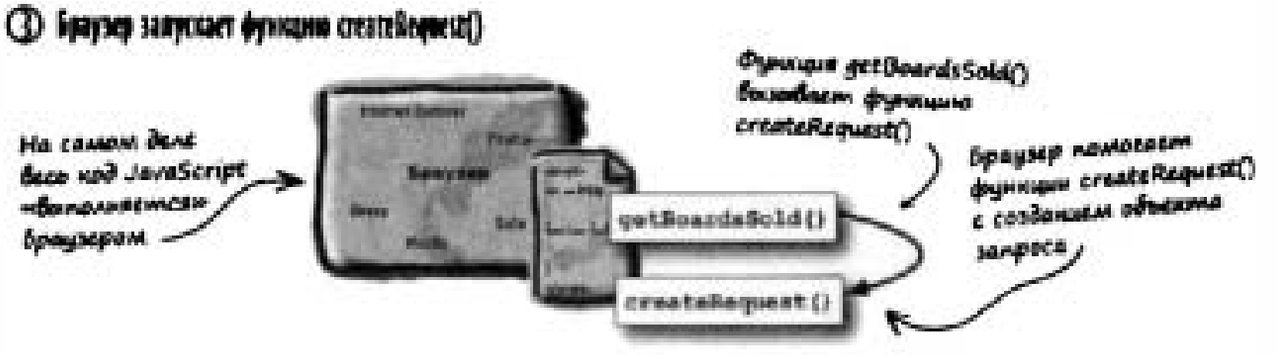
Как вы думаете, почему так важно, что обработка запросов в асинхронных приложениях осуществляется именно браузером? Напомним, что веб-сервер может ответить только тому, кто обратился к серверу с исходным запросом.

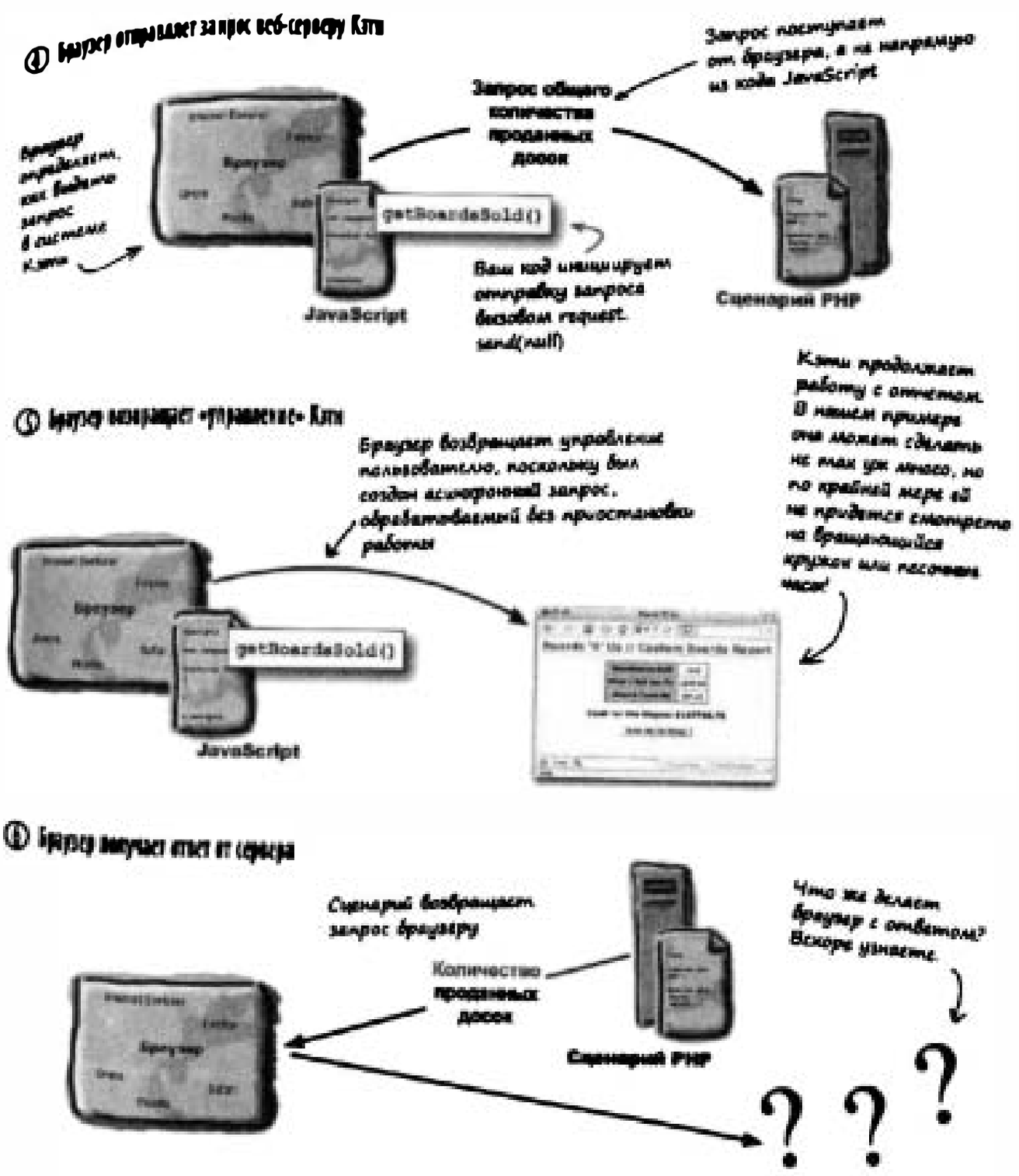
Браузер передает ответ сервера вашему коду JavaScript

Браузер занимается не только отправкой запросов, но и получением ответов с сервера. Давайте посмотрим, что делает браузер в приложении Boards:



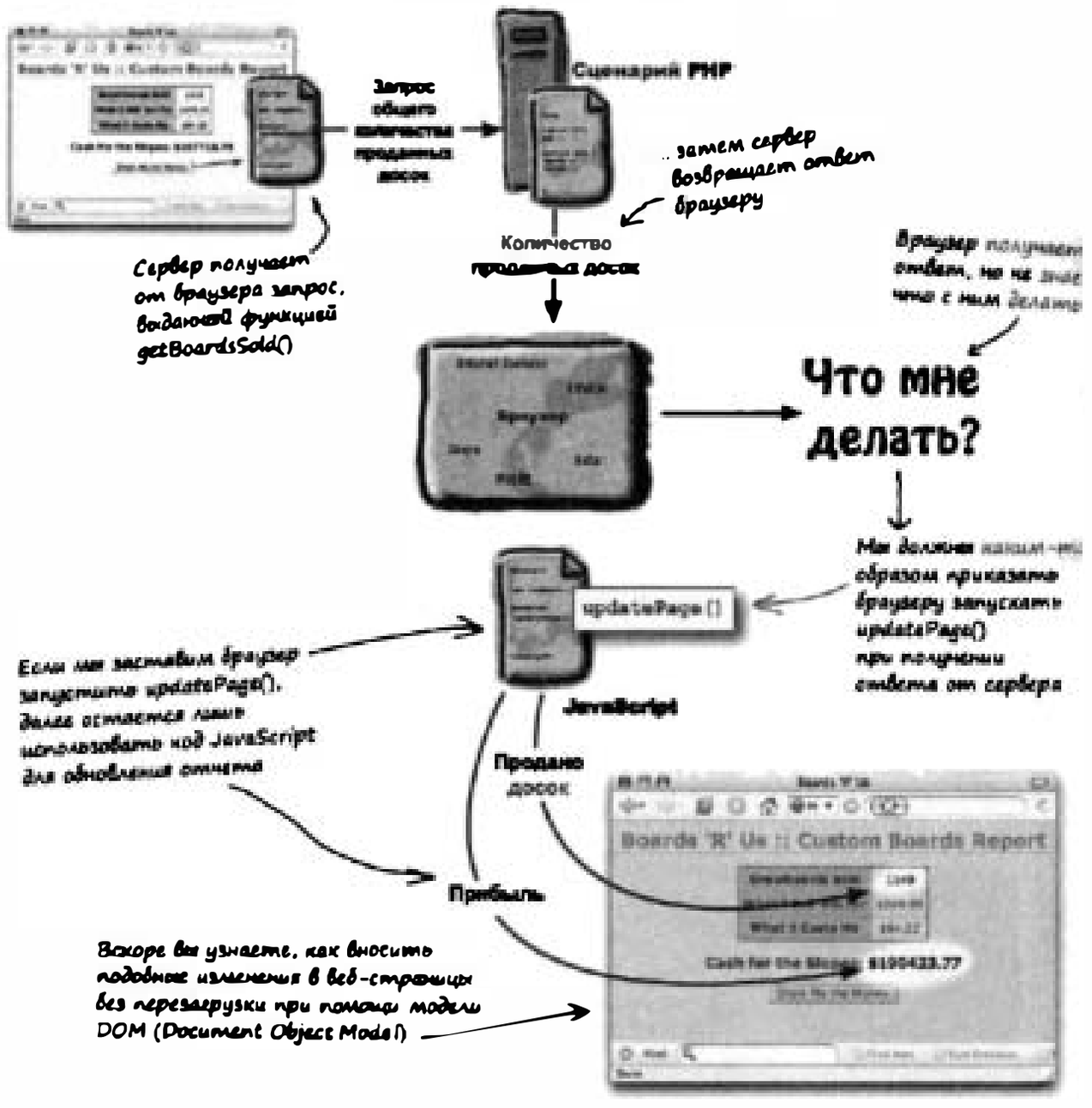





④ Браузер отправляет запрос веб-серверу Кэти
Запрос поступает от браузера, а не напрямую из кода JavaScript
Запрос общего количества проданных досок
Браузер управляет, как выбыло запрос в систему Кэти
getBoardsSold()
Ваш код инициирует отправку запроса вызовом теперь. send(null)
JavaScript
Сценарий PHP
⑤ Браузер возвращает «управление» Кэти
Браузер возвращает управление пользователю, поскольку был создан асинхронный запрос, обрабатываемый без приостановки работы
Кэти продолжает работу с отчетом. В нашем примере она может сделать не так уж много, но по крайней мере ей не придется смотреть на вращающийся кружок или песочные часы!
getBoardsSold()
JavaScript
⑥ Браузер получает ответ от сервера
Сценарий возвращает запрос браузеру
Количество проданных досок
Сценарий PHP
Что же делает браузер с ответом? Вскоре узнаете.
? ? ?

Что должен сделать браузер с ответом от сервера?

Браузер получает ответ от сервера, но ничего с ним не делает без вашего распоряжения. Следовательно, мы должны каким-то образом приказать браузеру запустить нашу функцию **updatePage()** при получении ответа.



Беседа за чашкой кофе

Сегодня у нас в гостях почетные персоны, любители кофе:

Браузер и страница HTML

Браузер: Привет, HTML. Приятно снова увидеться. Я только что общался с твоим знакомым, CSS.

Страница HTML: Неужели? Я в последнее время часто с ним вижусь. Похоже, многие пользователи осознают возможность применения CSS в своих страницах... Впрочем, мы оба этому только рады.

Браузер: Да, я тоже. Хотя из-за этого мне приходится выполнять дополнительную работу, потому что я должен взять два файла и объединить их.

Страница HTML: *Тебе?* О чем ты? Это мне пришлось распрощаться со своими старыми знакомыми — такими как align и font face. Не понимаю, при чем здесь ты.

Браузер: Шутишь? Ох уж эти языки разметки... Вечно не в тренде. И с XHTML то же самое... без меня вас вообще никто не увидит!

Страница HTML: Еще бы, ведь ты — программа, при помощи которой люди смотрят на нас. Послушай, ведь ты существуешь только по одной причине: для просмотра моей разметки!

Браузер: «Просматривать» HTML можно и в текстовом редакторе... но кому захочется смотреть на ваши угловые скобки и теги <font>. Людей хотят видеть то, что я создаю из вас — визуальные страницы с графикой, таблицами и ссылками.

Страница HTML: Как же... Развлекаться. Можно подумать, выстраивалась целая очередь из желающих увидеть, сколькими *разными* способами можно отобразить *одну и ту же* веб-страницу. Обожаю браузеры... Вот только почему они всегда меняют мой внешний вид?

Браузер: «Войны браузеров» подходят к концу, ты — большая неопределенность с кучей угловых скобок. Кроме того, если бы люди писали стандартные веб-страницы, то и проблем бы не было.

Страница HTML: Да неужели? А вот JavaScript мне говорит совсем другое. Ему очень не нравится, что для простого запроса к серверу приходится использовать два разных типа объектов.

Браузер: Послушай, два — гораздо лучше, чем пять или шесть... А когда выйдут новые версии Internet Explorer и Mozilla, ты и не заметишь, как мы перейдем на использование одного типа объектов запросов.

Страница HTML: Да ты уверен, что у тебя есть ответы на все вопросы?

Браузер: На все вопросы — не знаю, а на все запросы — точно. И это еще одна причина, по которой тебе без меня не обойтись.

Страница HTML: Ответы на все запросы? О чем ты?

Браузер: Может, тебе лучше спросить своего друга JavaScript. Все полагают, что запросы создаются веб-страницами и кодом...

Страница HTML: Конечно!

Браузер: ...Но на самом деле это я тайно стою за тем, чтобы страницы выглядели правильно, запросы отправлялись, а ответы принимались. Кем бы вы без меня были? Наборами странных символов в текстовом файле.

Страница HTML: <чернобровка>

Теперь, когда мы чуть больше знаем о браузерах, пора вернуться к getBoardsSold()

Передача инструкций браузеру

Не забудьте — мы должны сообщить браузеру, как поступить с ответом сервера на наш запрос, *до* отправки запроса... Потому что после отправки запроса функция `getBoardsSold()` завершается, и наш код JavaScript не будет знать, что ему делать с ответом. К счастью, у объекта запроса имеется свойство, предназначенное именно для этой цели:

```
function getBoardsSold() {
  createRequest();
  var url = "getUpdatedBoardSales-ajax.php";
  request.open("GET", url, true);
  [redacted]
  request.send(null);
}
```

Не забудьте задать это свойство до вызова send(), иначе функция не запустится

Если подставить сюда имя функции, браузер запустит ее при получении ответа с сервера

По правилам JavaScript круглых скобок после имени функции быть не должно

Часто задаваемые вопросы

В: Сервер общается с браузером, браузер общается с JavaScript, JavaScript обновляет страницу... Совсем запутался. Можно еще раз?

О: Помните, речь идет об асинхронных запросах. Когда ваш код приказывает браузеру отправить запрос серверу, сервер должен выдать ответ — однако ваш код не может дожидаться его получения. Из-за этого ответ, полученный от сервера, доставается браузеру, а последний решает, что делать дальше. Браузер передает управление функции, указанной в свойстве onreadystatechange объекта запроса.

В: Еще раз: что делает свойство onreadystatechange? Как все сложно...

О: На самом деле все проще, чем кажется. Данное свойство всего лишь указывает браузеру, что при изменении состояния готовности запроса (скажем, когда сценарий PHP ответит браузеру), браузер должен вызвать функцию JavaScript, назначенную в объекте запроса. В данном случае это функция updatePage.

В: Значит, существуют и другие обстоятельства, кроме завершения сценария, которые тоже могут привести к вызову updatePage()?

О: Да. Веб-запросы могут находиться в нескольких «состояниях готовности», и при каждой смене состояния будет вызываться функция updatePage(). В следующей главе данная тема рассматривается гораздо подробнее, а сейчас просто примите к сведению этот факт.

Получив ответ на асинхронный запрос, браузер передает управление функции обратного вызова.

Получение ответа сервера

Наконец-то можно перейти к программированию функции **updatePage()**... или еще нет? Мы задали свойство **onreadystatechange** объекта запроса, поэтому браузер будет запускать функцию **updatePage()** при каждом ответе на запрос. Но кое-чего еще не хватает...

Я понял, как заставить браузер выполнять updatePage() при ответе с сервера. А как насчет данных, которыми отвечает сервер? Как обратиться к ним из кода updatePage()?

Чтобы получить данные, переданные в ответе сервера, воспользуйтесь свойством responseText объекта запроса JavaScript.

Браузер снова приходит на помощь

Мы уже знаем, как приказать браузеру запускать функцию **updatePage()** при получении ответа сервера. Однако браузер (и объект запроса) способны сделать еще больше, чтобы только помочь вам.

Получив ответ от сервера, браузер определяет, что делать дальше, проверяя свойство **onreadystatechange** объекта. А поскольку нам понадобятся данные, возвращенные сервером, браузер помещает их в другое свойство объекта запроса: свойство с именем **responseText**.

Итак, когда потребуется узнать, что именно вернул сервер в своем ответе, достаточно обратиться к свойству **responseText** объекта запроса.

Браузер задает значения многих свойств объекта запроса до выполнения вашей функции JavaScript. Скоро мы расскажем, что делают эти свойства

Сделайте это...

Вы уже знаете два свойства объекта запроса, используемых в коде JavaScript. Пришло время проверить полученные знания. В левой колонке перечислены свойства объекта запроса, а в правой — описания этих свойств. Удастся ли вам связать имена свойств с их назначением?

responseText

readyState

onreadystatechange

status

Код состояния HTTP, полученный от сервера

Функция, которая должна вызываться браузером, когда сервер ответит на запрос

Числовой признак состояния запроса: загрузка, в процессе обработки, обработка завершена и т. д.

Данные, возвращаемые сервером в ответ на запрос

Некоторые из этих свойств нам еще не встречались. И все же попробуйте предположить смысл каждого из них

Функция updatePage()

Браузер сохранит ответ сервера в объекте запроса, после чего запускает **updatePage()**. Наконец-то можно перейти к программированию функции. Давайте посмотрим, что нужно сделать:

Первое: получить обновленное количество проданных досок

Сервер возвращает число браузеру, а браузер помещает его в свойство **responseText** объекта запроса:

```
function updatePage() {
  var newTotal = [redacted];
}
```

Свойство объекта запроса, в котором браузер передает данные ответа сервера

Объект запроса

Второе: определить элементы HTML, нуждающиеся в обновлении

В нашем отчете обновляются два элемента: количество проданных досок и прибыль, полученная Кэти. Для получения этих элементов по атрибутам id можно воспользоваться методом **getElementById()**.

```
function updatePage() {
  var newTotal = request.responseText;
  var boardsSoldEl = document.getElementById("boards-sold");
  var cashEl = document.getElementById("cash");
}
```

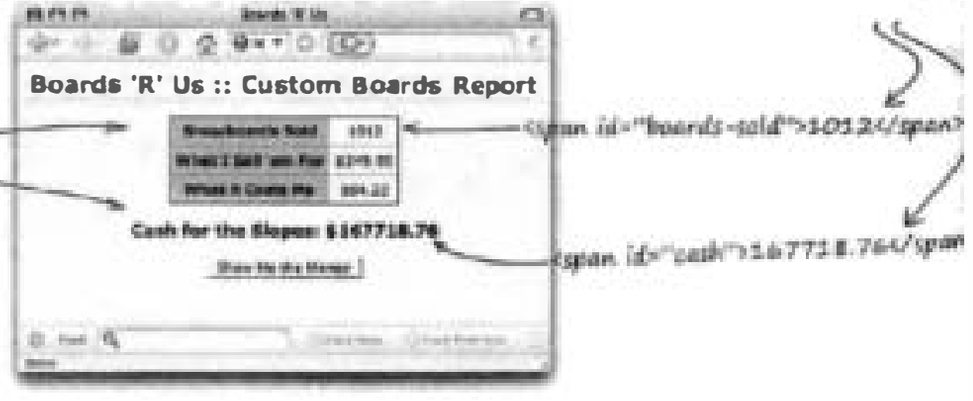
Каждый элемент сохраняется в переменной для последующих ссылок в программном коде

document представляет всю страницу HTML

getElementById() находит элемент с заданным идентификатором

Имена элементов HTML, которые мы хотим получить

Вспомните: Кэти добавила атрибуты id в элементы , чтобы определять их стили в CSS. Теперь атрибуты id позволят легко обратиться к каждому элементу из кода JavaScript



Third: Add a reference to Katie's text utilities

Все готово к обновлению количества проданных досок. Обновление текста в элементах `<span>` потребует нетривиального кода JavaScript. Мы рассмотрим эти методы в главе 4, когда речь пойдет о модели DOM, а пока Кэти разместила готовый код JavaScript в файле text-utils.js.

Вы можете найти файл text-utils.js в папке примеров этой главы chapter03/boards

Файл содержит удобные вспомогательные функции, однако сначала необходимо сообщить приложению Boards, где находится сам файл. Ссылка на файл включается в HTML при помощи тега `<script>`:

Все это находится в секции <head> страницы HTML

```
<head>
  <title>Boards 'R' Us</title>
  <link rel="stylesheet" type="text/css" href="boards.css" />
  [redacted]
  <script language="javascript" type="text/javascript">
```

Начало кода JavaScript, написанного нами в этой главе

После добавления этой строки вы сможете использовать служебные функции из файла text-utils.js в функции updatePage()

Четвертое: обновить отчет новым количеством проданных досок

Файл text-utils.js содержит функцию с именем `replaceText()`. Эта служебная функция обновляет количество проданных досок в отчете Кэти:

Все функции в text-utils.js используют модель DOM (Document Object Model) для оперативного обновления веб-страницы

Определение функции находится в text-utils.js. Мы рассмотрим эти функции намного позже

```
function updatePage() {
  var newTotal = request.responseText;
  var boardsSoldEl = document.getElementById("boards-sold");
  var cashEl = document.getElementById("cash");
  [redacted]
}
```

Элемент, текст которого требуется заменить

а это новое значение

При выполнении этого фрагмента в веб-странице обновляется количество проданных досок

Просто сделайте это

Для обновления функции **updatePage()** все готово. Ниже приведен код функции **updatePage()**, но в нем осталось несколько пропусков. Попробуйте определить, какой код должен находиться в каждом пропуске, и впишите свои ответы.
Когда все пропуски будут заполнены, сравните свои ответы с нашими на следующей странице.

```
function updatePage() {
  var newTotal = request.responseText;
  var boardsSoldEl = document.getElementById("boards-sold");
  var cashEl = document.getElementById("cash");
  replaceText(boardsSoldEl, newTotal);

  /* Figure out how much cash Katie has made */
  var priceEl = document.getElementById("_____");
  var price = getText(_____);
  var _____ = document._____("cost");
  var cost = getText(costEl);
  var cashPerBoard = _____ - _____;
  var cash = _____ * _____;

  /* Update the cash for the slopes on the form */
  cash = Math.round(cash * 100) / 100;
  replaceText(cashEl, _____);
}
```

Функция getText() определяется в text-utils.js. Она берет любой элемент и возвращает его текстовое содержимое

Благодаря этому маленькому трюку в прибыли остаются только две цифры после запятой, как в обычных денежных суммах

Просто сделайте это — решение

Для обновления функции **updatePage()** все готово. Ниже приведен код функции **updatePage()**, но в нем осталось несколько пропусков. Попробуйте определить, какой код должен находиться в каждом пропуске, и впишите свои ответы.

```
function updatePage() {
  var newTotal = request.responseText;
  var boardsSoldEl = document.getElementById("boards-sold");
  var cashEl = document.getElementById("cash");
  replaceText(boardsSoldEl, newTotal);

  /* Figure out how much cash Katie has made */
  var priceEl = document.getElementById("  price  ");
  var price = getText(  priceEl  );
  var  costEl  = document.  getElementById  ("cost");
  var cost = getText(costEl);
  var cashPerBoard =  price  -  cost  ;
  var cash =  cashPerBoard  *  newTotal  ;

  /* Update the cash for the slopes on the form */
  cash = Math.round(cash * 100) / 100;
  replaceText(cashEl,  cash  );
}
```

Функция getText() определяется в text-utils.js. Она берет любой элемент и возвращает его текстовое содержимое

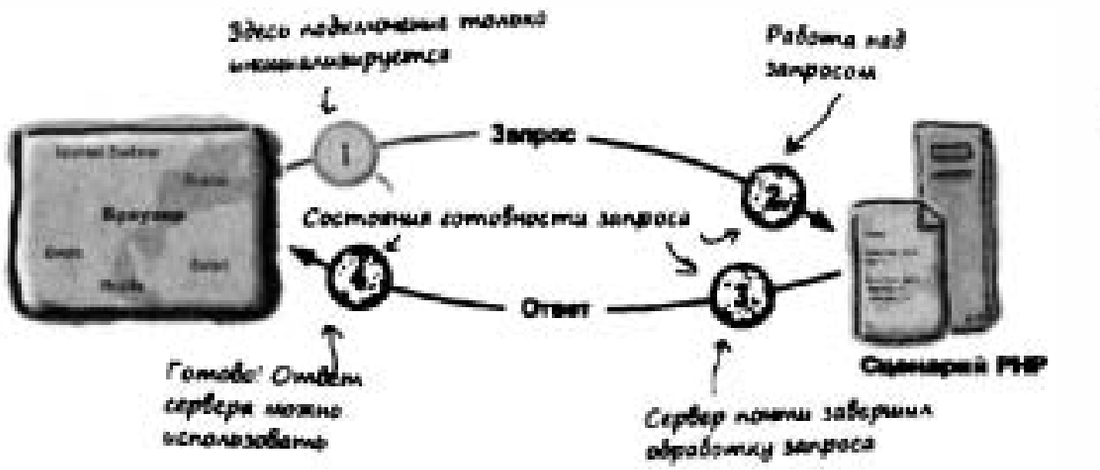
Благодаря этому маленькому трюку в привели остаются только две цифры после запятой, как в обычных денежных суммах

Q: Модель Document Object Model (сокращенно DOM) используется браузерами для интерпретации страниц HTML. Работая с DOM, можно динамически обновлять веб-страницу, как мы это делаем с отчетом Katie. Модель DOM подробно рассматривается в главе 4, а пока просто используйте функции вспомогательного файла text-utils.js и не беспокойтесь по поводу DOM.

Q: Вы ранее упоминали модель DOM. Что это такое? И какое отношение она имеет к этому Ajax?

Проверка завершения работы сервера

На данный момент функция **updatePage()** предполагает, что к моменту ее запуска сервер уже отработал... но это не так! Чтобы понять, что происходит с запросом, мы должны познакомиться с состояниями готовности. Объект запроса обладает признаком, по которому браузер может получить кое-какую информацию о текущем состоянии запроса.



Признак состояния готовности сообщает браузеру на какой стадии находится обработка запроса

Состояния запроса связаны со свойством onreadystatchange объекта запроса

Помните то свойство, при помощи которого мы в **getBoardsSold()** сообщали браузеру, что делать при получении ответа от сервера? Напомним еще раз, чтобы освежить память:

```
function getBoardsSold() {
  createRequest();
  var url = "getUpdatedBoardSales-ajax.php";
  request.open("GET", url, true);
  request.onreadystatechange = updatePage;
  request.send(null);
}
```

Это свойство определяет функцию, которая должна запускаться браузером каждый раз при смене состояния готовности запроса

Свойство действует во всех состояниях готовности, а не только в том, которое указывает на завершение обработки запроса сервером

Проверка состояния готовности

Мы знаем, что браузер запускает функцию updatePage() при получении ответа от сервера. Однако здесь имеется одна тонкость: браузер запускает updatePage() при каждом изменении состояния готовности, а не только тогда, когда сервер завершает отправку ответа. Еще раз взгляните на диаграмму на с. 54; обратите внимание на то, как по мере обработки запроса увеличивается числовой признак состояния готовности.

При завершении обработки запроса состояние готовности равно 4, поэтому мы можем сравнить текущее значение признака с этим числом в своем коде. Такая проверка гарантирует, что обновление отчета Кэти будет происходить только при получении ответа от сервера.

readyState — свойство объекта запроса, в котором хранится текущее состояние готовности

Если свойство readyState объекта запроса равно 4, значит, сервер завершил обработку запроса

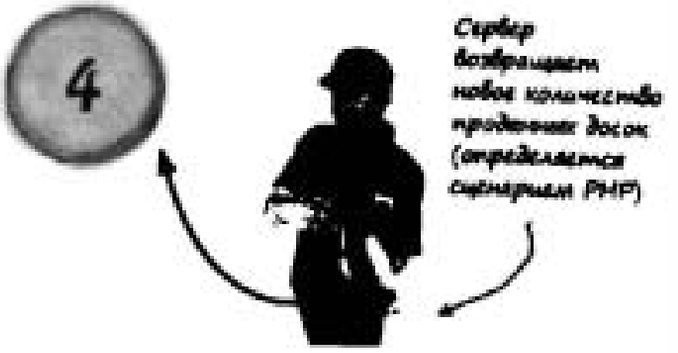
Эта функция будет запускаться при каждом изменении состояния

```
function updatePage() {
  if (request.readyState == 4) {
    var newTotal = request.responseText;
    var boardsSoldEl = document.getElementById("boards-sold");
    var cashEl = document.getElementById("cash");
    replaceText(boardsSoldEl, newTotal);

    /* Figure out how much cash Katie has made */
    ...

    /* Update the cash total on the web form */
    ...
  }
}
```

Весь этот код выполняется только тогда, когда состояние готовности запроса равно 4, то есть сервер завершил обработку запроса

Код, написанный нами несколько страниц назад

Не забудьте закрывающую скобку!



Если запрос обработан сервером, его состояние готовности будет равно 4. Это означает, что мы можем использовать данные, возвращенные сервером (и сохраненные браузером в объекте запроса)

Сервер возвращает новое количество продаж досок (определяется сценарием PHP)

Просто сделайте это

Откройте файл boards.html и добавьте весь новый код JavaScript. Убедитесь в том, что функция **updatePage()** содержит код JavaScript для проверки состояния готовности объекта запроса, а также код обновления продаж и вычисления прибыли. Копия файла text-utils.js должна находиться в одном каталоге с файлами boards.html и boards.css.

Часто задаваемые вопросы

В: Я не понимаю, почему в конце функции updatePage() на с. 52 мы сначала делим, а затем умножаем прибыль на 100. Разве получится не то же число, с которого мы начали?

О: У JavaScript есть некоторые странности с умножением чисел. Иногда в процессе вычислений появляется длинная цепочка цифр после запятой. Например, вместо 59,95 JavaScript может выдать 59,9499995. Разумеется, Кэти совсем не хотелось бы видеть в своем отчете нечто подобное.

Чтобы исправить число вида 59,9499995, сначала умножь его на 100 — получится 5994,99995. Далее функция **Math.round()** округляет результат до ближайшего целого, то есть 5994. Наконец, результат делится на 100, и мы получаем 59,94...именно то, что нужно Кэти.

В: Что делает функция getText()? Я встретил ее в функции updatePage() на с. 52.

О: **getText()** — еще одна вспомогательная функция, как и **replaceText()**. При вызове ей передается элемент веб-страницы; функция возвращает текст, содержащийся в заданном элементе. В отчете Кэти функция **getText()** берет цену, по которой Кэти продает свои доски, и собственность одной доски из элементов в коде HTML.

В: Как насчет вспомогательных функций из text-utils.js? Мне действительно можно не беспокоиться о них?

О: Если вы не забудете включить ссылку на файл text-utils.js в тег <script>, программа будет работать нормально. Функции, работающие с моделью DOM, будут подробно рассмотрены в главе 4.

Весь код из text-utils.js приведен в приложении 1. Не беспокойтесь, если сейчас он вам кажется непонятным. Когда вы перевернете последнюю страницу, смысл всех вспомогательных функций станет предельно ясным.

В: Значит, DOM используется для работы со страницами HTML?

О: Все верно. Веб-браузеры используют модель DOM для представления веб-страниц. Ваш код JavaScript тоже может использовать DOM для оперативного обновления содержимого страницы.

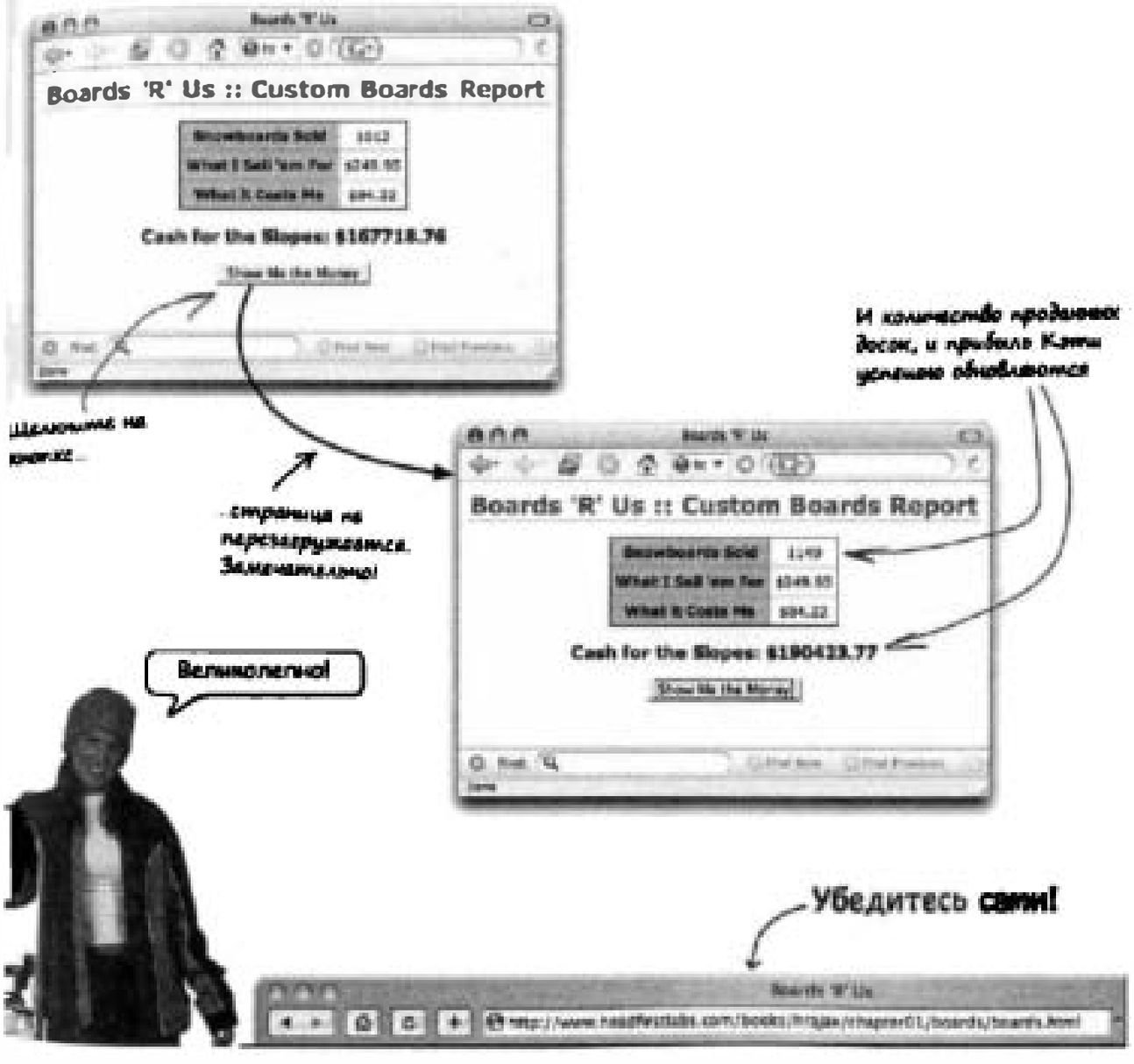
Более того, мы уже немного работали с DOM! Каждый раз, когда вы обращаетесь к объекту JavaScript **document** или вызываете функцию **getElementById()**, вы используете DOM.

В: Насчет свойства readyState... Можно объяснить еще раз?

О: **readyState** — свойство объекта запроса, по которому можно узнать, на какой стадии находится обработка запроса. В следующей главе оно будет рассматриваться довольно подробно, а пока достаточно знать, что когда свойство **readyState** равно 4, сервер завершил обработку вашего запроса.

Демонстрируем Кэти волшебство Ajax

Итак, пора посмотреть, что же мы в итоге сделали для веб-отчета Кэти. Убедитесь в том, что вы включили весь код JavaScript, упоминавшийся в этой главе; лишний раз проверьте, чтобы кнопка Show Me the Money запускала функцию **getBoardsSold()** вместо отправки формы. Загрузите файл boards.html в браузере и посмотрите, на что способна технология Ajax!



Да, мы это сделали! Мы взяли скучный отчет и превратили его в динамическое Ajax-приложение!

Отличная работа. Веб-отчет снова работает; Кэти знает, сколько она зарабатывает, ей не приходится ждать перезагрузок страниц — при таких преимуществах Кэти даже предложит вам бесплатные уроки сноуборда.

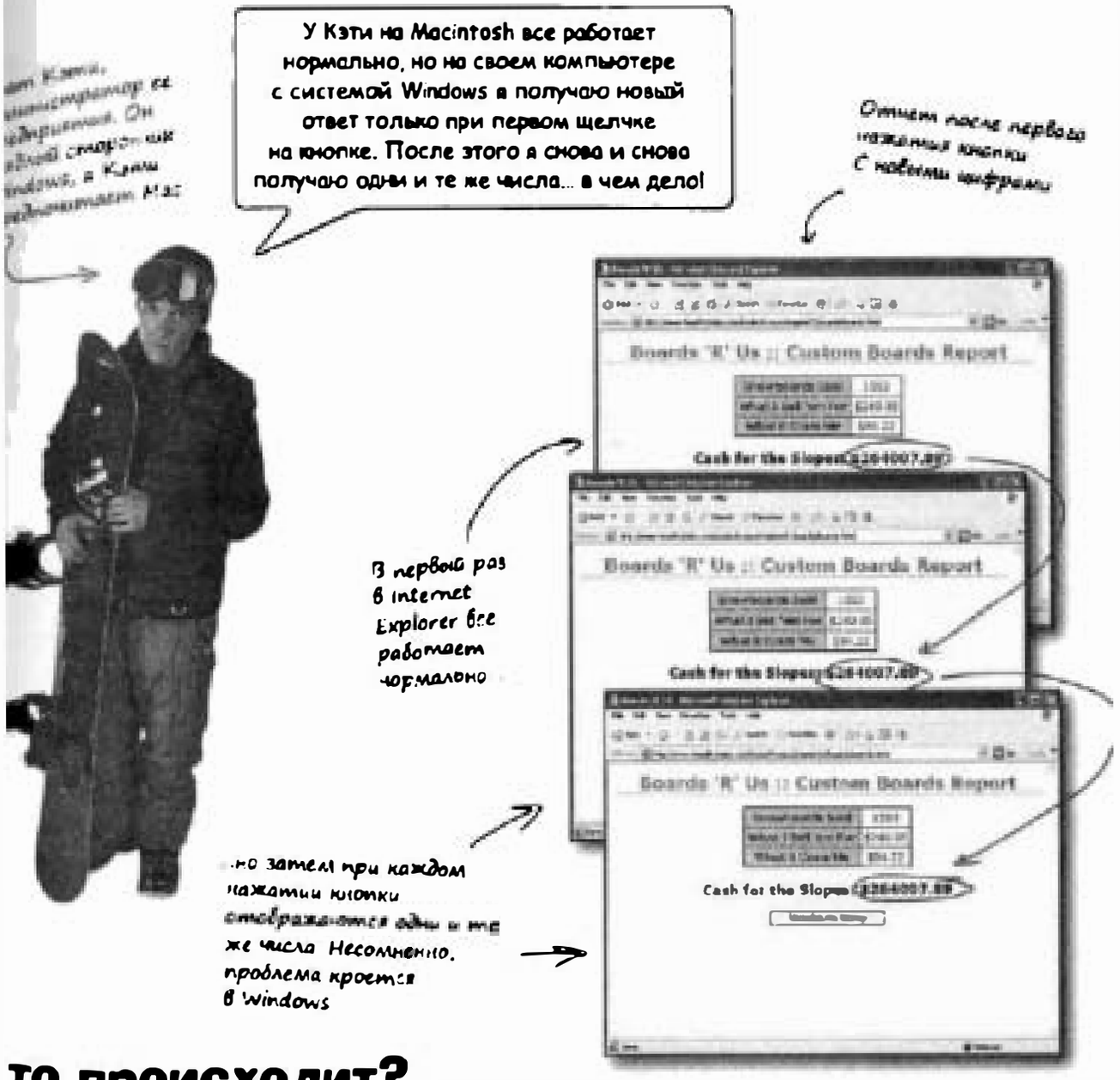
Вспоминаем основные моменты

- Для отправки запросов в асинхронных приложениях используются объекты JavaScript, а не отправка формы.

- Запросы и ответы обрабатываются браузером, а не кодом JavaScript.

- Получив ответ на асинхронный запрос, браузер «вызывает» код JavaScript и передает ему ответ сервера.

Стоп! Рано радоваться!



то происходит?

Мы где-то ошиблись?

Ajax-приложения не работают в Internet Explorer?

Ответы на эти и другие вопросы вы найдете в главе 2...

Обзор на 60 секунд

- Традиционное веб-приложение основано на выдаче запросов серверу и получении ответов — как правило, обновленные данные встроены в полностью сгенерированную страницу HTML.
- В Ajax-приложениях используется асинхронный код JavaScript.
- Ajax-приложения могут выдавать запросы и получать ответы без перезагрузки всей страницы.
- Асинхронный код JavaScript не ожидает, пока сервер вернет ответ. Пользователь продолжает работать со страницей даже в то время, пока сервер обрабатывает запрос.
- Браузер преобразует HTML и CSS в страницу, отображаемую на экране, и обеспечивает выполнение кода JavaScript.
- В Ajax-приложениях сервер обычно возвращает только запрашиваемые данные, без дополнительной разметки и презентации.
- JavaScript может использоваться как для синхронных, так и для асинхронных запросов к серверу.
- Обработчики событий JavaScript позволяют организовать вызов кода JavaScript при выполнении различных условий. Типичные примеры — onChange() и onClick().
- Браузер всегда знает, в каком состоянии готовности находится запрос, и предоставляет эту информацию вашим функциям JavaScript.
- Браузер может запускать функцию JavaScript при каждом изменении состояния готовности запроса. Для этой цели используется свойство атрибутов readystate объекта запроса.
- Если состояние готовности запроса равно 4, значит, запрос был обработан, и у сервера готов ответ.

Просто сделайте это — решение

Откройте файл boards.html и добавьте новую функцию JavaScript с именем **getBoardsSold()**, сразу же после функции **createRequest()**. Затем включите в **getBoardsSold()** строку кода JavaScript для создания нового объекта запроса (шаг «а» на с. 18).

```
<script language="javascript" type="text/javascript">
  var request = null;

  function createRequest() {
    try {
      request = new XMLHttpRequest();
    } catch (trymicrosoft) {
      try {
        request = new ActiveXObject("Msxml2.XMLHTTP");
      } catch (othermicrosoft) {
        try {
          request = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (failed) {
          request = null;
        }
      }
    }

    if (request == null)
      alert("Error creating request object!");
  }

  function getBoardsSold() {
    createRequest();
  }
</script>
```

Готовый код JavaScript, который вам нужно ввести самостоятельно

Начало функции getBoardsSold()...

...которая использует createRequest() для получения нового объекта запроса

КТО Я?

Настало время проверить ваши способности к аналитическому мышлению. В пьесе «Ajax» имеется один важный персонаж. Мы еще с ним не встречались, но к торжественному появлению все готово. Посмотрим, удастся ли вам переиграть нового персонажа, заранее вычислив его личность до того, как вы перевернете страницу. Для упрощения вашей задачи мы приведем несколько высказываний таинственного персонажа

- «Пение? Многие этого не сознают, но я получил классическое образование. И еще я люблю Вагнера».

Браузер Opera хвастается своими вокальными пристрастиями

- «Приведите ко мне усталые, бедные, стесненные угловые скобки, жаждущие получить свой стиль...»

Определенно это браузер, жаждущий HTML-контента

- «Разумеется, я люблю водный спорт. Более того, я был в массовке в фильме «На гребне волны» и чуть не получил главную роль в «Голубой волне»».

Веб-серфинг... Основное занятие браузера

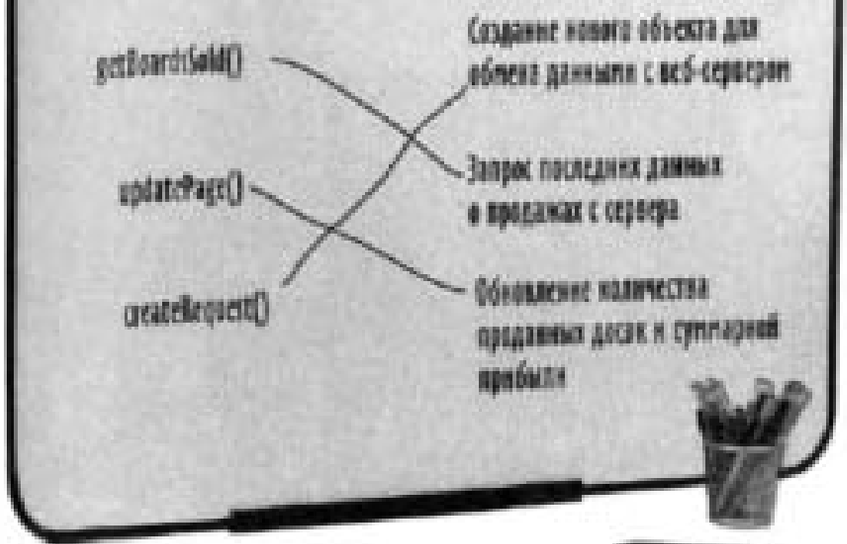
- «Я по-настоящему крут... За последние годы я даже поучаствовал в нескольких войнах».

Netscape против IE, кто кого? Еще не забыли войны браузеров?

КТО Я? Браузер

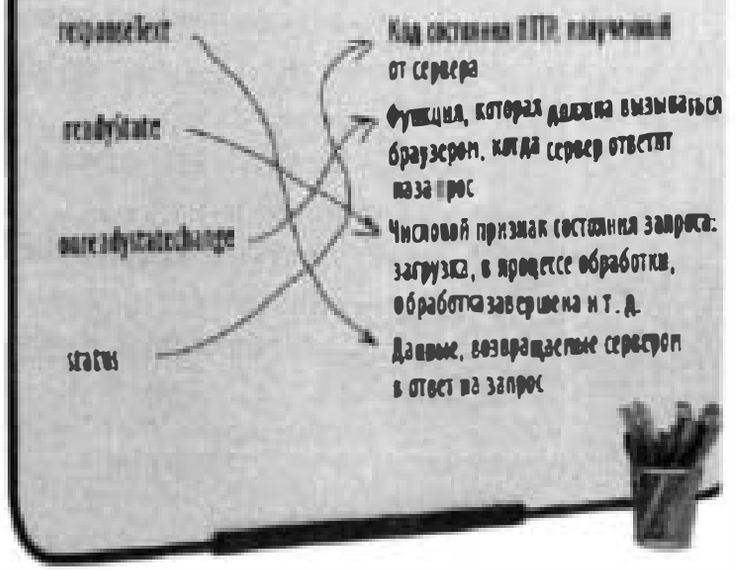
Сделайте это... — Ответ

Для превращения ответа Кэти в Ajax-приложение нам потребуется пара функций JavaScript. Ниже приведены имена трех функций JavaScript. Соедините каждую функцию с описанием того, что она, по вашему мнению, должна делать в итоговой версии приложения Boats.



Сделайте это... — Ответ

Вы уже знаете два свойства объекта запроса, используемых в коде JavaScript. Пришло время проверить полученные знания. В левой колонке перечислены свойства объекта запроса, а в правой — описания этих свойств. Удастся ли вам связать имена свойств с их назначением?



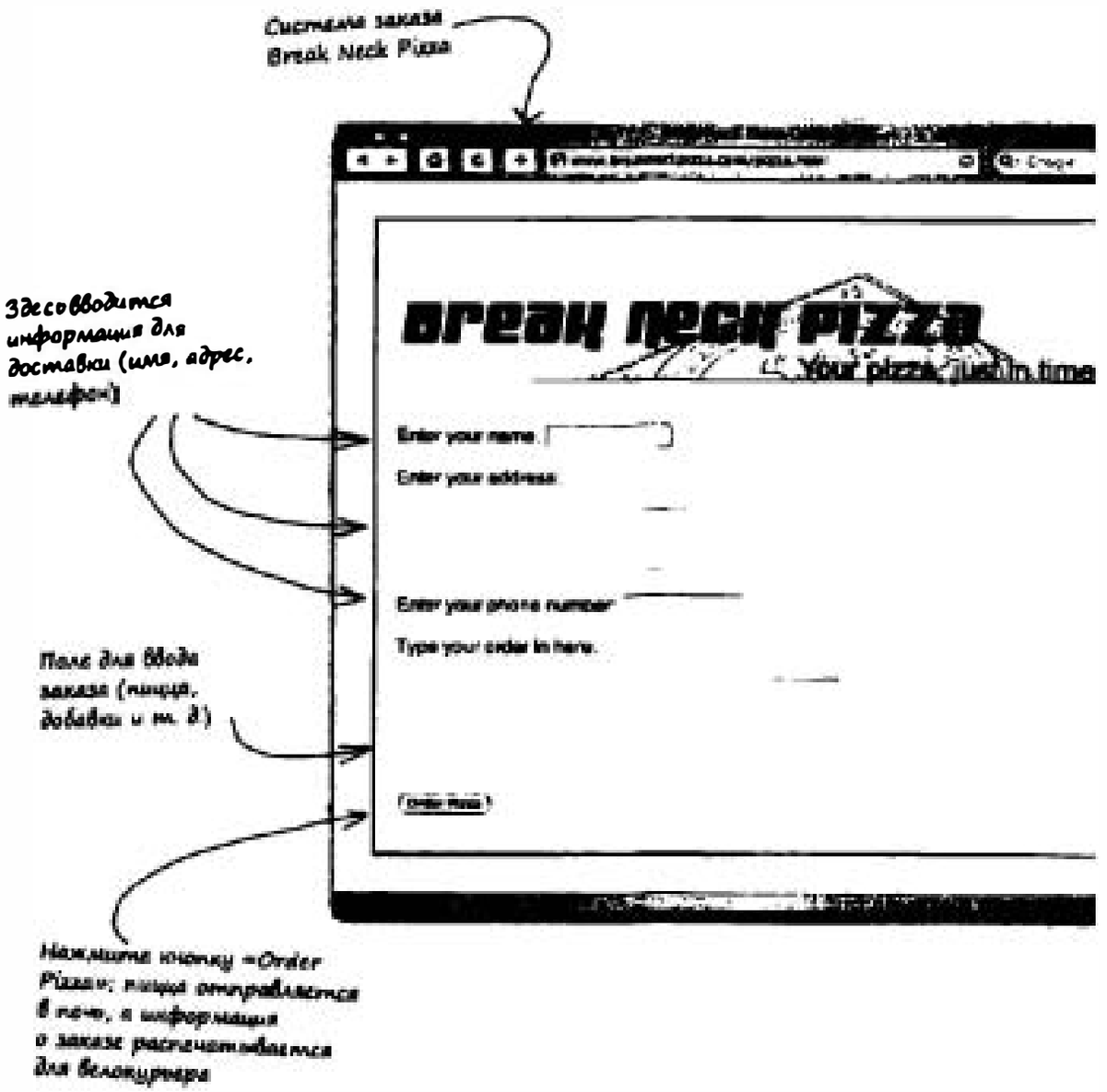
2 Создание запросов Ajax

Учимся говорить на языке Ajax

Пора научиться говорить асинхронно. Чтобы написать очередное мега-приложение, необходимо знать Ajax во всех подробностях. В этой главе представлено сокровенное знание асинхронного кода JavaScript: вы научитесь отправлять запросы разным браузерам, управлять состояниями готовности и статусом, и даже попутно освоите несколько полезных трюков из области динамического HTML-кода. К концу главы вы будете составлять запросы и обрабатывать ответы как настоящий профессионал... кстати, я упомянул, что вашим пользователям не придется ждать, пока вы будете учиться?

Скоростная доставка пиццы

Доставка пиццы за 30 минут? Отстаете от жизни. Служба Break Neck Pizza произвела революцию в области скоростной доставки пиццы благодаря новаторскому кулинарному процессу и распределенной сети велокурьеров. Что еще лучше, Break Neck принимает заказы только на сайте, без общения с оператором Когда вам в следующий раз потребуется пицца за 10 минут— вы знаете, к кому следует обратиться



В чем проблема?

Когда клиент вводит заказ, правильность введенного адреса не проверяется — приказ просто передается Алексу напрямую. Если клиент допустит ошибку в адресе, Алекс тратит много времени на поиск нужного дома. Пицца остывает, а клиент недоволен.

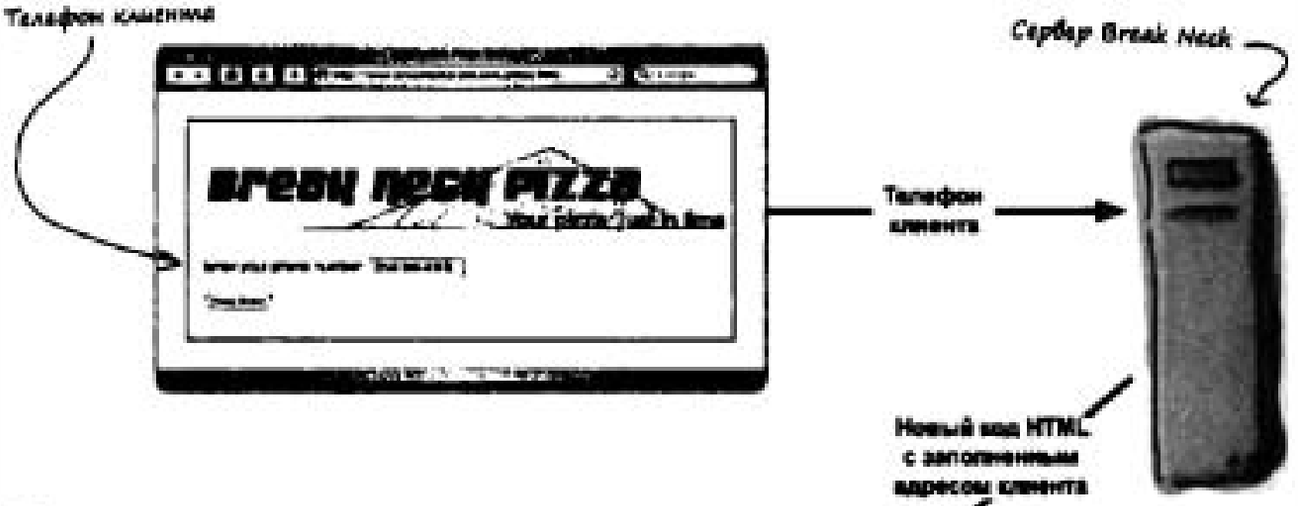
Нужно каким-то образом передать Алексу правильный адрес клиента. Но мы не можем рассчитывать на то, что клиент никогда не ошибается при вводе. Как проверить правильность адреса?

Решение проблемы с доставкой пиццы

К счастью, веб-разработчики уже не первый год занимаются подобными проблемами. Вместо того чтобы заставлять пользователя вводить собственный адрес (с возможными ошибками), мы организуем поиска адреса в базе данных Break Neck, и Алекс всегда будет получать правильный адрес для доставки. Вот как это делается:

1 Клиент вводит свой номер телефона

Клиент вводит номер телефона, который передается на веб-сервер Break Neck



2 Сервер заполнит форму информацией о клиенте

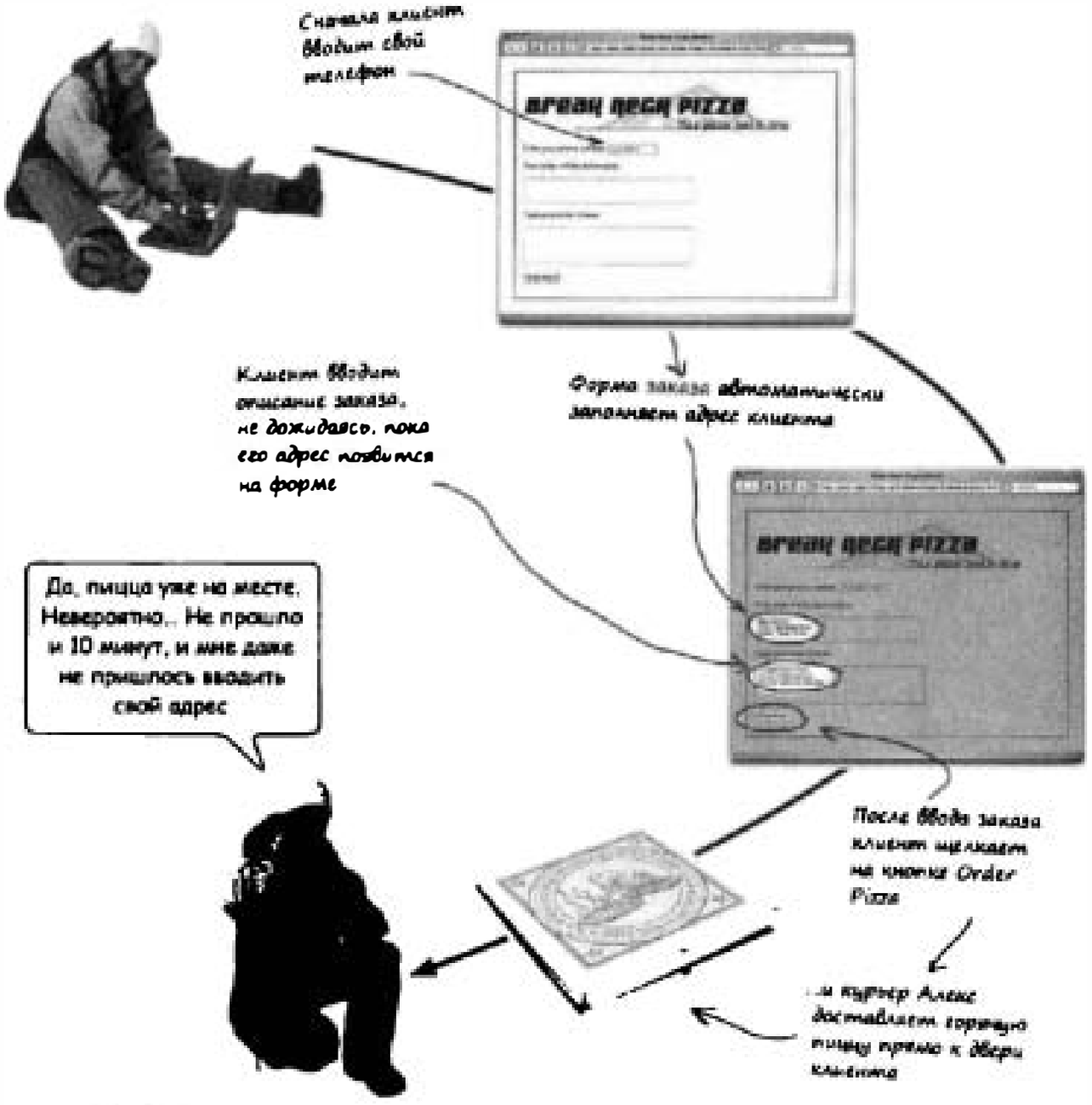
Сервер возвращает новую форму, в коде HTML, которой уже присутствует имя клиента, телефон и адрес.

3 Клиент вводит информацию о заказе

После проверки информации клиенту остается внести на форме описание заказа и щелкнуть на кнопке Complete Order.

Скоростная доставка пиццы в стиле Ajax

Не будем придерживаться старых концепций разработки в стиле 1990-х годов. Лучше давайте посмотрим, как можно разбогатеть приложение Break Neck. Мы хотим превратить форму заказа Break Neck в быстрое, удобное для клиента средство заказа пиццы. Вот как все должно происходить:



Сделайте это...

Как и в главе 3, нам понадобится несколько функций JavaScript, обеспечивающих работу приложения Break Neck. В левом столбце приведены имена двух функций JavaScript, а в правом — несколько строк кода JavaScript. Соедините каждую строку с именем функции, которой принадлежит данная строка.

getCustomerInfo()

updatePage()

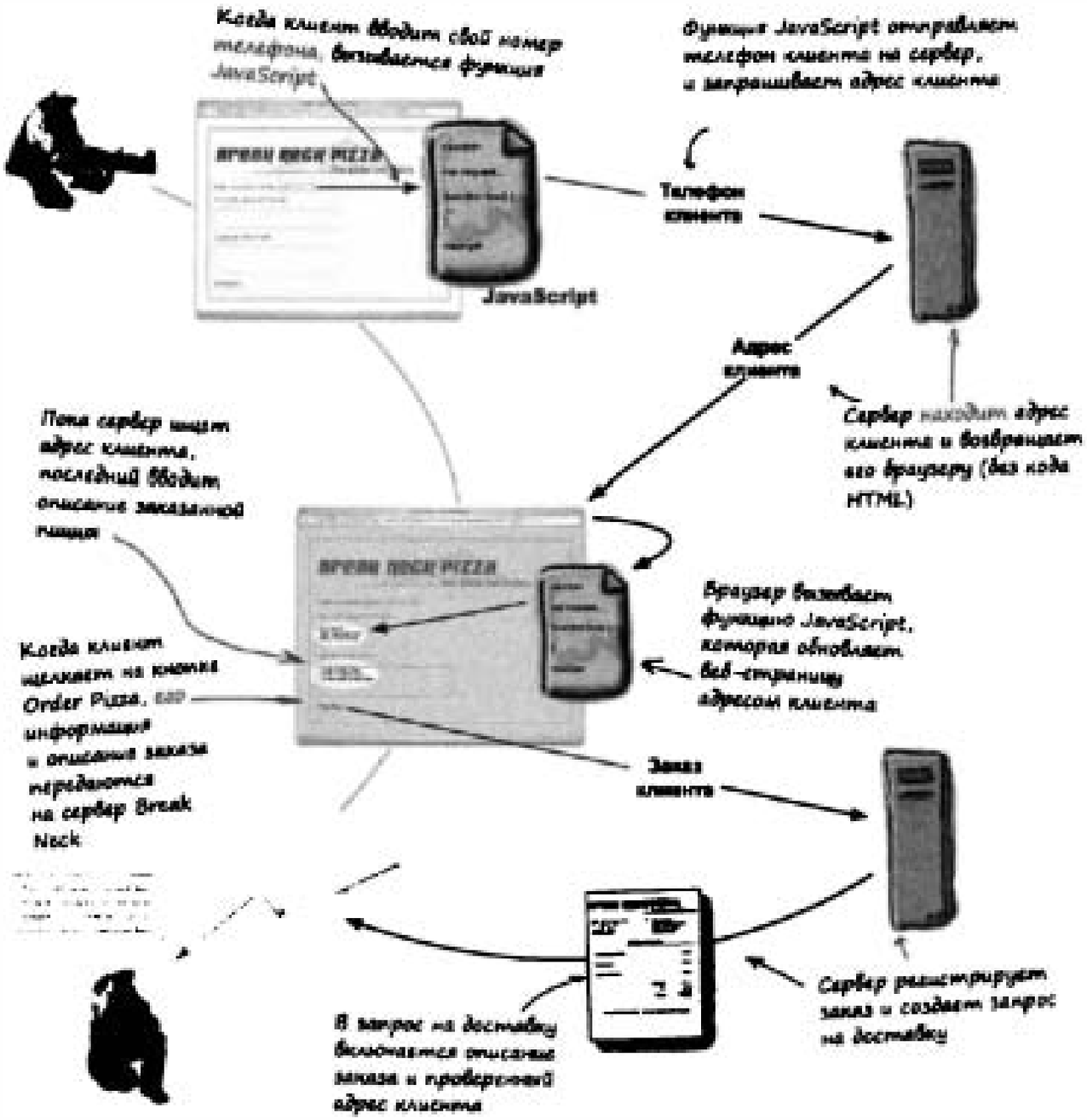
```
request.send(null);

if (request.readyState == 4) { ... }

request.open("GET", url, true);

request.onreadystatechange = updatePage;

var phone =
  document.getElementById("phone").value;

if (request.status == 200) { ... }
```

Мы уже провели одну линию, чтобы вам было проще взяться за задание. Ничего программировать не придется — просто проведите линии от каждой строки кода к той функции, которой она принадлежит.

Ответ на с. 125.

Структурная диаграмма приложения Break Neck

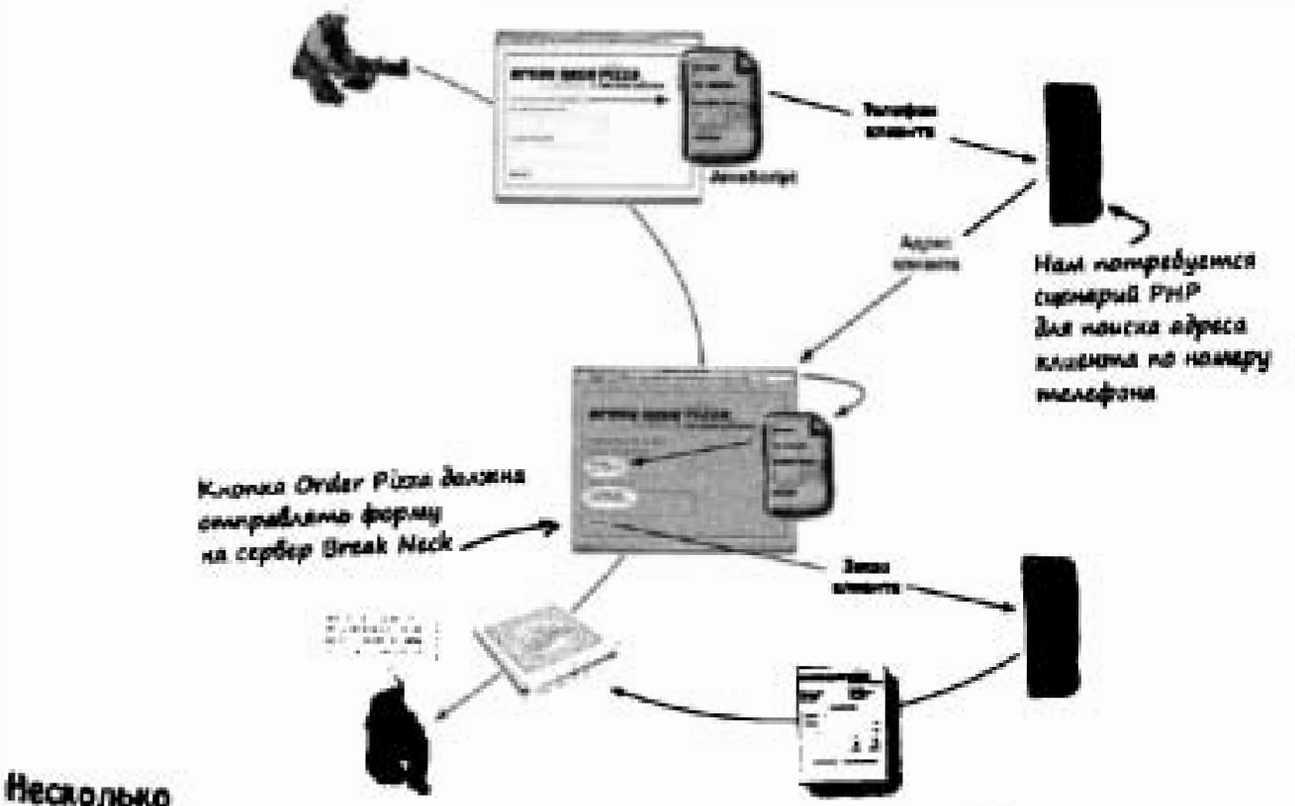
Итак, мы выяснили, как клиенты будут работать с приложением Break Neck. Теперь давайте посмотрим, что происходит «за кулисами».



Сделай сам

Вы уже видели, что должна делать форма заказа Break Neck, и достаточно хорошо представляете, что должно происходить «за кулисами» для превращения Break Neck в удобное и современное веб-приложение. Теперь ваша очередь потрудиться.

Ниже приведена структурная диаграмма приложения Break Neck. Снабдите диаграмму пометками, обозначающими основные действия по ее реализации. Укажите, какие функции нам могут потребоваться; свяжите их с диаграммой. Чтобы вам было проще, мы добавили несколько пометок и примечаний.



Несколько идей:

Нам понадобится сценарий PHP, который будет регистрировать заказ и генерировать запрос на доставку

После того как клиент введет номер телефона, должна запуститься функция JavaScript

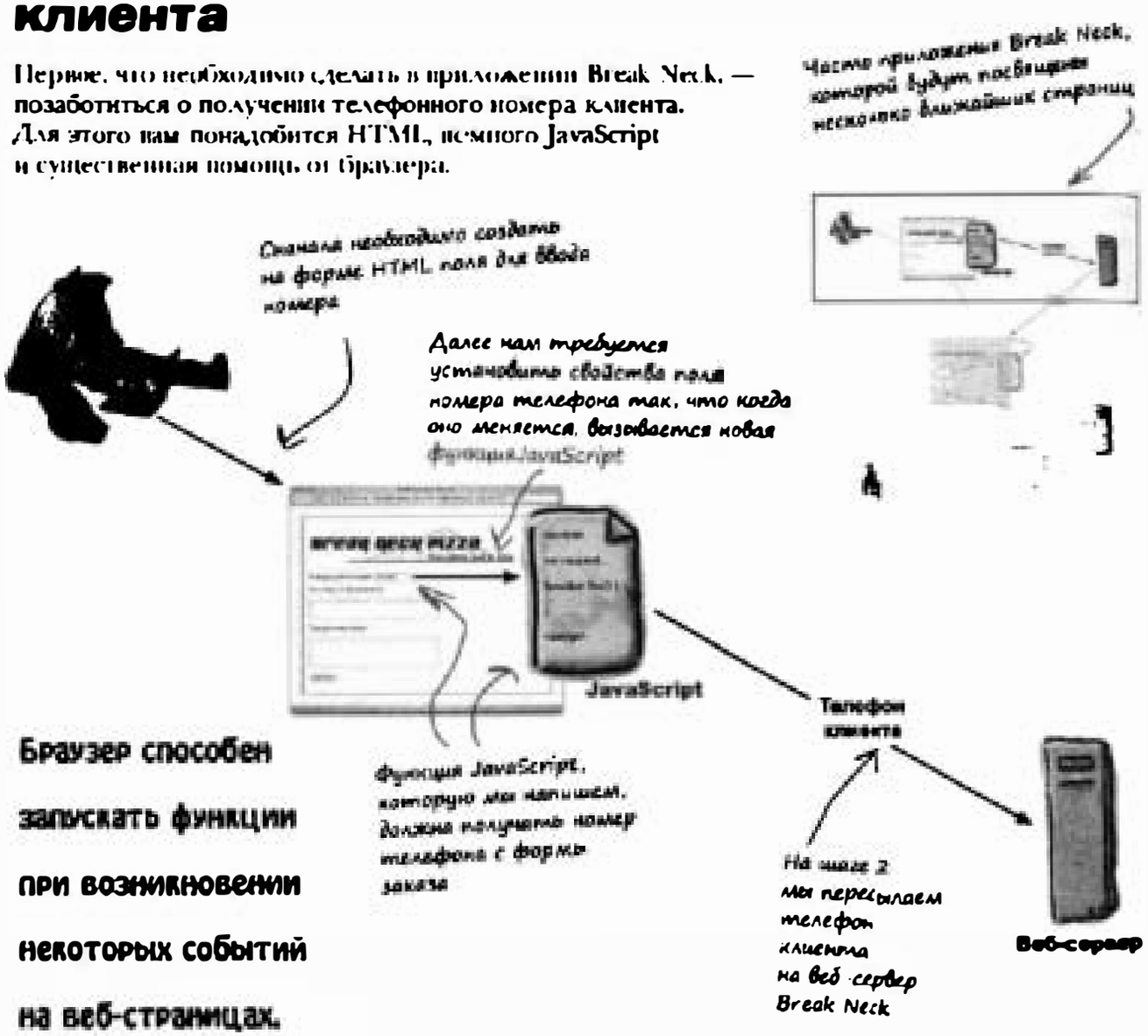
Нам потребуется функция JavaScript для ввода номера телефона клиента и его отправки на сервер Break Neck

Пример запускает написанную нами функцию обратного вызова.

Функция обратного вызова должна обновлять форму заказа, подставляя в нее адрес клиента.

Шаг 1: получение номера телефона клиента

Первое, что необходимо сделать в приложении Break Neck, — позаботиться о получении телефонного номера клиента. Для этого вам понадобится HTML, немного JavaScript и существенная помощь от браузера.



Браузер способен запускать функции при возникновении некоторых событий на веб-страницах, а также предоставлять коду JavaScript доступ к информации, введенной пользователем на странице.

Краткий курс HTML: ввод пользовательских данных

Форма заказа уже содержит поля, в которых пользователь вводит номер телефона, адрес и заказанную пиццу. Рассмотрим код HTML формы заказа и попробуем связать его с кодом JavaScript, который нам еще предстоит написать.

```
<html>
 <head>
  <title>Break Neck Pizza Delivery</title>
  <link rel="stylesheet" type="text/css" href="breakneck.css"
/>
 </head>
 <body>
  <p>
   <img src="breakneck-logo.gif"
        alt="Break Neck Pizza" />
  </p>
  <form method="POST" action="placeOrder.php">
   <p>Enter your phone number:
    <input type="text" size="14" name="phone" />
   </p>
   <p>Your order will be delivered to:</p>
   <p><textarea name="address" rows="4" cols="50">
      </textarea></p>
   <p>Type your order in here:</p>
   <p><textarea name="order" rows="6" cols="50"></textarea></p>
   <p><input type="submit" value="Order Pizza" /></p>
  </form>

 </body>
</html>
```

Все стили приложения Break Neck определяются во внешней таблице CSS

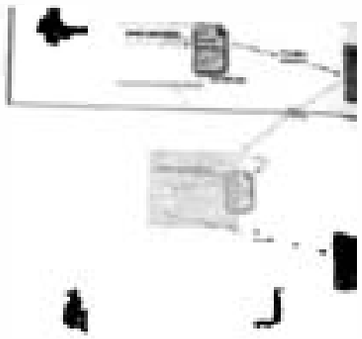
URL для регистрации заказа после заполнения формы

Здесь клиент вводит свой номер телефона

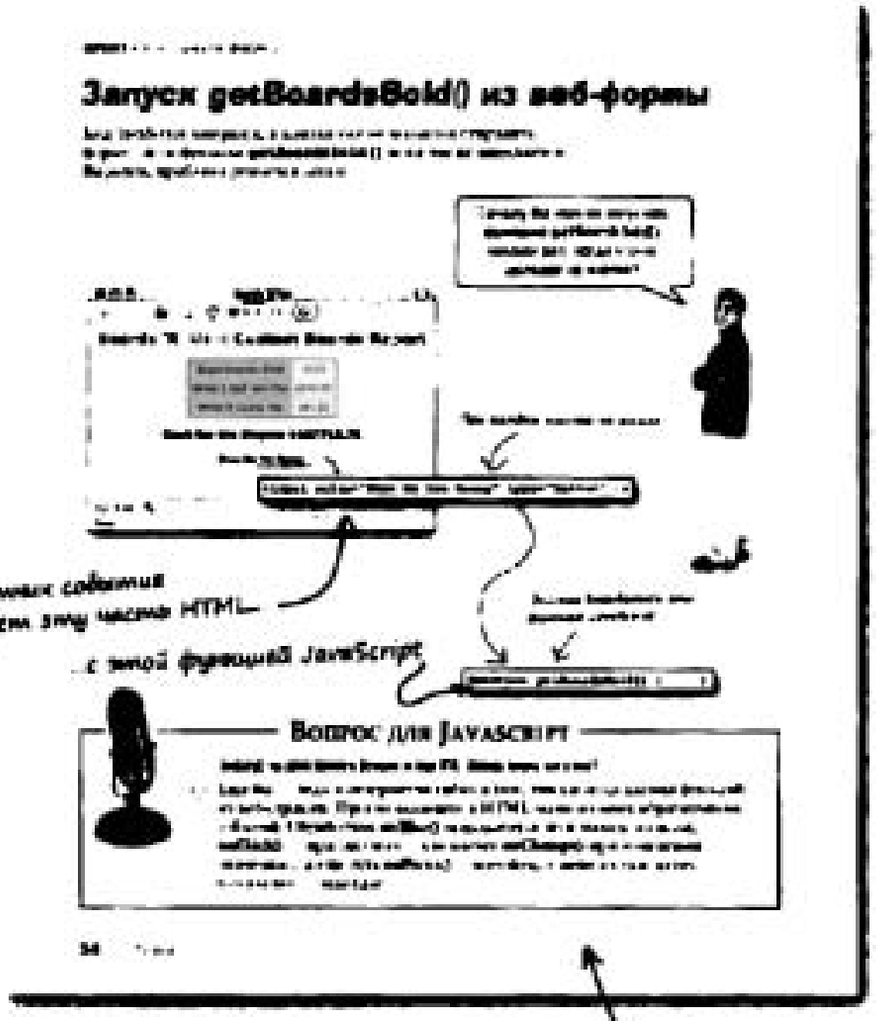
Заполняется адресом, возвращенным сервером. Оставим элемент полем, чтобы клиент при желании мог ввести другой адрес

Для отправки заказа используется обычная кнопка Submit (метод POST формы)

Обработчики событий связывают HTML с JavaScript



Помните обработчики событий из главы 1? Мы использовали обработчик `onClick` для связывания кнопки на странице HTML с функцией JavaScript. Давайте кратко вспомним главу 1:



Помните? Мы связали кнопку в приложении Boards с функцией JavaScript при помощи обработчика события

Обработчик события связывает эту часть HTML

с этой функцией JavaScript

Один из этих обработчиков событий свяжет поле телефонного номера с функцией JavaScript

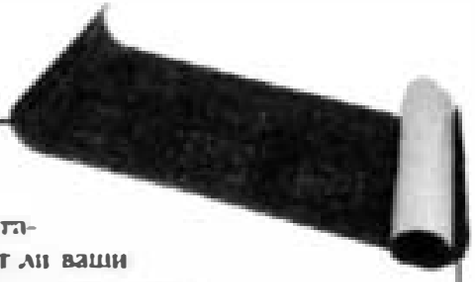
Сначала планирование, потом программирование

Иногда бывает лучше провести небольшое планирование перед тем, как браться за написание кода. В нашем случае стоит заранее решить, как мы собираемся вызывать функцию JavaScript, обращающуюся с запросом к серверу Break Neck (несмотря на то, что мы еще не начинали ее программировать). Назовем эту функцию `getCustomerInfo()`; как следует из имени, функция запрашивает с сервера информацию о клиенте.

Теперь, когда вы знаете имя функции, попробуйте обновить код HTML формы так, чтобы функция `getCustomerInfo()` запускались при вводе телефонного номера... при том, что мы займемся программированием этой функции лишь через несколько страниц.

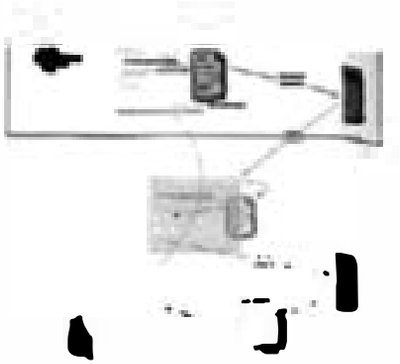
Возможно, вам стоит вернуться к схеме приложения на с. 73 и включить имя функции JavaScript в свои пометки

Поговорим о планировании...



Ищете ответы к упражнению «Сделай сам» на с. 73? Вы их найдете... на протяжении всей главы. По мере знакомства с материалом внимательно следите за тем, как мы решили организовать приложение для заказа пиццы. Проверьте, совпадут ли ваши конструктивные решения с нашими, и подумайте, что бы вы сделали иначе.

Обработчики событий



Существует немало способов присоединения кода JavaScript к страницам HTML. Далее представлены лишь некоторые, наиболее популярные обработчики событий, с краткими описаниями. Присмотритесь повнимательнее: один из этих обработчиков будет использоваться на следующей странице...

onChange

Событие **onChange** происходит при каждом изменении содержимого поля формы — например, при вводе нового или удалении существующего значения.

```
<input type="text"
       name="street"
  onChange="updateMap();" />
```

Так событие onChange задается в коде HTML

Условия возникновения:

- ___ Вход в поле
- ___ Выход из поля
- ✓ Изменение поля

onFocus

```
<input type="text" name="state"
       onFocus="popupStates();" />
```

Обработчик события onFocus в действии...

Любой код JavaScript, связанный с событием **onFocus**, выполняется при получении фокуса полем или другим компонентом страницы (в результате перебора или щелчка на компоненте).

Условия возникновения:

- ✓ Вход в поле
- ___ Выход из поля
- ___ Изменение поля

onBlur

Обработчик **onBlur** используется для запуска программного кода при потере фокуса полем (в результате перебора или щелчка на компоненте).

Код вызывается при потере фокуса полем zipCode

Условия возникновения:

- ___ Вход в поле
- ✓ Выход из поля
- ___ Изменение поля

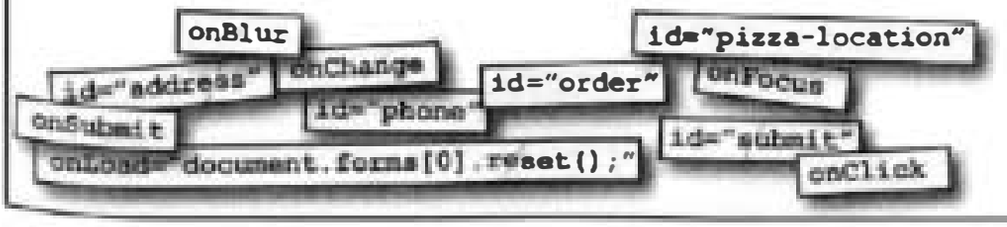
```
<input type="text"
       name="zipCode"
  onBlur="validateZip();" />
```

Все готово к обновлению формы Break Neck. Прежде всего необходимо добавить обработчик события для запуска функции getCustomerInfo(), которую мы написали через несколько страниц. Напомним, что эта функция должна запускаться каждый раз, когда клиент вводит новый номер.

Затем к форме заказа добавляются атрибуты id; они понадобятся позднее, при обращениях к полям формы из кода JavaScript. А если уж мы все равно этим занялись, почему бы не очищать форму при каждой загрузке?

Ниже приведен код HTML текущей версии формы. Чтобы обновить форму, расставьте правильные фрагменты HTML и JavaScript из нижней части кроссворда по пропускам в разметке.

```
<body _____>
 <p>
  <img src="breakneck-logo.gif" alt="Break Neck Pizza" />
 </p>
 <form method="POST" action="placeOrder.php">
  <p>Enter your phone number:
   <input type="text" size="14" name="phone"
          _____="getCustomerInfo();" />
  </p>
  <p>Your order will be delivered to:</p>
  <p><textarea name="address" _____
             rows="4" cols="50"></textarea></p>
  <p>Type your order in here:</p>
  <p><textarea name="order" _____
             rows="6" cols="50"></textarea></p>
  <p><input type="submit" _____
            value="Order Pizza" /></p>
 </form>
</body>
```



Разложите по полкам

Решения

А вот как должна выглядеть готовая веб-форма Break Neck Pizza. Проверьте, совпадают ли приведенные ответы с вашими.

Очистка первой формы (forms[0]) в документе HTML

```
<body onload="document.forms[0].reset();">
 <p>
  <img src="breakneck-logo.gif" alt="Break Neck Pizza" />
 </p>
 <form method="POST" action="placeOrder.php">
  <p>Enter your phone number:
   <input type="text" size="14" name="phone"
          onchange="getCustomerInfo();" />
  </p>
  <p>Your order will be delivered to:</p>
  <p><textarea name="address" id="address"
            rows="4" cols="50"></textarea></p>
  <p>Type your order in here:</p>
  <p><textarea name="order" id="order"
            rows="6" cols="50"></textarea></p>
  <p><input type="submit" id="submit"
          value="Order Pizza" /></p>
 </form>
</body>
```

onChange следит за тем, чтобы при каждом изменении телефонного номера запускалась функция getCustomerInfo()

Неиспользованные фрагменты

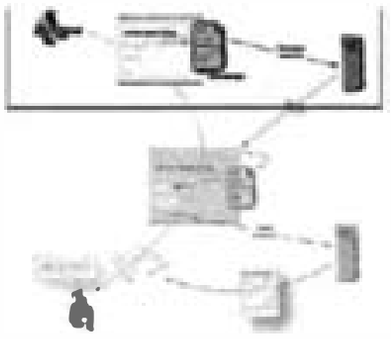
onBlur
onFocus
id="phone"
onClick
onSubmit
id="pizza-location"

Часто задаваемые вопросы

В: Я использовал обработчик onBlur вместо onChange. Не подойдет?

О: Обработчик onBlur активизируется каждый раз, когда поле с номером телефона теряет фокус, так что этот вариант тоже годится. Но при использовании onBlur функция getCustomerInfo() будет запускаться даже в том случае, если клиент не изменял содер жимое поля. С другой стороны, обработчик onChange активизируется только при изменении телефона; вероятно, такое решение следует признать более удачным.

Переходим к JavaScript



Теперь можно заняться и кодом JavaScript. Вы уже знаете имя функции, которая получает с формы телефон клиента: `getCustomerInfo()`. Эта функция должна отправить телефон клиента серверу Break Neck и запросить адрес клиента.

Начнем с имени функции:

```
function getCustomerInfo() {
[redacted]
}
```

Здесь будет размещаться весь код

Первое, что необходимо сделать, — получить номер телефона клиента из формы HTML

Шевели мозгами

Как вы думаете, могут ли возникнуть проблемы с обработчиками событий, запускающим код JavaScript с вызовом асинхронных запросов? Что происходит, если форма выдает запрос на получение адреса, а затем отправит другой запрос прежде, чем будет получен ответ на первый запрос? Как вы думаете, что увидит клиент на форме заказа?

Просто сделайте это

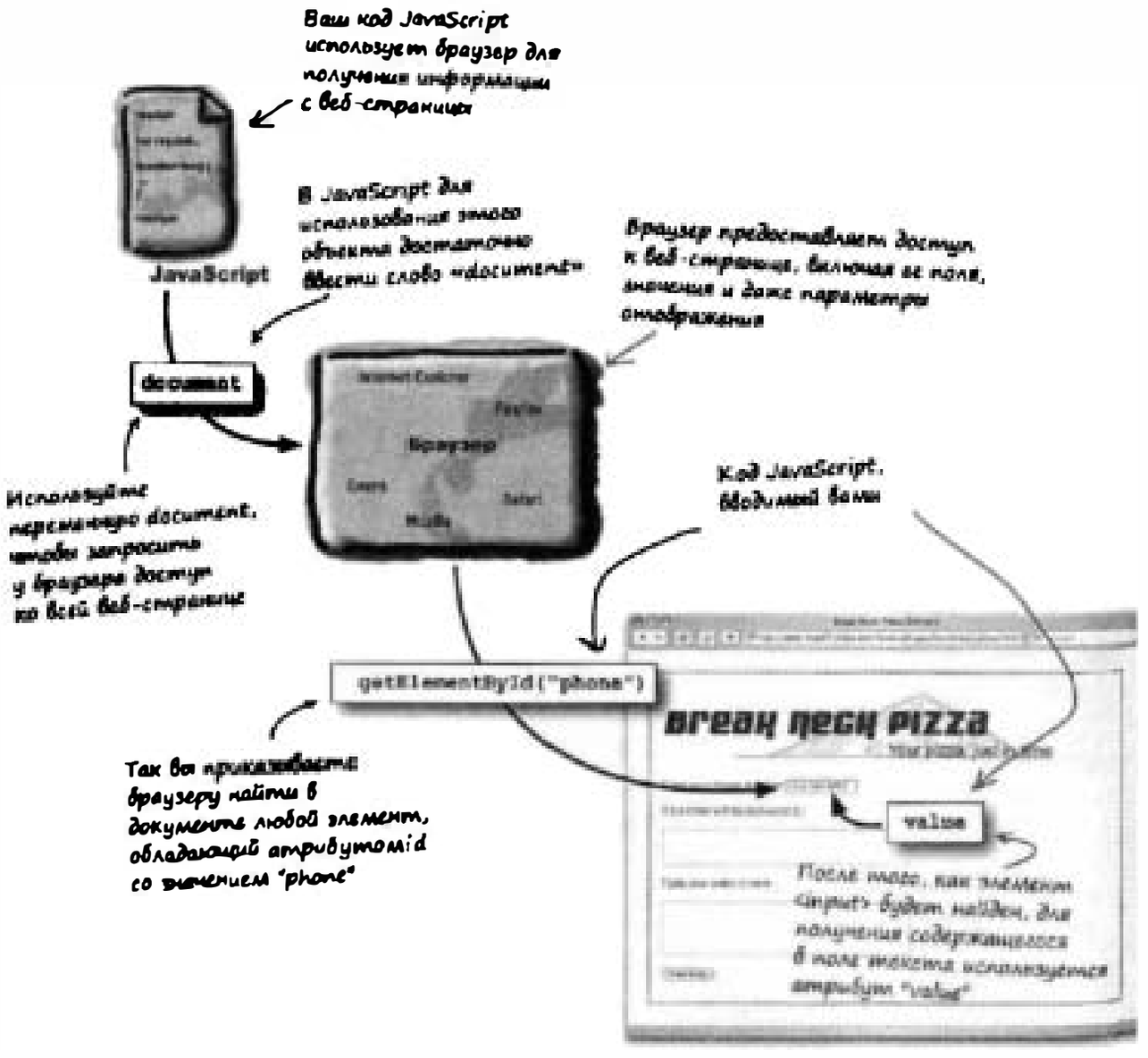
Откройте в примерах папку chapter02/breakneck и найдите в ней файл pizza.html — форму заказа Break Neck. Чтобы форма заработала с положенной скоростью, в нее необходимо внести ряд изменений.

Для начала проверьте, что код HTML совпадает с ответом из упражнения на с. 80. Затем включите в секцию <head> кода HTML теги <script>, как в приложении Boards главы 1. Наконец, введите пустой заголовок функции getCustomerInfo(). Мы заполним эту функцию кодом на нескольких ближайших страницах.

Не переходите к следующей странице, пока не обновите свою версию pizza.html

Используем DOM для получения номера телефона

Для получения из кода JavaScript номера телефона, введенного клиентом на форме Break Neck, можно воспользоваться моделью DOM (Document Object Model). Модель DOM будет достаточно подробно рассмотрена в главе 4, а пока просто рассматривайте DOM как механизм обмена информацией с веб-страницей, отображаемой в браузере.



Картина постепенно проясняется

Давайте воспользуемся DOM для получения номера телефона клиента. Ниже мы соединили фрагменты кода DOM, приведенного на предыдущей странице, и включили их в пустую функцию `getCustomerInfo()`:

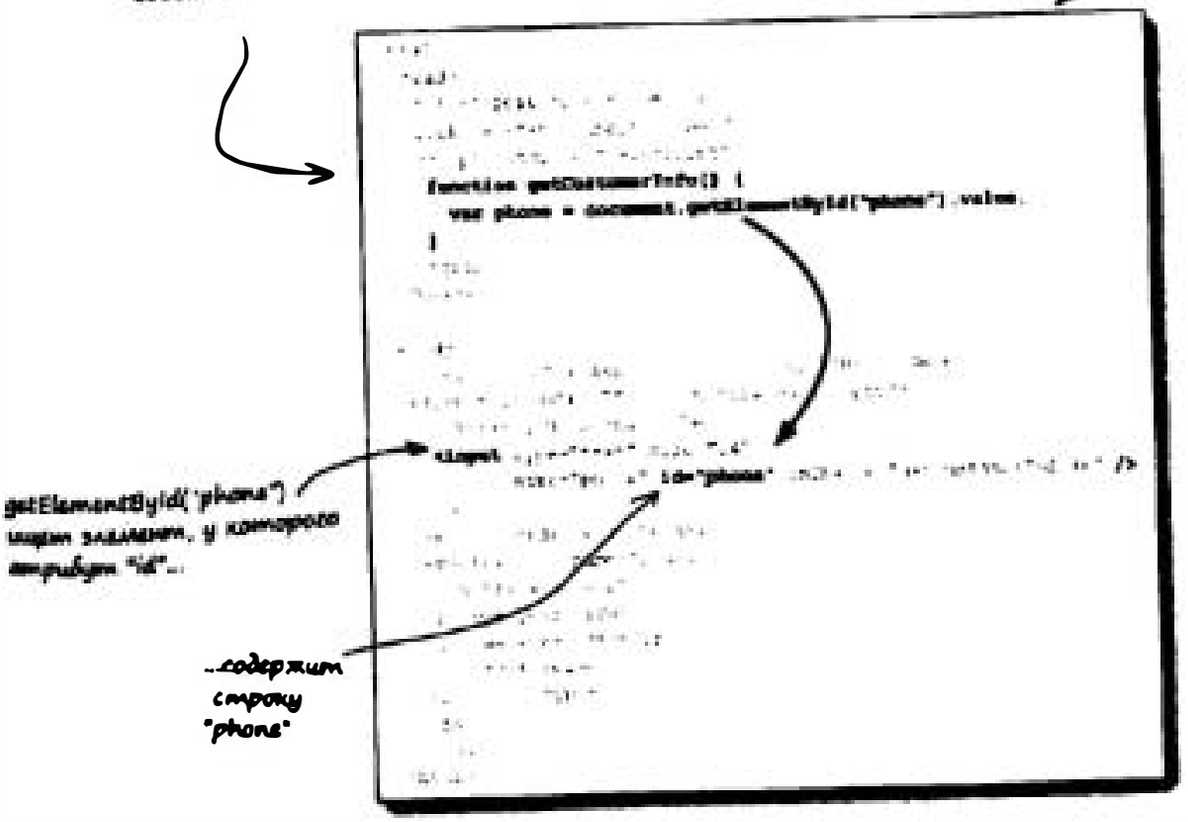
```
function getCustomerInfo() {
  var phone = document.getElementById("phone").value;
}
```

Переменная для хранения номера телефона

Объединяем фрагменты JavaScript с предыдущей страницы в одну строку кода

Включаем эту строку в свою версию pizza.html

Вся страница HTML представлена в JavaScript объектом "document"



getElementById('phone') ищет элемент, у которого атрибут "id"...

...содержит строку "phone"

Это для шага 1! На шаге 2...

Шаг 2: Запрос адреса клиента

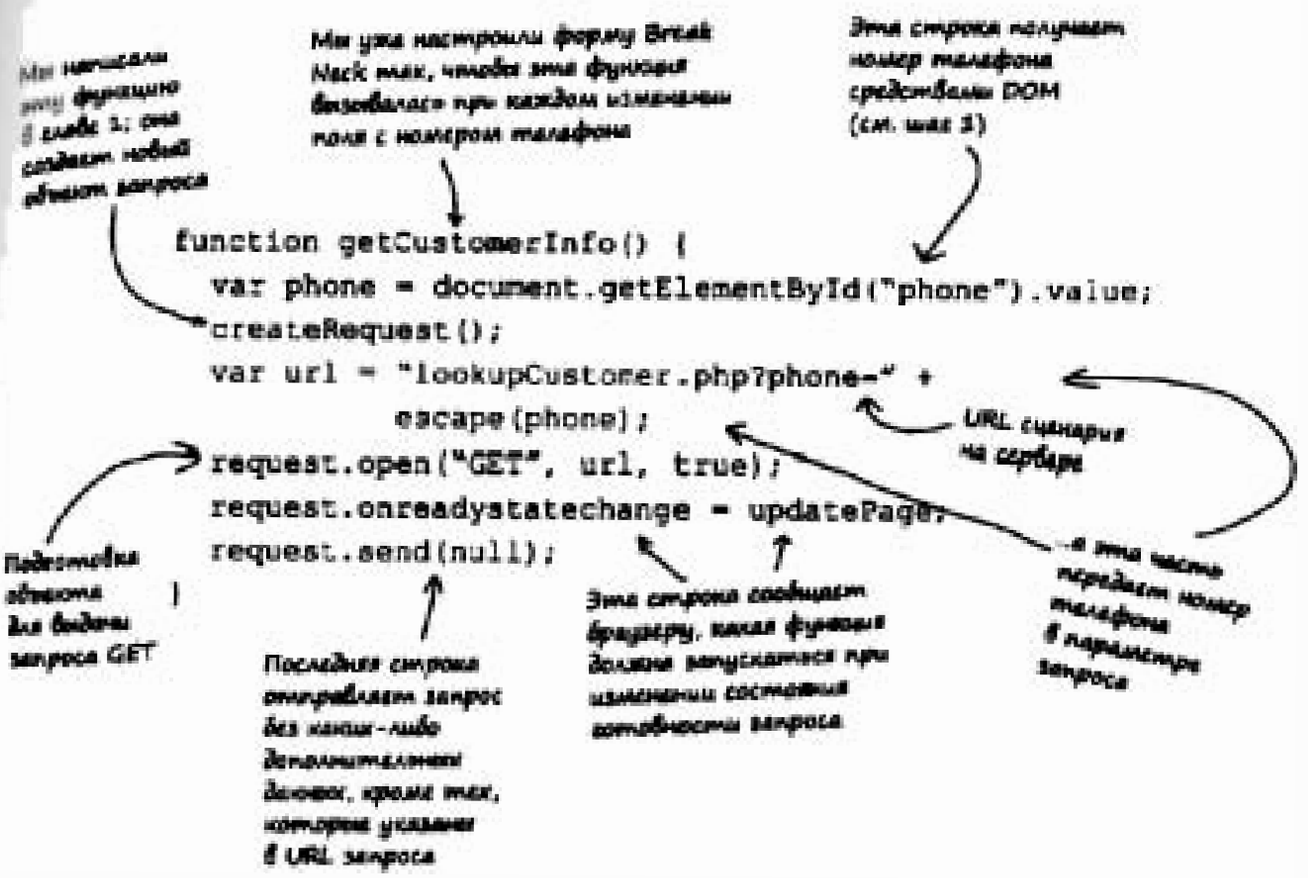
На следующем шаге мы отправляем номер телефона, полученный на шаге 1, веб-серверу, и запрашиваем соответствующий ему адрес клиента.

Где же браузер?

Задействован ли браузер на этом шаге? Если вы думаете, что задействован, поставьте пометку и укажите, где именно.

getCustomerInfo() в двух словах

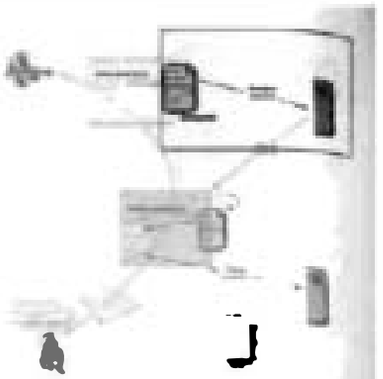
Вероятно, после написания обновленной версии приложения Boards из главы 1 вы уже достаточно хорошо разбираетесь в выдаче запросов GET. Далее приведен код функции `getCustomerInfo()`; основная часть кода JavaScript имеет многообщего с кодом, написанным нами для главы 1.



Что-что?

Ничего страшного, если у вас остались вопросы относительно этого кода. Каждая его строка будет подробно рассмотрена в этой главе, так что не старайтесь исчерпывающе разобраться во всем сейчас.

Создание объекта запроса



Наша задача на этом шаге — выдать запрос на сервер Break Neck. Для этого нам понадобится объект запроса, который может использоваться в функции JavaScript. К счастью, код для создания объекта запроса уже был написан в главе 1.

Помните функцию `createRequest()`? Давайте снова рассмотрим код JavaScript и убедимся в том, что он подходит для приложения Break Neck:

Начало файла recipebox.js с добавленной функцией createRequest() из главы 1

Запрос создается вне функции, поэтому переменная запроса доступна для всех функций

```
var request = null;

function createRequest() {
  try {
    request = new XMLHttpRequest();
  } catch (tryMicrosoft) {
    try {
      request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (otherMicrosoft) {
      try {
        request = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (failed) {
        request = null;
      }
    }
  }

  if (request == null)
    alert("Error creating request object!");
}
```

Готовый код JavaScript из главы 1

Вспомните: для разных браузеров приходится использовать разный код

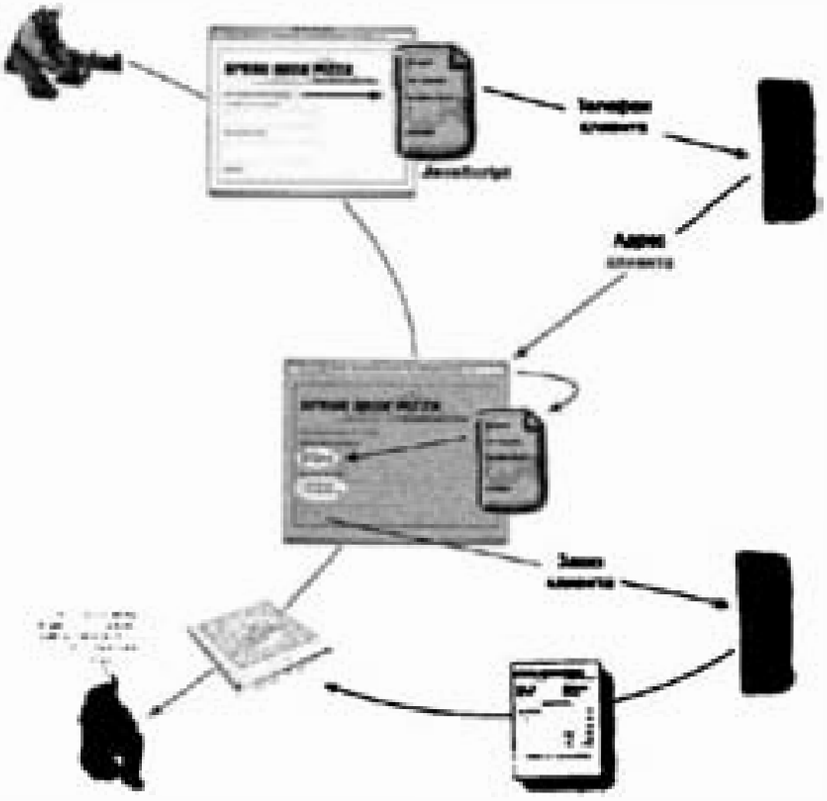
Проверка ошибок подтверждает, что все идет правильно

На последующих ближайших страницах мы досконально разберемся, что делает каждая строка этого кода JavaScript

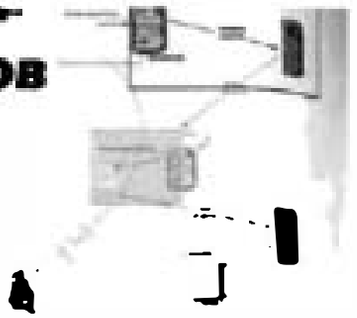
Планы меняются

Помните схему приложения Break Neck на с. 73? С тех пор вы узнали много нового; сейчас вам предоставляется возможность немного скорректировать свои планы. Вернитесь к исходной схеме и посмотрите — может, вам захочется что-нибудь предложить? Работа над Break Neck еще далека от завершения.

Добавьте на схему внизу новые пометки — что, по вашему мнению, нужно сделать, чтобы приложение Break Neck сокрушило своих конкурентов? Если вы согласны со всеми пометками, сделанными на с. 73, повторите их на этой схеме и похлопайте себя по плечу за то, что все сделали правильно с первого раза.



Поддержка различных браузеров



Настало время проанализировать готовый код JavaScript и подробно разобраться, что же в нем происходит. Давайте последовательно, шаг за шагом переберем каждую строку `createRequest()`

❶ Объявление переменной запроса

Сначала мы объявляем новую переменную для представления объекта запроса. Переменная будет использоваться в коде JavaScript.

```
var request = null;
```

Напомним: переменная не объявляется внутри функции — Она просто болтается в теге <script> страницы pizza.html

Переменная, не объявленная внутри функции, может использоваться любой функцией JavaScript

❷ Попытка создания XMLHttpRequest для большинства браузеров

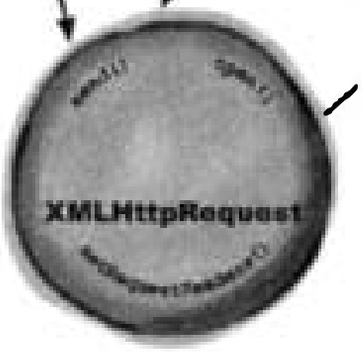
Затем мы определяем новую функцию с именем `createRequest()`. Работа этой функции должна начаться с попытки создания нового объекта запроса с типом XMLHttpRequest; объект должен создаваться успешно во всех браузерах, кроме Internet Explorer.

Переменная должна ссылаться на объект запроса JavaScript

```
function createRequest() {
  try {
    request = new XMLHttpRequest();
  } catch (trymicrosoft) {
    // Try something different
    //   for Microsoft
    //   (check out step 3)


  if (request == null)
    alert("Error creating XMLHttpRequest!");
}
```

XMLHttpRequest работает в Safari, FireFox, Mozilla, Opera и большинстве альтернативных браузеров

Если попытка завершилась неудачно, пробуем другой вариант



Если и после этого переменная запроса равна null, следует вывести сообщение об ошибке

Попытка создания объекта ActiveXObject для браузеров Microsoft

В блоке catch мы пытаемся создать объект запроса, используя один из Microsoft-совместимых типов... Для этого каждый тип будет проверяться в отдельном блоке try/catch:

```
try {
  request =
    new ActiveXObject("Msxml2.XMLHTTP");
} catch (othermicrosoft) {
  try {
    request =
      new ActiveXObject("Microsoft.XMLHTTP")
  } catch (failed) { request = null; }
}
```

Поддерживается большинством версий IE...

...но для некоторых версий нужен другой тип

Об-ой. Если управление будет передано этой строке, значит, у нас проблемы. Следует убедиться в том, что переменная запроса по-прежнему равна null

Снова наша переменная запроса

ActiveXObject работает в Internet Explorer

Внимание! В IE5 для Mac этот способ не работает, даже со специализированным кодом для IE. Впрочем, если вы используете IE на Mac... что тут можно сказать?

Теперь соберем все вместе...

```
var request = null;
function createRequest() {
  try {
    request = new XMLHttpRequest();
  } catch (trymicrosoft) {
    try {
      request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (othermicrosoft) {
      try {
        request = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (failed) {
        request = null;
      }
    }
  }
  if (request == null)
    alert("Error creating XMLHttpRequest!");
}
```

Для большинства браузеров — таких как Safari и Firefox

Для любителей Internet Explorer

Обработка ошибок, если при создании объекта возникнут проблемы

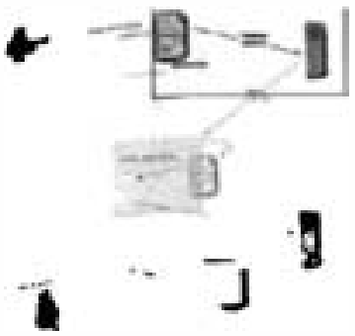
Не забывайте общую картину — мы все еще работаем над передачей запроса веб-серверу Break Neck

Просто сделайте это

Пора принять участие в войнах браузеров. Включите код JavaScript `getCustomerInfo()` и `createRequest()` в свою копию pizza.html. Закомментируйте части `createRequest()`, создающие объект запроса для вашего браузера.

Если вы используете Internet Explorer, закомментируйте код с ActiveXObject; для альтернативных браузеров закомментируйте строки с созданием объекта типа XMLHttpRequest. Теперь загрузите pizza.html в браузере и введите номер телефона. Вы получите сообщение об ошибке следующего вида:

Internet Explorer сообщает об ошибке, потому что часть кода с объектом ActiveXObject в функции createRequest() была закомментирована



Не раздражайте клиента!

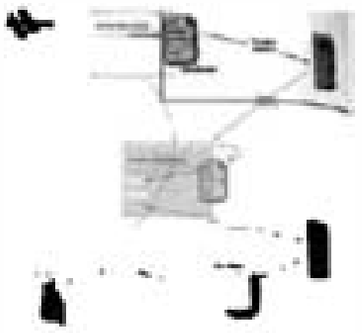
В приложении Break Neck клиенту достаточно заполнить только одно поле перед тем, как будет вызвана функция getCustomerInfo(), и программа попытается создать новый объект запроса. Если создать объект не удастся, клиент получит сообщение об ошибке

Но представьте, как неприятно заполнить всю форму, и только потом узнать о возникших проблемах... сколько времени потеряно напрасно!

Похоже, мы должны найти способ оповещения пользователей на более ранней стадии... например, до того, как они начнут вводить текст на форме Break Neck.

ШЕВЕЛИМ МОЗГАМИ

Обратитесь к шагу 1 на с. 88 и подумайте над тем, как мы обрабатываем создание переменной запроса. Может, вы придумаете какой-нибудь способ оповещения клиентов о возникающих проблемах до начала работы с формой Break Neck?



Но ведь мы не должны запустить функцию createRequest(), верно? Где еще можно создать объект запроса?

Код JavaScript может находиться вне функции

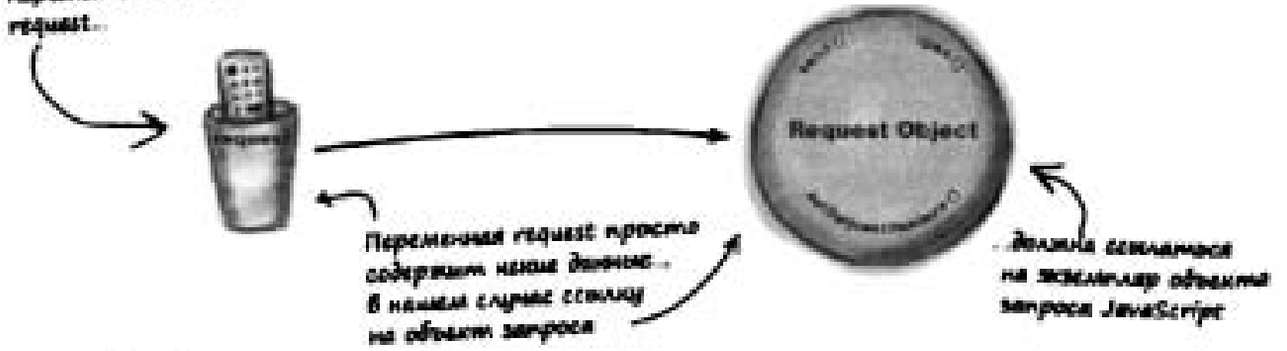
Помните, как мы объявили переменную **request**, не включая эту строку кода ни в одну функцию?

```
<head>
  <title>Break Neck Pizza Delivery</title>
  ...
  <script language="JavaScript" type="text/javascript">
    var request = null;
  </script>
</head>
```

Код выполняется автоматически при загрузке страницы

Любой код JavaScript вашей веб-страницы, не принадлежащий ни одной функции, запускается статически. Это означает, что при загрузке страницы браузер автоматически выполняет весь код JavaScript, не входящий в функции, до того, как пользователь приступит к вводу данных или начнет щелкать на кнопках.

Итак, если мы выведем весь код createRequest() за пределы функции, то весь код, в котором мы пытаемся создать объект запроса, будет выполняться сразу же после загрузки формы Break Neck. Если при создании объекта произойдет ошибка, пользователь узнает о ней немедленно... а функции getCustomerInfo() отныне не нужно вызывать createRequest() — либо объект запроса готов к использованию, либо пользователь уже получил сообщение об ошибке.



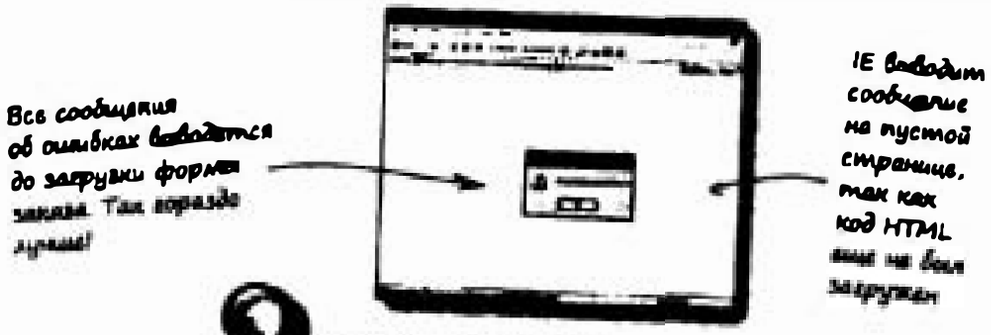
Просто сделайте это

Теперь, когда вы знаете про статический JavaScript, создание объектов запросов и расширенную обработку ошибок, в pizza.html можно внести некоторые изменения. Откройте файл HTML и вынесите весь код `createRequest()` за пределы функции. Вставьте его сразу же после строки JavaScript, которая выглядит так:

```
<script language="javascript" type="text/javascript">
 var request = null;

</script>
```

Весь код, который прежде находился в createRequest(), должен находиться здесь

Затем полностью удалите функцию `createRequest()`, а также исключите из `getCustomerInfo()` строку с вызовом `createRequest()`. Для проверки внесенных изменений попробуйте закомментировать код JavaScript, создающий объект запроса для используемого вами браузера. Все сообщения об ошибках должны выводиться печатленно — до того, как вы получите возможность работать с формой заказа Break Neck. Завершив тестирование, удалите все комментарии и убедитесь, что pizza.html работает как в браузерах Microsoft, так и в альтернативных браузерах. Сохраните изменения... а теперь можете перевернуть страницу.



ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

В: Значит, браузер выполнит весь код JavaScript, не входящий в функции, перед отображением HTML-кода страницы?

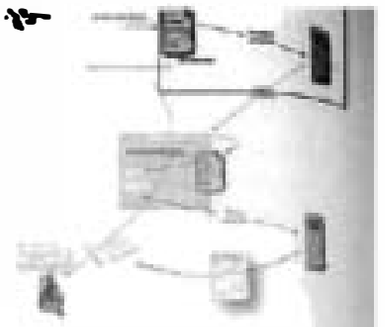
О: Весь код JavaScript, находящийся в секции <head> кода HTML, будет выполнен до загрузки страницы. Тем не менее код HTML может находиться в любом месте страницы... даже между элементами <p> и <form> в середине страницы. Код JavaScript в этих секциях выполняется в тот момент, когда браузер добирается до соответствующей части страницы. Но весь статический код JavaScript заведомо будет выполнен до того, как пользователь начнет работать со страницей; сейчас для нас важно именно это.

В: Нельзя ли еще раз объяснить, почему для работы с Break Neck в Internet Explorer необходимо использовать ActiveXObject?

О: Помните войны браузеров? В наши дни один объект в разных браузерах может называться разными именами. К счастью, Internet Explorer 7.0 должен перейти на стандартную схему имен, и объект ActiveXObject должен быть заменен новым объектом с именем XMLHttpRequest. Однако вам все равно придется поддерживать старые версии InternetExplorer, поэтому код почти не изменится.

Вернемся к getCustomerInfo()

Разобравшись с объектом запроса, вернемся к кодированию `getCustomerInfo()`. Помните, на чем мы остановились?



```
function getCustomerInfo() {
  var phone = document.getElementById("phone").value;
  createRequest();
  var url = "lookupCustomer.php?phone=" +
            escape(phone);
  request.open("GET", url, true);
  request.onreadystatechange = updatePage;
  request.send(null);
}
```

Эту строку убираем... теперь объект запроса создается в статическом коде JavaScript

Код, написанный на шаге 1

Следующее, что нам потребуется, — работающий сценарий PHP на сервере Head First Labs

Работа для серверных программистов

Вот что должны сделать для нас специалисты, занимающиеся программированием на стороне сервера:

1. Написать новый сценарий PHP для поиска адреса клиента по номеру телефона.
2. Присвоить сценарию имя lookupCustomer.php.
3. Выяснить, как сценарий должен получать телефон клиента из запроса.
4. Позаботиться о том, чтобы сценарий не возвращал код HTML... нам нужен только адрес клиента, и ничего более.

Нарисуйте линии, соединяющие каждую задачу с одним из пунктов диалога на следующей странице

Сценарий PHP

Давайте попросим Фрэнка из серверной группы написать нам сценарий PHP для поиска адреса клиента. Сценарий должен получать телефон клиента и возвращать ответ с адресом.

Просто сделайте это

Откройте каталог chapter12\start и найдите в нем файл lookupCustomer.php. Это сценарий PHP, работающий без сервера баз данных. Разместите файл на своем компьютере или отправьте на веб-сайт через FTP — вскоре мы будем использовать его.

PHP...на первый взгляд

Помимо: напишите этот сценарий во всех тонкостях не обязательно это дополнительный материал

Вот как выглядит сценарий, написанный Фрэнком для получения телефонного номера с формы заказа и поиска адреса клиента:

```
<?php

// Connect to database
$conn = @mysql_connect("mysql.headfirstlabs.com",
                       "secret", "really-secret");
if (!$conn)
  die("Error connecting to MySQL: " . mysql_error());

if (!mysql_select_db("headfirst", $conn))
  die("Error selecting Head First database: " . mysql_error());

$phone = preg_replace("/[\. \(\)\-]/", "", $_REQUEST['phone']);
$select = 'SELECT *';
$from   = '  FROM braj_breakneck';
$where  = ' WHERE phone = \'' . $phone' . '\'';

$queryResult = @mysql_query($select . $from . $where);
if (!$queryResult)
  die('Error retrieving customer from the database.');

while ($row = mysql_fetch_array($queryResult)) {
  echo $row['name'] . "\n" .
       $row['street1'] . "\n" .
       $row['city'] . ", " .
       $row['state'] . " " .
       $row['zipCode'];
}

mysql_close($conn);

?>
```

Стандартный код подключения к базе данных

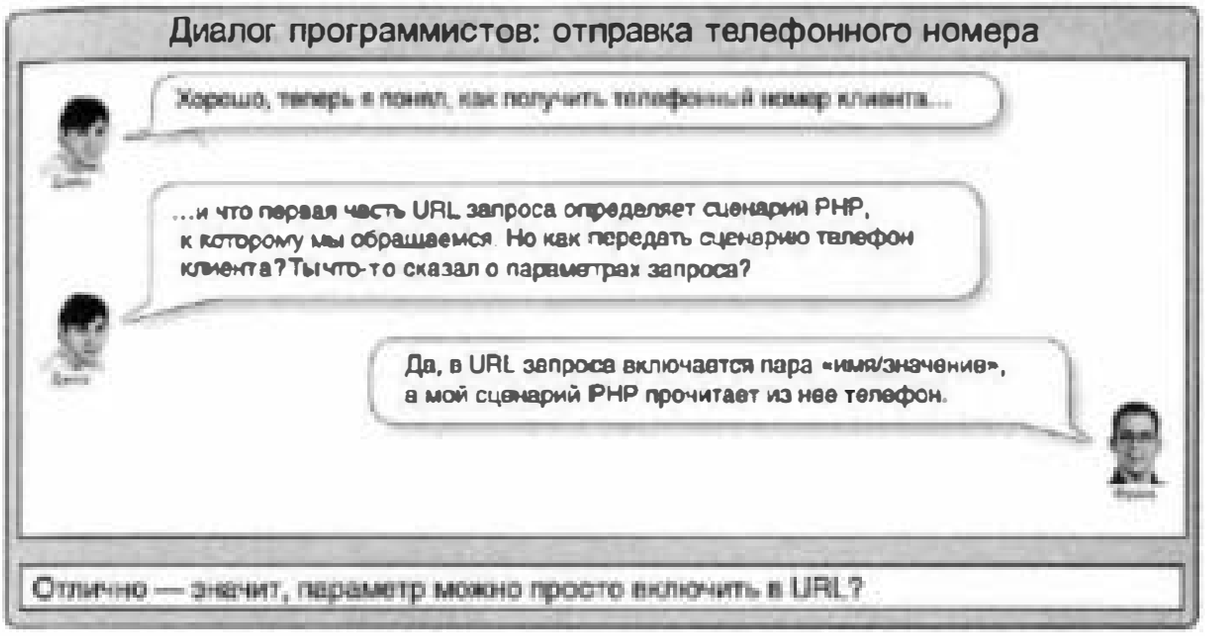
Этот фрагмент кода удаляет из телефонного номера все нежелательные символы (скобки, дефисы и пробелы)

По номеру телефона, переданному в составе запроса, сценарий ищет адрес клиента.

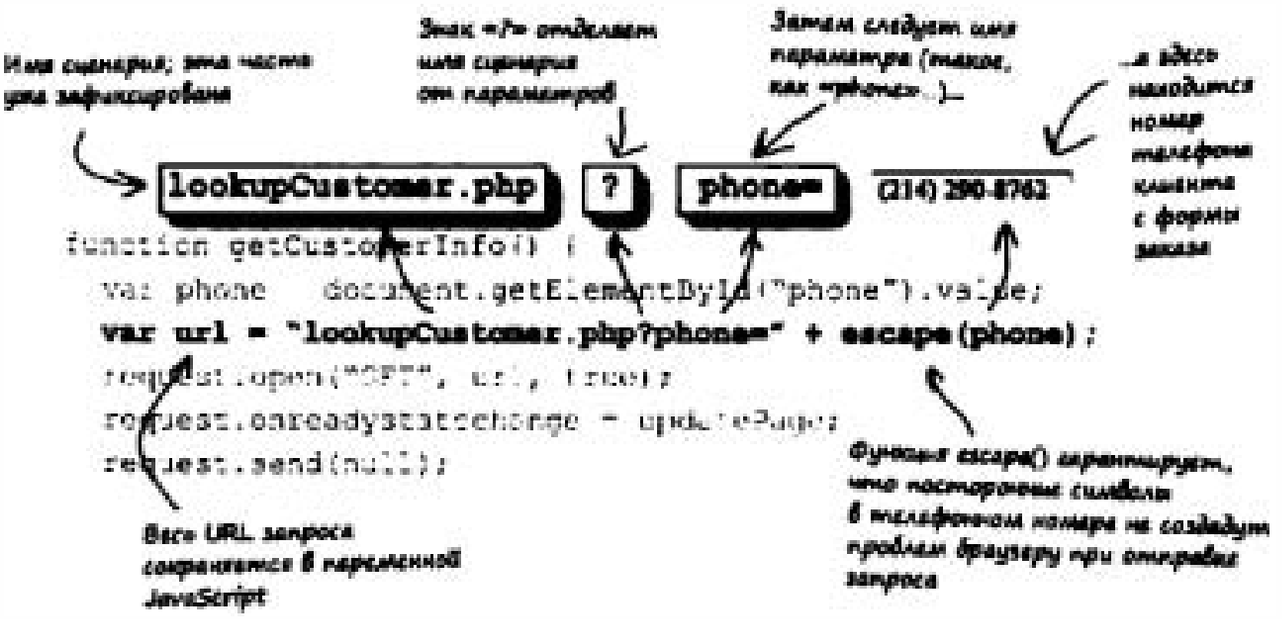
после чего возвращает адрес запрашивающей программе

* Версия lookupCustomer.php в примерах не использует базу данных и просто возвращает случайный адрес. Это сделано для того, чтобы пример Break Neck работал на вашем компьютере без сервера MySQL.

Диалог программистов: отправка телефонного номера

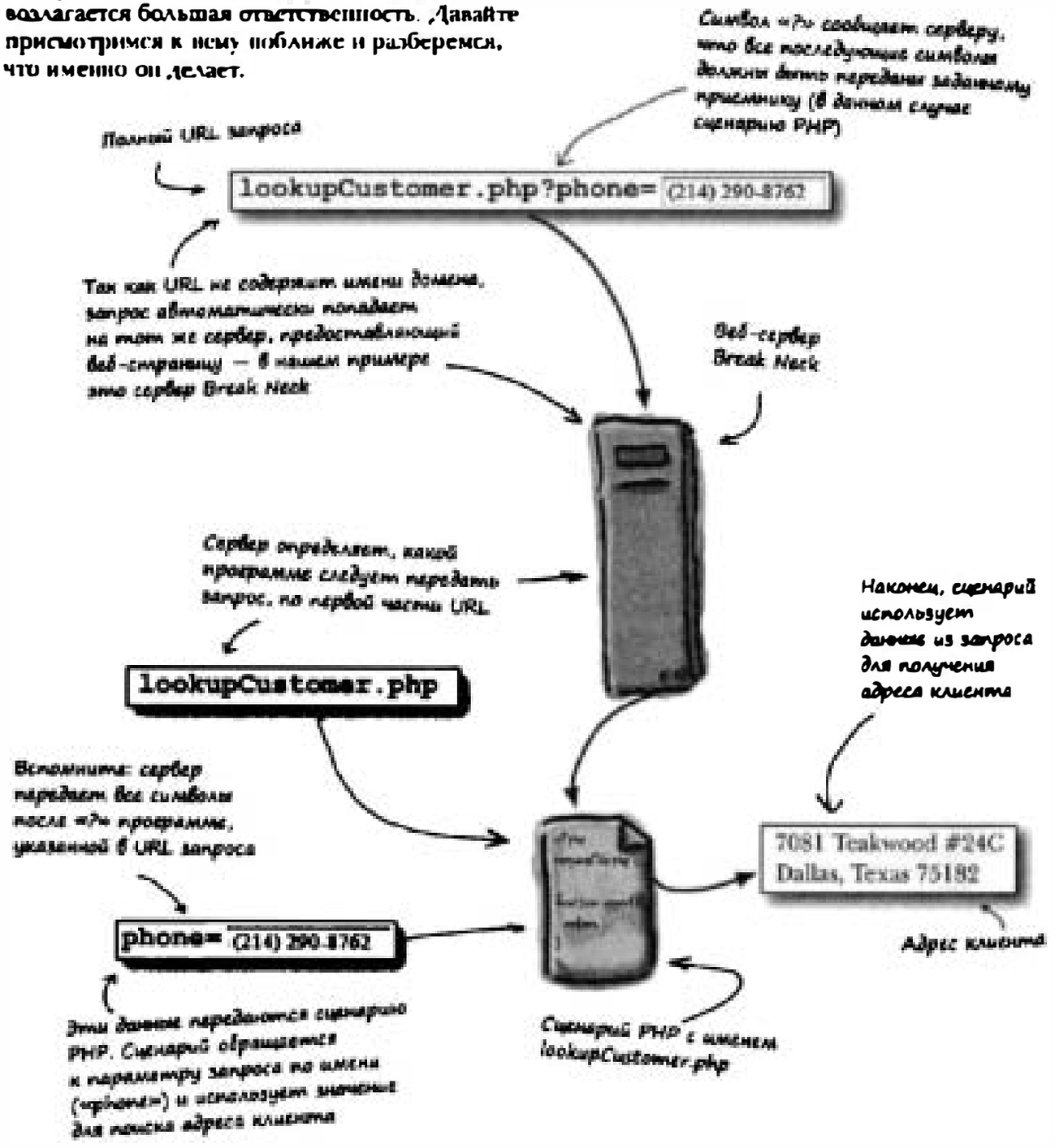


Отлично — значит, параметр можно просто включить в URL?



Передача данных серверу в URL запроса

На простой URL запроса в `getCustomerInfo()` возлагается большая ответственность. Давайте присмотримся к нему поближе и разберемся, что именно он делает.



Прости сделайте это

Зная URL запроса, мы можем инициализировать подключение при помощи метода **open()** объекта запроса. О том, как работает метод **open()**, было рассказано в главе 1. Подключение инициализируется следующим фрагментом кода JavaScript; каждый параметр приводится в отдельной строке. Напишите против каждого параметра, что он приказывает сделать объекту запроса.

```
request.open(
  "GET",
  url,
  true
);
```

Передача инструкций браузеру

Затем мы должны указать браузеру, что он должен делать при получении ответа от сервера. Не забудьте: браузер запускает указанную функцию *при каждом* изменении состояния готовности

```
function getCustomerInfo() {
  var phone = document.getElementById("phone").value;
  var url = "lookupCustomer.php?phone=" + escape(phone);
  request.open("GET", url, true);
  request.onreadystatechange = updatePage;
  request.send(null);
}
```

Имя функции, которая должна запускаться браузером при изменении состояния готовности объекта запроса

Свойство onreadystatechange должно задаваться перед вызовом send(); в противном случае браузер не будет знать, что ему делать с ответом

В: Если мы не вызываем `createRequest()` из метода `getCustomer()`, то как мы можем быть уверены в том, что объект запроса доступен?

О: Помните, что весь код JavaScript, создающий объект запроса, выполняется статически — до того, как браузер даст возможность пользователю приступить к вводу информации на форме заказа. При возникновении каких-либо проблем браузер выдаст сообщение об ошибке.

К моменту вызова `getCustomerInfo()` объект запроса уже создан и доступен через переменную `request` — или клиент уже знает о возникших проблемах. Следовательно, вам остается лишь использовать переменную `t` в своем коде.

В: Почему мы передаем номер телефона в URL? Разве нельзя воспользоваться методом `send()` для передачи телефона сценарию на сервере?

О: Несомненно, метод `send()` может использоваться для передачи данных серверу, но для этого необходимо отправить запрос методом POST, а также указать тип содержимого для передаваемых данных. Затем запрос переходит в формат пары «имя/значение» в методе `send()`. Если вам кажется, что это существенно увеличивает объем работы — вы абсолютно правы! Для формы Break Neck проще воспользоваться запросом GET с простым включением номера телефона в URL запроса. В этом случае нам не придется беспокоиться о запросах POST и типах содержимого. А поскольку номер телефона передается в составе URL запроса, мы передаем `null` при вызове `request.send()`, как это делалось в главе 1.

Запомните: запросы POST обычно лучше подходят для передачи конфиденциальной информации (скажем, номеров кредитных карт) или запросов, связанных с большим объемом данных. Мы еще вернемся к запросам POST и более интересным случаям использования `send()` при рассмотрении запросов XML и ответов в главе 5.

В: Нельзя ли чуть подробнее о тех «посторонних символах», которые необходимо исключить из параметра URL запроса?

О: Конечно. Возьмем типичный телефонный номер вида «(214)290-8762». Кроме цифр, дефиса и круглых скобок в него также входят пробелы. При передаче этих данных веб-серверу объект запроса использует URL запроса; он выглядит так же, как если бы URL был введен прямо в адресной строке браузера.

Но если попытаться включить пробелы в URL-адрес, введенный в браузере, вы либо получите сообщение об ошибке, либо часть URL будет проигнорирована. Аналогичная проблема также существует для объекта запроса. Функция JavaScript `escape()` решает ее, заменяя пробелы и другие «посторонние» символы специальными последовательностями, разрешенными в составе URL запроса. Например, функция `escape()` заменяет символ пробела последовательностью %20. Сценарии и серверные программы знают, что символы %20 необходимо преобразовать в пробел, поэтому данные будут переданы верно.

Отправка запроса серверу

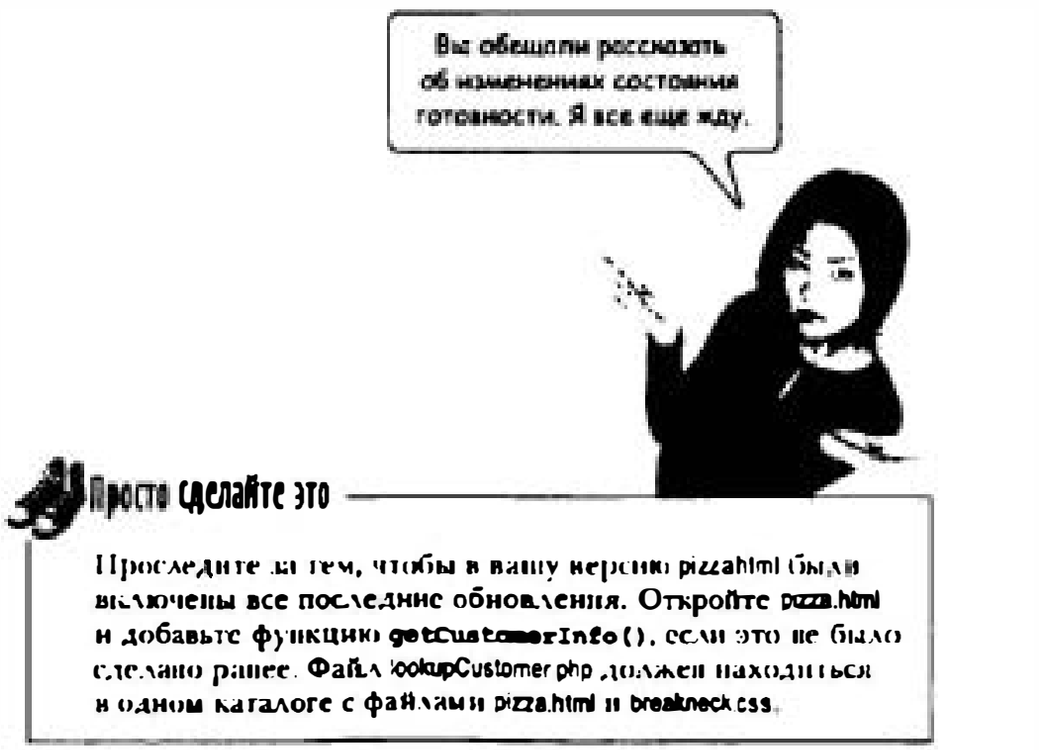
В завершение этого шага остается лишь передать запрос
Как было показано в главе 1, сделать это проще простого.
Взгляните на последнюю строку **getCustomerInfo()**:

```
function getCustomerInfo() {
  var phone = document.getElemer.tById("phone").value;
  var url = "lookupCustomer.ph.p?ph.one=" + escape(phone);
  request.open("GET", url, true);
  request.onreadystatechange = updatePage;
  [redacted]
}
```

Мы отправили телефон клиента в URL запроса, поэтому серверу не нужно пересылать дополнительные данные

Эта строка отправляет запрос на веб-сервер Break Neck

Проследите за тем, чтобы функция, запускаемая браузером, назначалась до вызова send()



Просто сделайте это

Проследите за тем, чтобы в вашу версию pizza.html были включены все последние обновления. Откройте pizza.html и добавьте функцию **getCustomerInfo()**, если это не было сделано ранее. Файл lookupCustomer.php должен находиться в одном каталоге с файлами pizza.html и breakneck.css.

Разговор в студии

Редакция: У нас в гостях неизменно популярный Браузер. Мы давно ждали этой встречи.

Браузер: Большое спасибо за приглашение.

Редакция: Для начала немного поговорим о запросах и состояниях готовности. В наше время по этой теме задают больше вопросов, чем о чем-либо еще.

Браузер: Ну конечно, я с радостью обсужу эту тему. Ведь состояния готовности — одна из немногих областей, в которых пользователи меня замечают. Обычно все говорят только о JavaScript и PHP.

Редакция: Что же, это очень важные языки программирования...

Браузер: Конечно, конечно, но какой прок от JavaScript без меня и функций обратного вызова? Без нас JavaScript — всего лишь набор строчек текста.

Редакция: Один момент... Обратный вызов? Что это? Нам этот термин еще не встречался

Браузер: Обратный вызов? Могу поспорить, что встречались, но просто не осознали этого. Знаете эту функцию, которую я должен запускать при изменениях готовности запросов?

Редакция: Ту, которая задается свойством onreadystatechange объекта запроса, верно?

Браузер: Точно. Это и есть функция обратного вызова. В сущности, функции обратного вызова составляют особую разновидность функций. Обнаружив, что с запросом что-то произошло, я передаю управление функции обратного вызова. Она позаботится обо всех деталях обработки ответа сервера.

Редакция: Значит, на передаче управления... эээ... функции обратного вызова, ваша работа завершается?

Браузер: Вовсе нет. Для типичного запроса мне приходится еще несколько раз обращаться к функция обратного вызова. Более того, я должен позаботиться о том, чтобы функция обратного вызова узнавала обо всем, что скажет сервер

Редакция: Да, через объект запроса.

Браузер: Абсолютно верно. В свойстве responseText я сообщаю функции обратного вызова, какая информация была получена от сервера. Я также присматриваю за свойством responseXML, но пожалуй, эта тема для другой главы...

Редакция: Да, позднее мы вернемся к этой теме. Итак, при запуске функция обратного вызова получает ответ сервера при помощи свойства responseText?

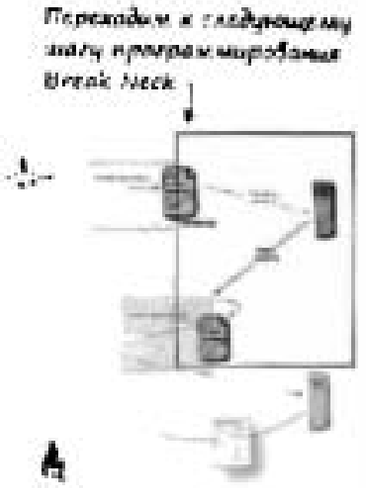
Браузер: Постойте... Вы кое-что забыли. Я вызываю функцию обратного вызова каждый раз, когда с запросом что-то происходит, а не только при получении ответа. Функция обратного вызова должна действовать разумно и не пытаться использовать объект запроса, не убедившись в завершении его обработки. В противном случае беды не миновать.

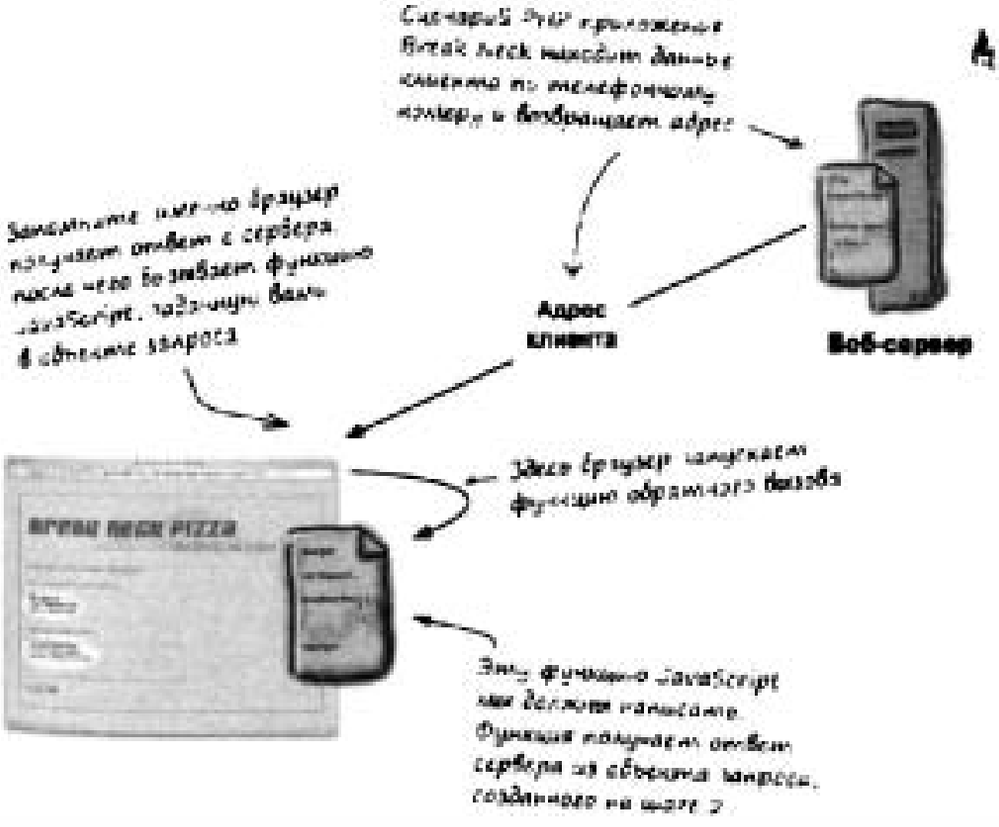
Редакция: Потому что если сервер не завершил обработку запроса...

Браузер: ...То я не помещу ответ сервера в объект. Кстати, мне как раз нужно обслужить изменяющееся состояние готовности. Бегу... До встречи!

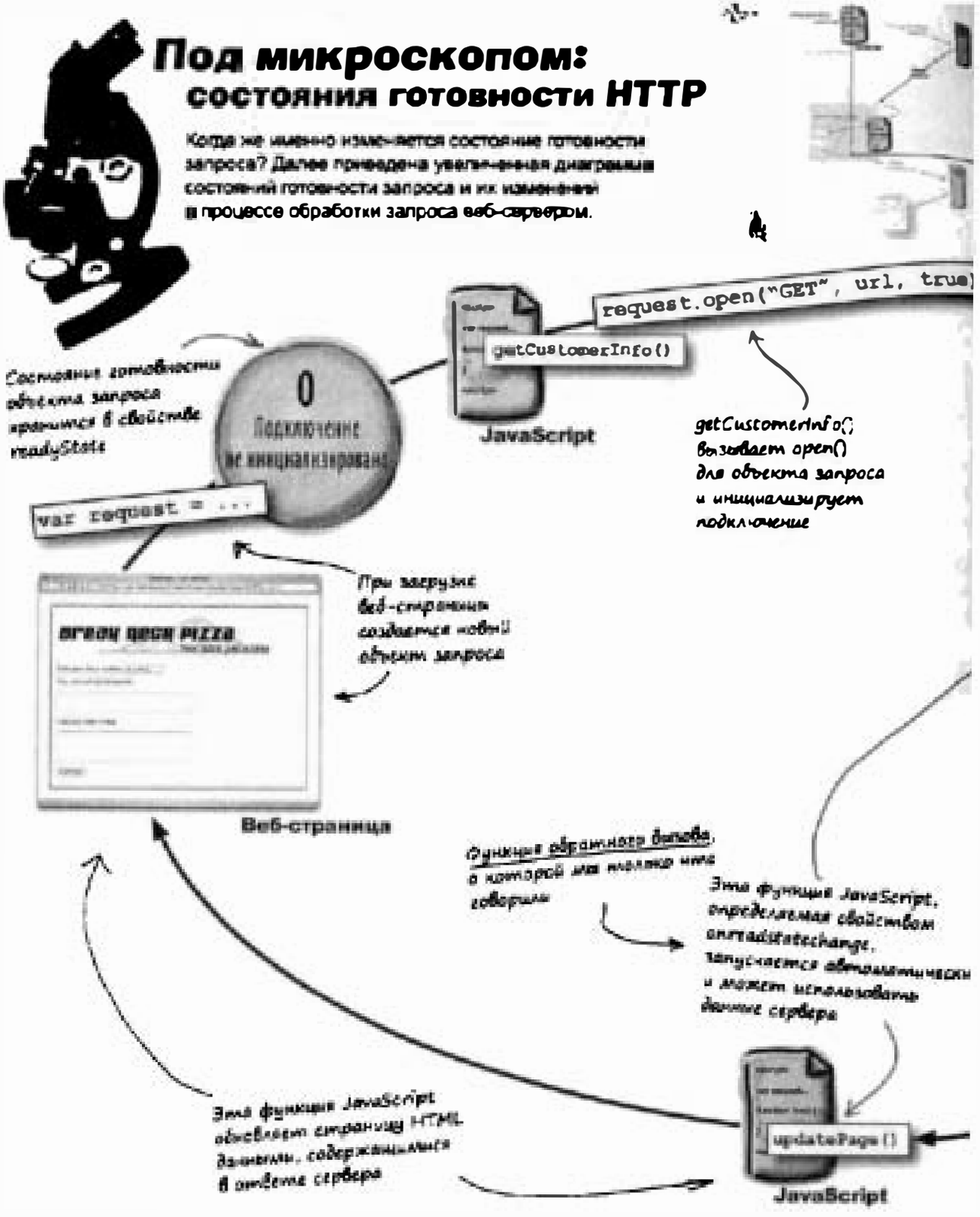
Шаг 3: Получение адреса клиента

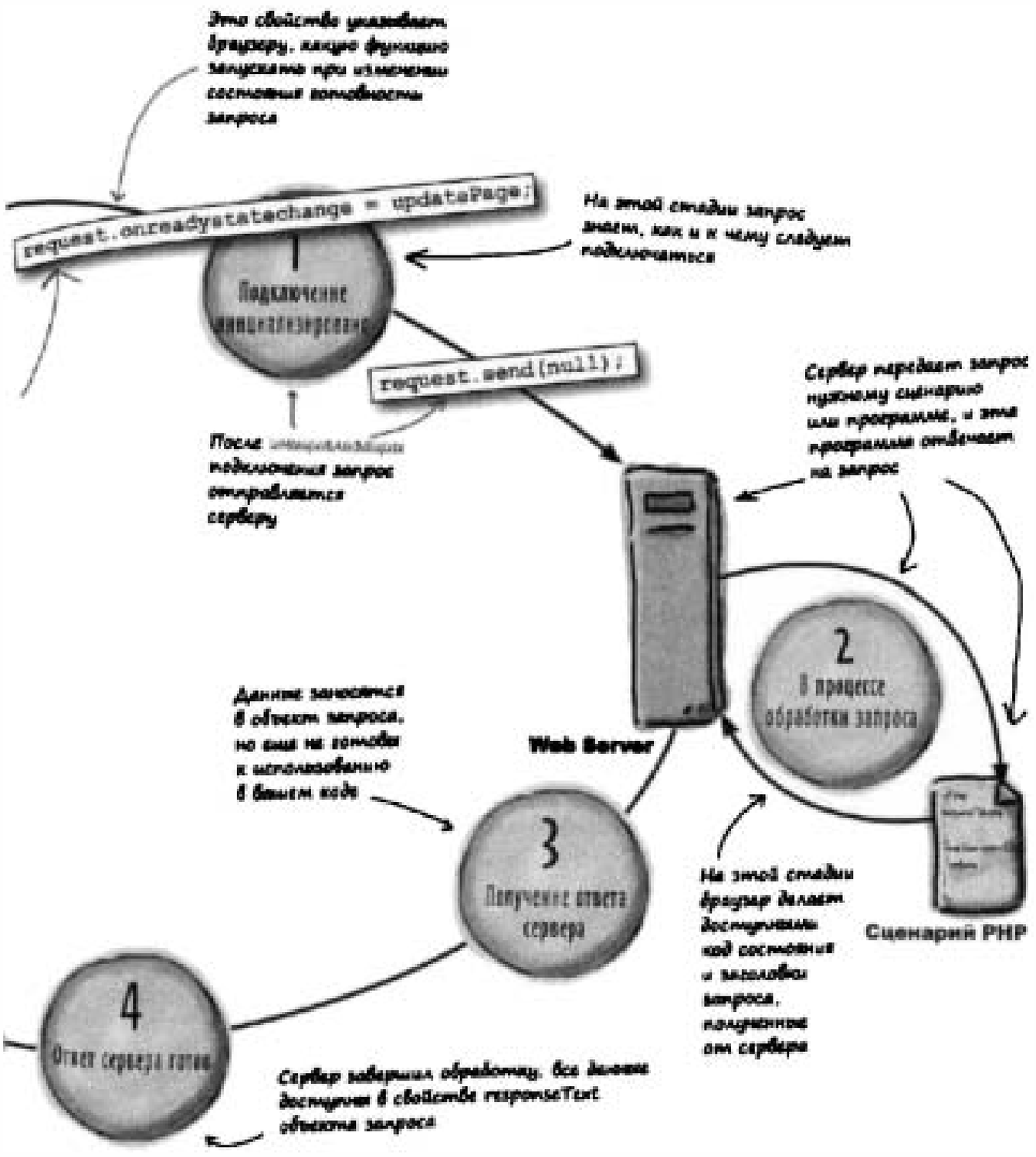
На этом шаге мы используем адрес, который возвращается сценарием PHP в ответ на запрос, отправленный на шаге 2. Получив ответ сервера Head First Pizza, браузер запускает функцию JavaScript и передает ей адрес, полученный от сценария.

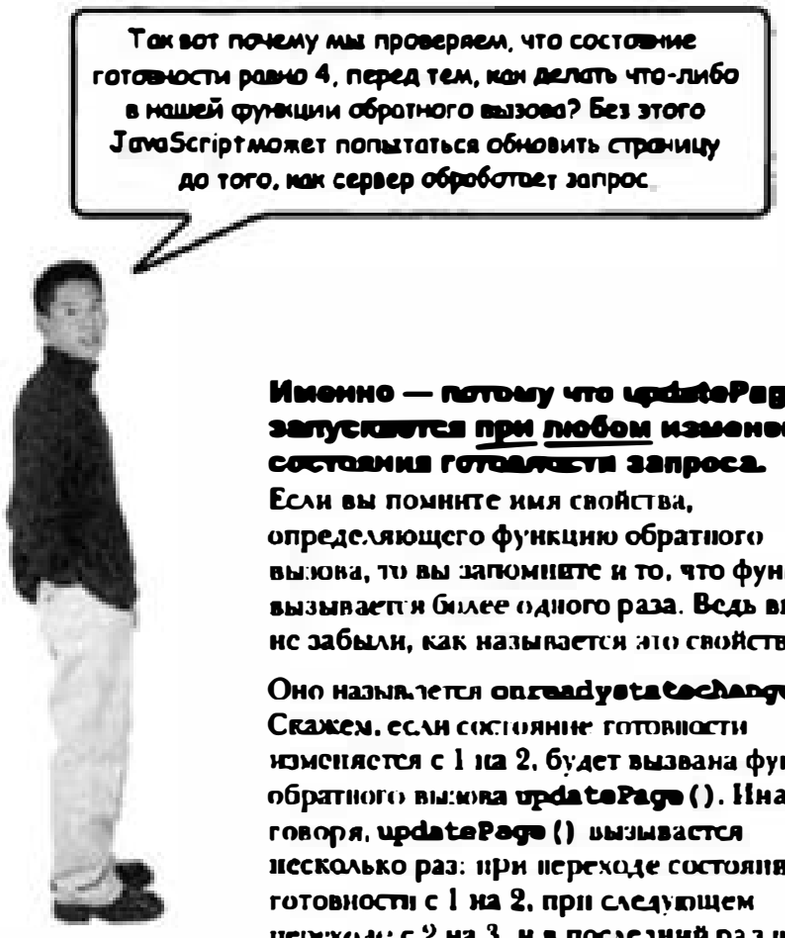




Браузер запускает функцию обратного вызова при каждом изменении состояния готовности объекта запроса.


Под микроскопом:
состояния готовности HTTP
Когда же именно изменяется состояние готовности запроса? Далее приведена увеличенная диаграмма состояний готовности запроса и их изменения в процессе обработки запроса веб-сервером.
request.open("GET", url, true)
getCustomerInfo()
JavaScript
getCustomerInfo() вызывает open() для объекта запроса и инициализирует подключение
Состояние готовности объекта запроса хранится в свойстве readyState
0
Подключение не инициализировано
var request = ...
При загрузке веб-страницы создается новый объект запроса
Веб-страница
Функция обратного вызова, в которой мы можем что говорили
Эта функция JavaScript, определяемая свойством onreadystatechange, запускается автоматически и может использовать данные сервера
updatePage()
JavaScript
Эта функция JavaScript обновляет страницу HTML данными, содержащимися в ответе сервера


Это свойство указывает браузеру, какую функцию запускать при изменении состояния готовности запроса
request.onreadystatechange = updatePage;
На этой стадии запрос знает, как и к чему следует подключаться
1
Подключение инициализировано
request.send(null);
После [illegible] подключения запрос отправляется серверу
Сервер передает запрос нужному сценарию или программе, и эта программа отвечает на запрос
2
В процессе обработки запроса
Web Server
Данные записаны в объект запроса, но еще не готовы к использованию в вашем коде
3
Получение ответа сервера
На этой стадии браузер делает доступными код состояния и заголовки запроса, полученные от сервера
Сценарий PHP
4
Ответ сервера готов
Сервер завершил обработку. Все данные доступны в свойстве responseText объекта запроса



Именно — потому что updatePage() запускается при любом изменении состояния готовности запроса.

Если вы помните имя свойства, определяющего функцию обратного вызова, то вы запомните и то, что функция вызывается более одного раза. Ведь вы еще не забыли, как называется это свойство?

Оно называется `onreadystatechange`. Скажем, если состояние готовности изменяется с 1 на 2, будет вызвана функция обратного вызова `updatePage()`. Иначе говоря, `updatePage()` вызывается несколько раз: при переходе состояния готовности с 1 на 2, при следующем переходе с 2 на 3, и в последний раз при переходе состояния готовности с 3 на 4.

Однако сервер гарантирует наличие данных, пригодных для использования, только в состоянии готовности 4. А это означает, что текущее состояние готовности должно проверяться перед обновлением формы заказа, потому что в противном случае страница может содержать неполные или недействительные данные.

Проверка состояния готовности

Итак, функция **getCustomerInfo()** работает, а браузер знает, что функция **updatePage()** должна вызываться при изменении состояния готовности запроса. Пришло время написать функцию обратного вызова для приложения Break Neck. Прежде чем делать что-либо с кодом HTML, мы проверим состояние готовности и убедимся в том, что обработка запроса завершилась.

Включите в файл pizza.html новую функцию с именем **updatePage()**. Начните со следующего кода:

Имя этой функции ДОЛЖНО совпадать с именем функции, заданным свойству onreadystatechange в getCustomerInfo()

Команда if гарантирует, что остальной код функции будет выполняться лишь в состоянии готовности 4 (то есть когда сервер завершил обработку, а данные запроса можно безопасно использовать)

```
function updatePage() {
  if (request.readyState == 4) {
    /* Get the response from the server */
    /* Update the order form */
  }
}
```

Переменная request объявлена в статическом коде JavaScript, поэтому она может использоваться в любой функции

Мы напишем код, решающий обе задачи, на нескольких ближайших страницах

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

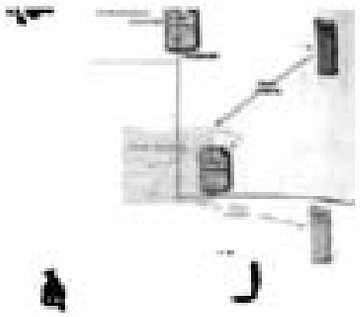
В: Как сервер запускает функцию обратного вызова при изменении состояния готовности? Я не знал, что сервер может взять и запустить код JavaScript в странице HTML.

О: Верно... На самом деле функцию обратного вызова запускает браузер. Завершив обработку запроса, сервер сообщает об этом браузеру. На этой стадии работа сервера закончена, а браузер должен разобраться, что делать дальше. Для этого он смотрит, какая функция указана в свойстве onreadystatechange объекта запроса, и вызывает эту функцию. Таким образом, код JavaScript запускается браузером, а не сервером.

В: Нам когда-нибудь придется писать код, работающий в других состояниях готовности (кроме 4)?

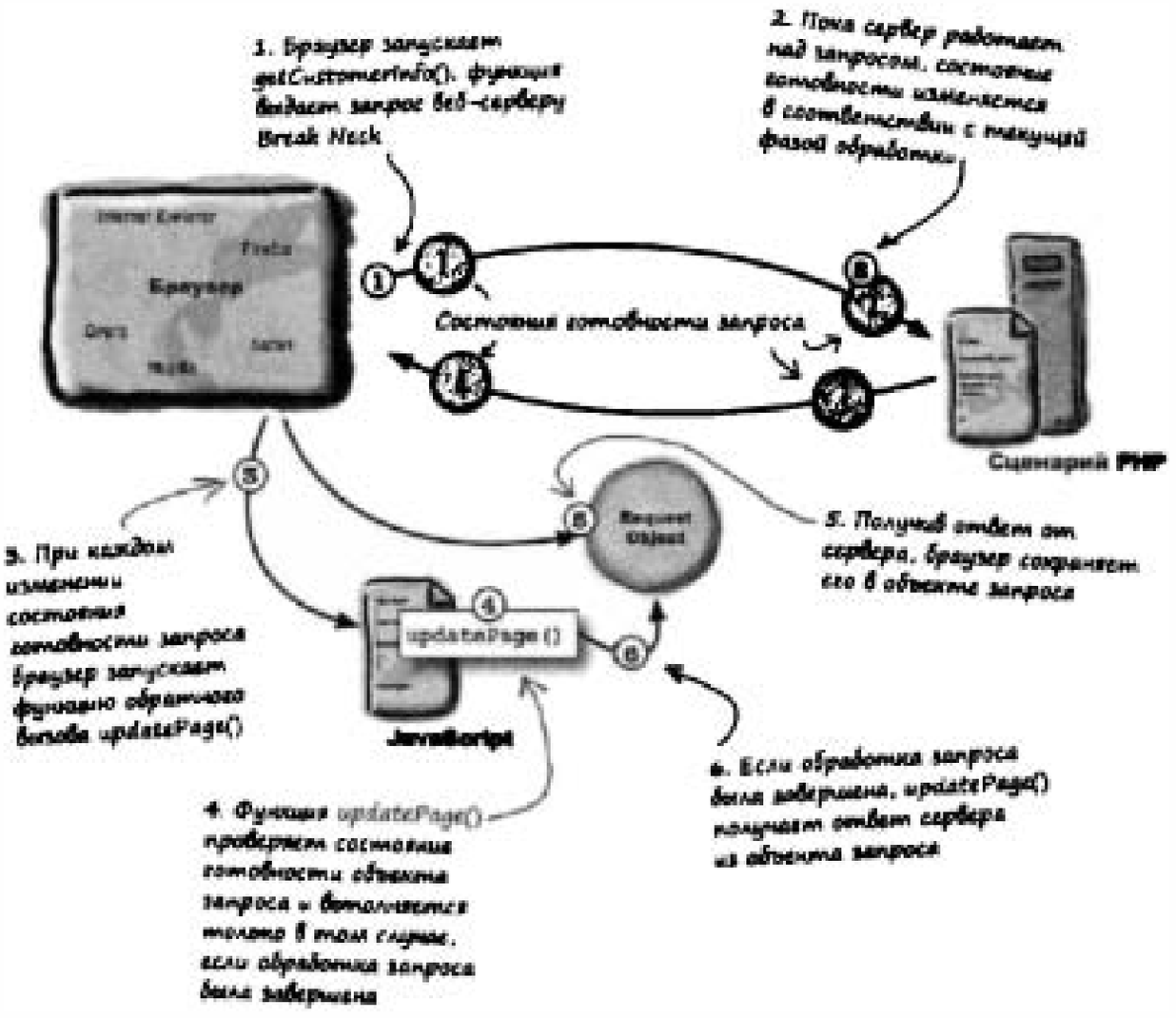
О: Весьма редко. Речь идет об асинхронных запросах, пользователи не ждут реакции сервера и им не обязательно знать, в каком состоянии находится запрос. Обычно функция обратного вызова программируется так, чтобы она выполняла свои действия только при завершении обработки запроса сервером — то есть в состоянии готовности 4.

Что делает браузер?



Итак, мы уже видели, что делает сервер Break Neck, и написали немало кода JavaScript. Но что делает браузер при изменении состояний готовности?

Давайте посмотрим.



Браузер предоставляет доступ к ответу сервера через объект запроса JavaScript.

Получение ответа сервера из объекта запроса

Если состояние готовности равно 4, значит, браузер уже поместил ответ запроса в свойство **responseText** объекта запроса.

```
function updatePage() {
  if (request.readyState == 4) {
    /* Get the response from the server */
    var customerAddress = request.responseText;

    /* Update the HTML web form */
  }
}
```

Сервер возвращает адрес клиента

Браузер сохраняет ответ запроса в свойстве responseText

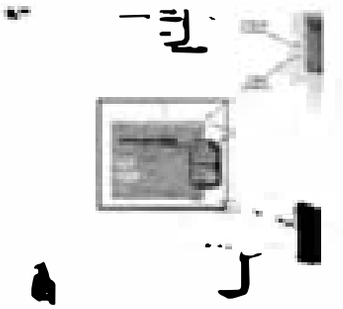
Возвращаемся к исходным планам

Вы еще не забыли структурную диаграмму и пометки, сделанные вами для приложения Break Neck? Вернитесь к с. 73 и проверьте, соответствуют ли ваши пометки тому, что мы делали до сих пор. У вас еще остается последняя возможность внести изменения перед тем, как мы выйдем на финишную прямую.

Напишите внизу, что, по вашему мнению, осталось сделать для завершения приложения Break Neck.

Шаг 4: Обновление формы заказа

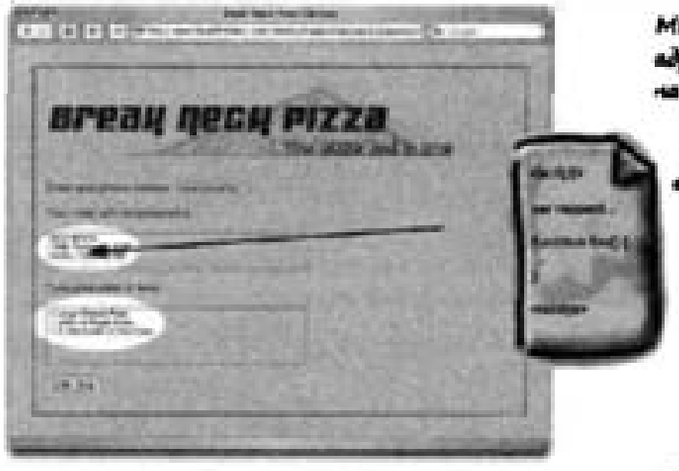
Получив адрес клиента, мы должны обновить форму заказа пиццы. В этом нам снова поможет браузер и модель DOM.



Работа над приложением Break Neck близится к концу!

Функция обратного вызова JavaScript обновляет форму заказа при помощи браузера и модели DOM

Если внести изменения в веб-страницу средствами DOM, браузер обновит страницу немедленно.



Мы уже получили адрес клиента на шаге 3...

...и можем обновить веб-форму средствами DOM (по аналогии с тем, как ранее мы читали с формы номер телефона)

Завершение функции обратного вызова

После получения адреса с сервера остается лишь обновить веб-форму. Поскольку адрес хранится в поле формы, мы снова можем воспользоваться моделью DOM. Обновление напоминает чтение поля с номером телефона в функции **getCustomerInfo()** (с. 82):

```
function updatePage() {
  if (request.readyState == 4) {
    /* Get the response from the server */
    var customerAddress = request.responseText;

    /* Update the HTML web form */
    document.getElementById("address").value =
      customerAddress;
  }
}
```

Адрес хранится в поле формы, поэтому для работы с его текстовым представлением можно использовать свойство value

Нам снова пригодились атрибуты id

Простозанесите адрес клиента в свойство value поля

Мы снова используем DOM. На этот раз для записи данных на веб-форму

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

В: Почему для обновления поля адреса не использовался вспомогательный файл JavaScript из главы 1?

О: В главе 1 мы обновляли текст в элементе . Однако не является элементом HTML, в котором обычно вводится текст, и поэтому не обладает свойством value; это относится и к таким элементам HTML, как <p>, и .

С другой стороны, поля форм обычно предназначены для ввода данных пользователем, поэтому для простоты мы можем воспользоваться свойством value и читать записывать текстовое содержимое поля напрямую.

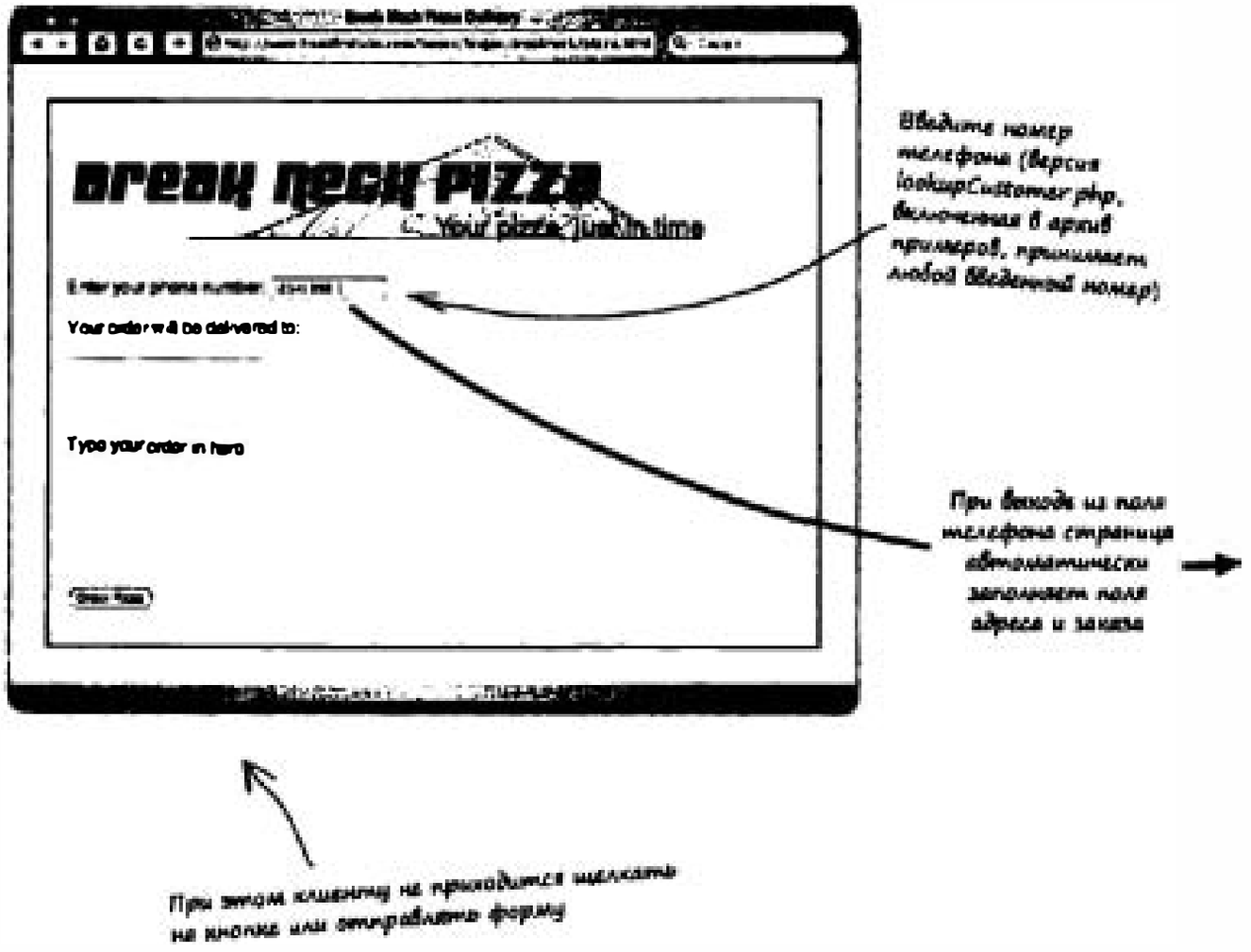
В: Почему мы не использовали для адреса клиента? Почему именно поле?

О: Хотя сервер Breakfast Finder может адрес клиента автоматически, нельзя исключать, что клиент захочет заказать пиццу на другой адрес (скажем, в клуб). Хотя адрес вводится автоматически, клиент при желании может ввести другой адрес.

Элементы , и обновление HTML будут подробно рассматриваться в главе 4.

Проверяем приложение Break Neck

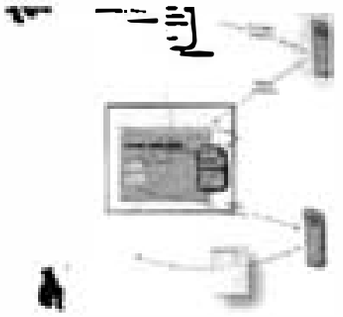
Убедитесь в том, что весь описанный код JavaScript был включен в файл pizza.htm. Откройте страницу в браузере и введите телефон. Похоже, все работает! Сервер отвечает на запрос и заполняет форму адресом, взятым из базы данных клиентов.

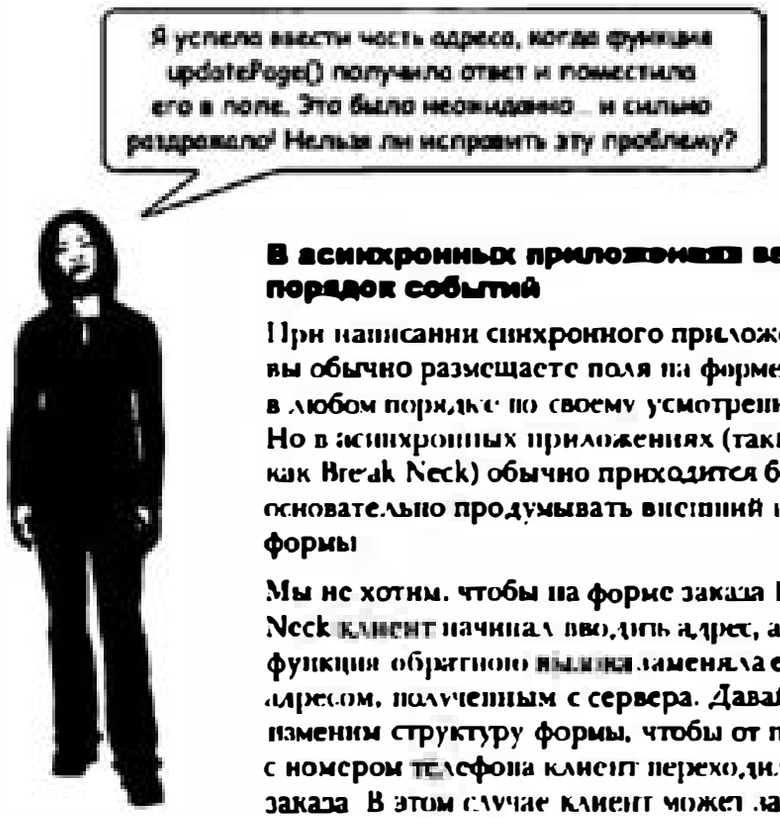


Поле заполняется адресом клиентом, найденным сценарием PHP Break Neck

Клиент продолжает работу над заказом, пока сервер производит выборку адреса

Секундочку...





В асинхронных приложениях важен порядок событий

При написании синхронного приложения вы обычно размещаете поля на форме в любом порядке по своему усмотрению. Но в асинхронных приложениях (таких, как Break Neck) обычно приходится более основательно продумывать внешний вид формы.

Мы не хотим, чтобы на форме заказа Break Neck клиент начинал вводить адрес, а потом функция обратного вызова заменяла его адресом, полученным с сервера. Давайте изменим структуру формы, чтобы от поля с номером телефона клиент переходил к полю заказа. В этом случае клиент может заняться вводом описания пиццы, пока сервер ищет его адрес в базе данных.

А как насчет проблемы Windows, упоминавшейся в главе 1? Разве приложение не нужно исправить так, чтобы оно работало на Mac и PC?

Что происходит в Windows?

Браузер Internet Explorer достаточно умен; он пытается сделать многое для того, чтобы вам работалось быстрее и удобнее. Например, он кэширует графику, чтобы ускорить загрузку при повторных посещениях страниц с большим количеством графики.

IE пытается проделывать нечто подобное и с URL. Если вы обращаетесь с запросом к серверной программе, IE запоминает запрашиваемый URL. Если после этого запросить тот же URL (без изменений в данных), IE считает, что вы получите тот же ответ. Вместо повторной отправки запроса браузер просто возвращает результат, полученный при первом запросе.

Раз уж мы впервые столкнулись с этой проблемой в приложении Boards, давайте начнем с исправления этого приложения для Windows. Затем полученный опыт будет применен для исправления Break Neck Pizza.

Браузер Opera в подобной ситуации делает то же самое: он кэширует URL.

ШЕВЕЛИМ МОЗГАМИ

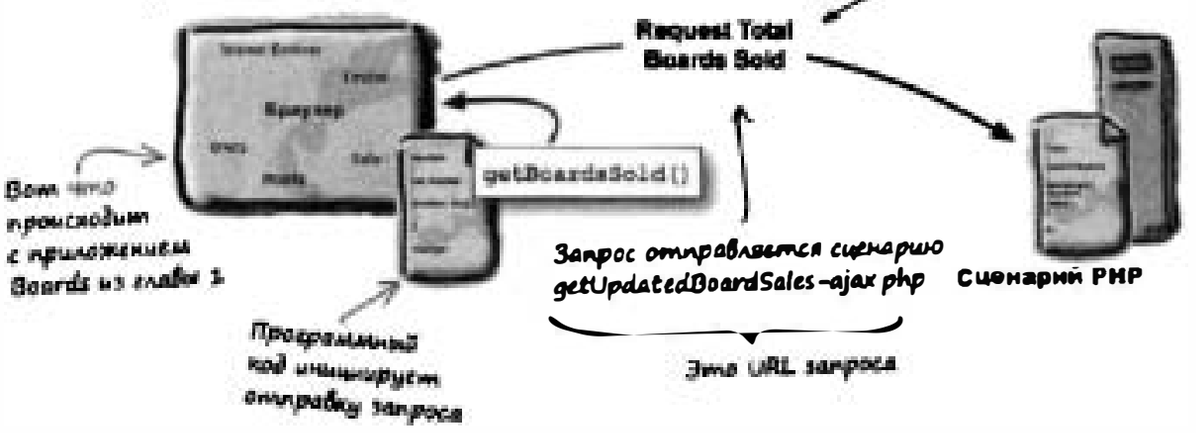
Почему кэширование запросов в Internet Explorer и Opera создает проблемы в приложении Boards? Сохраняется ли проблемы и в форме заказа пиццы Break Neck? И что нужно сделать для их решения?

Когда браузеры кэшируют URL запросов...

Давайте повнимательнее разберемся, что делают такие браузеры, как Internet Explorer и Opera, и почему их «помощь» создает проблемы в асинхронных приложениях.

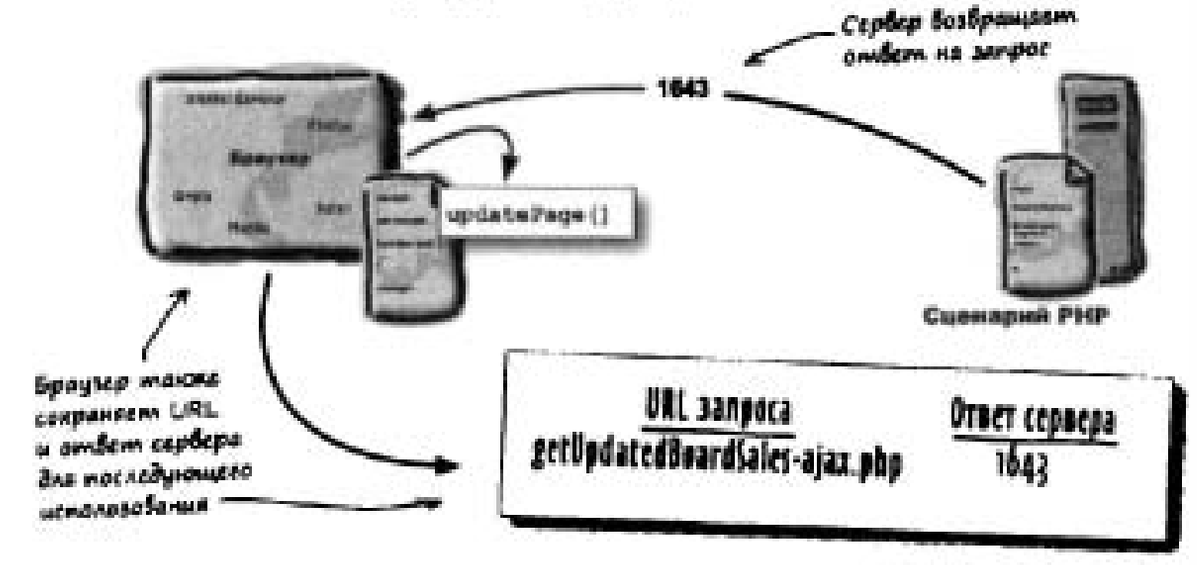
Программный код выдает запрос веб-серверу

Работа большинства приложений Ajax начинается с запуска функции JavaScript при выполнении некоторого условия (скажем, при вводе телефонного номера). Код JavaScript строит URL и отправляет по нему запрос.



Сервер возвращает ответ

При получении ответа от сервера браузер запускает заданную функцию обратного вызова. Но если браузер кэширует URL запросов, он запоминает URL и ответ сервера для последующего использования.



Пока все идет замечательно. Все вас любят и считают гением

Мы создали динамическое приложение

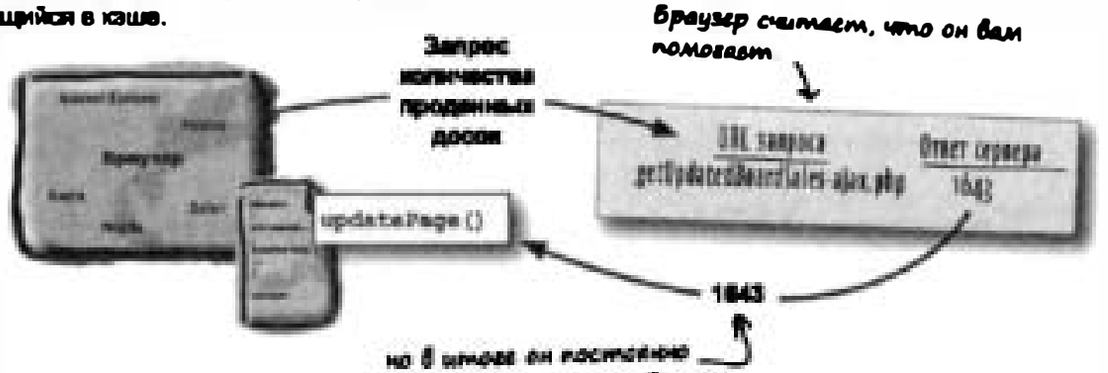
Функция обратного вызова JavaScript обновляет веб-страницу новыми данными, полученными из ответа сервера, причем для этого не нужно ни отправлять формы, ни перерисовывать страницу.

● Программный код выдает следующий запрос к веб-серверу

Ваше приложение всем понравилось, и кто-то захочет воспользоваться им повторно. Допустим, в приложении Boards Кэти щелкает на кнопке Show Me the Money во второй раз

● Браузер возвращает ответ программному коду

Браузер видит, что у него уже есть ответ для указанного URL запроса, и решает сэкономить немного времени. Вместо того чтобы отправить запрос серверу, он выдает ответ, хранящийся в кэше.

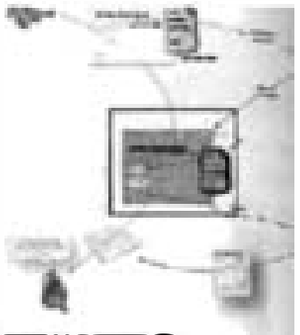


Теперь вас ненавидят. Пользователи считают, что вы идиот, а Ajax — пустая трата времени

Что за дурацкое приложение!

Внезапно динамическое приложение начинает снова и снова возвращать одни и те же данные. Функция обратного вызова получает старые данные, а веб-страница не обновляется свежей информацией

Чтобы обойти механизм кэширования, мы должны каждый раз изменять URL. Но если мы обратимся к одному сценарию и не передадим данные в приложении Boards, как URL могут быть разными?



Иногда приходится действовать нестандартно...

Браузер не интересует, чем различаются URL запроса; для него важно лишь то, чтобы они различались. Следовательно, мы можем просто включить в URL фиктивный параметр и присваивать ему разные значения при каждой отправке запроса. А чтобы не писать генератор случайных чисел или выполнять иную лишнюю работы, мы просто возьмем текущее время (в секундах) и передадим его в URL запроса.

Давайте посмотрим, как исправить функцию **getBoardsSold()** из главы 1:

Помните древние времена, когда объект запроса создавался в функции?

```
function getBoardsSold() {
  createRequest();
  var url = "getUpdatedBoardSales-ajax.php";
  [redacted]
  request.open("GET", url, true);
  request.onreadystatechange = updatePage;
  request.send(null);
}
```

Первая строка задает нормальный URL, а вторая добавляет к нему фиктивный параметр

Значением фиктивного параметра является текущее время

Теперь URL запроса изменяется...

getUpdatedBoardSales-ajax.php?dummy=1139262723388

getUpdatedBoardSales-ajax.php?dummy=1139262774440

getUpdatedBoardSales-ajax.php?dummy=1139262797519

Двух одинаковых URL не будет

...потому что каждый момент построения URL хотя бы незначительно отличается от других

Просто сделайте это

Вернитесь к своей копии boards.html из примеров главы 1 и обновите функцию **getBoardsOld()**. Опробуйте новую версию в Internet Explorer для Windows, или в Opera (для Mac или Windows). Останется ли проблема кэширования в новой версии?

Часто задаваемые вопросы

В: Нельзя ли отключить кэширование в браузере и обойтись без лишнего кода?

О: Нет. Большинство параметров в настройках браузера управляют кэшированием страниц, загружаемых напрямую — вводом URL в строке адреса или щелчком на гиперссылке. IE и Opera не позволяют управлять обработкой запросов, сгенерированных кодом JavaScript.

Впрочем, даже если бы кэширование удалось отключить, у многих пользователей оно наверняка включено. Поэтому нам в любом случае придется искать обходные пути.

В: Значит, добавление параметра фиктивному позволяет обойти кэширование?

О: Дело не в параметре, и не в его имени. Нам важно изменить сам URL запроса. Мы использовали имя dummy только по одной причине: чтобы с первого взгляда было ясно, что параметр не содержит данных, существенных для приложения. Вы можете присвоить фиктивному параметру любое имя по своему усмотрению.

В: Не останется ли проблемы при отправке двух запросов за одну минуту? Ведь время в обоих случаях будет одинаковым, верно?

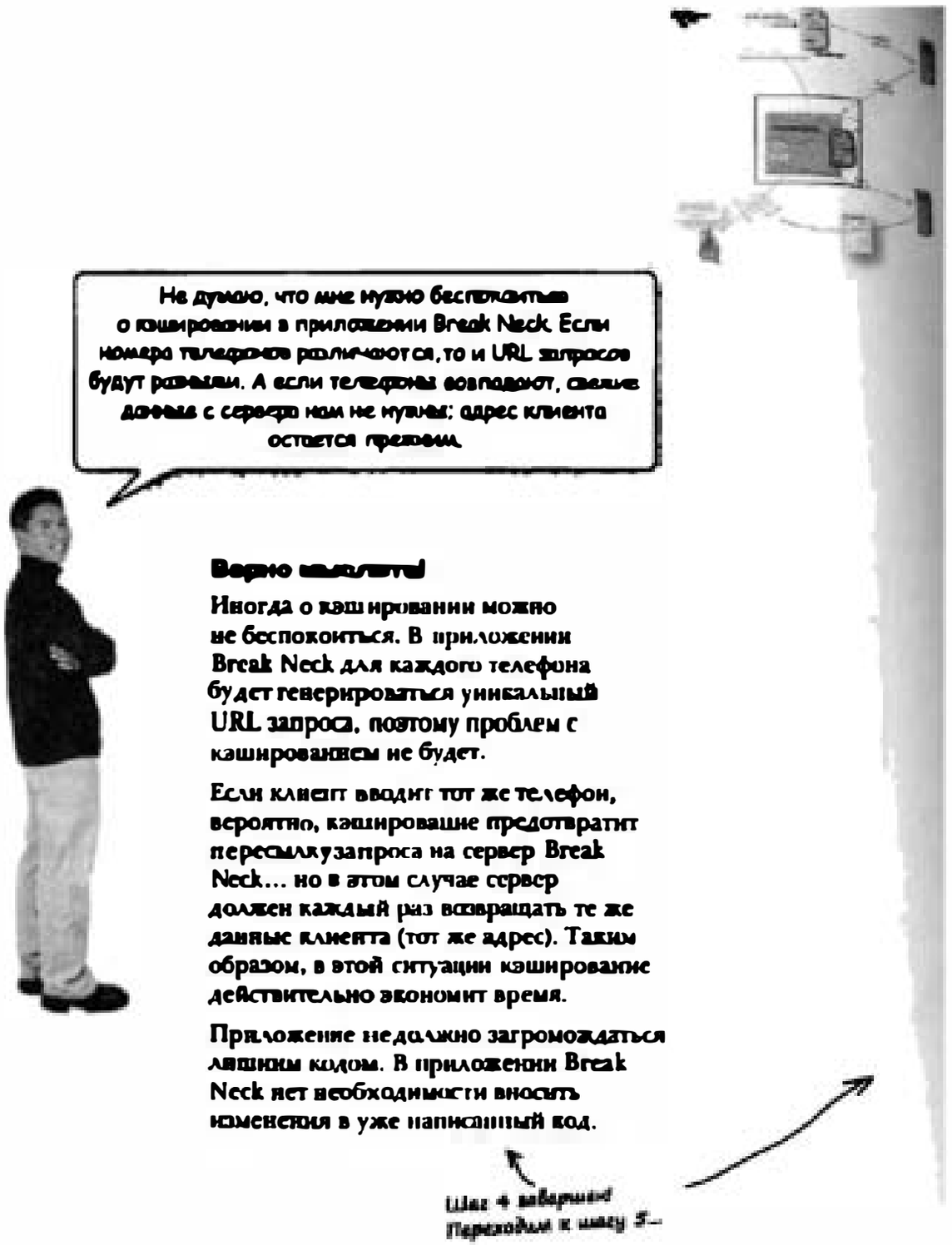
О: Нет, функция getTime() возвращает время в миллисекундах, прошедших с 1 января 1970 года. Если только вам не удастся щелкнуть на кнопке дважды за одну миллисекунду, URL запросов будут разными.

В: Что делает сценарий PHP с фиктивным параметром при получении запроса?

О: Ничего. Этот параметр сценарию не нужен, поэтому он просто игнорируется.

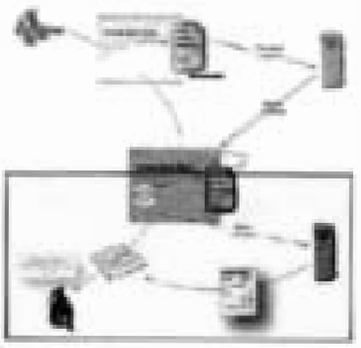
В: Значит, нам остается вместе со всеми изменениями в функцию getCustomerInfo() в файле pizza.html?

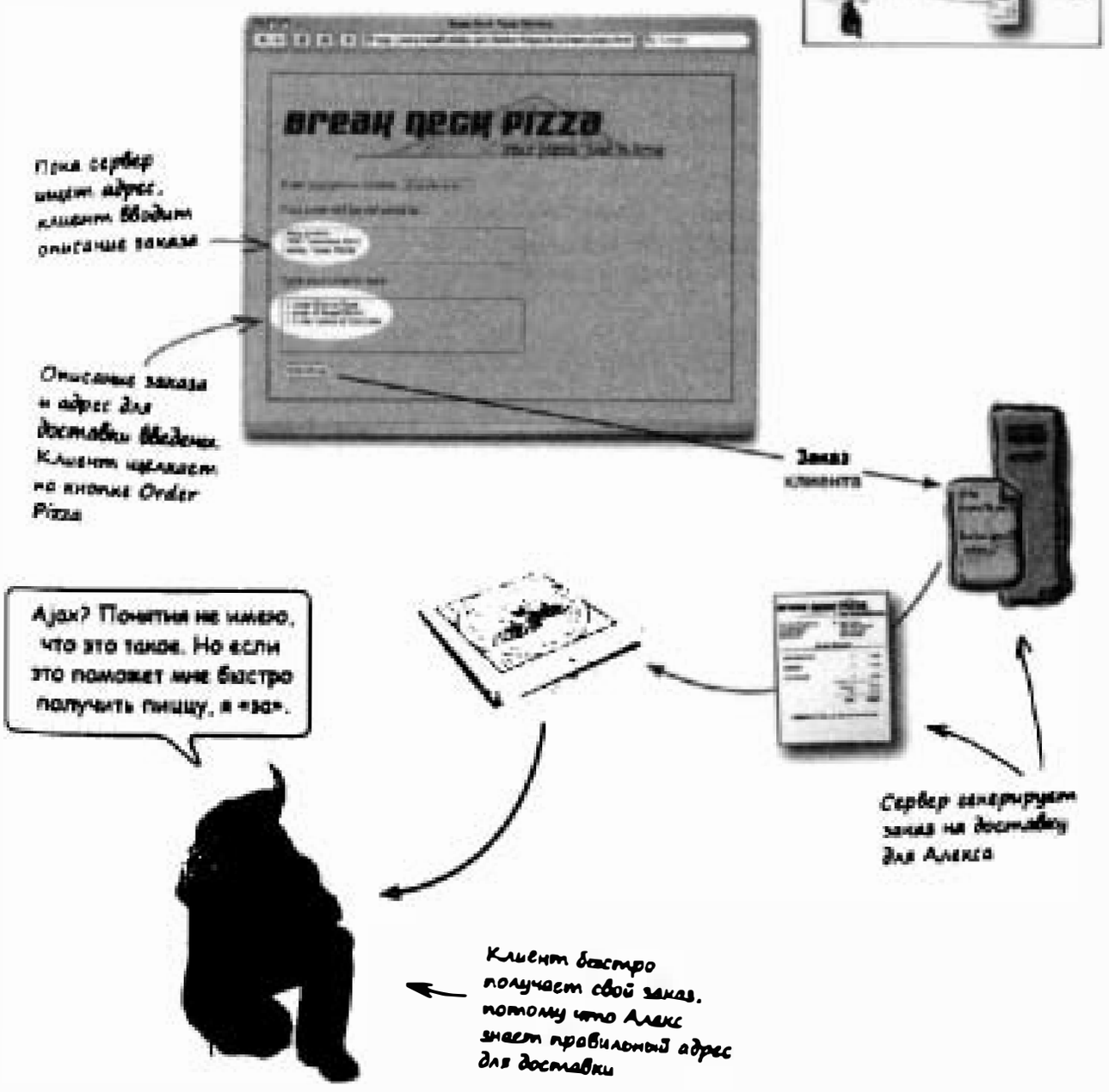
О: Хорошая идея... Или нет? Перечитайте страницу, и мы поговорим об этом подробнее.


Не думаю, что мне нужно беспокоиться о кешировании в приложении Break Neck. Если номера телефонов различаются, то и URL запросов будут разными. А если телефоны совпадают, свежие данные с сервера нам не нужны; адрес клиента остается прежним.
Верно мыслите!
Иногда о кешировании можно не беспокоиться. В приложении Break Neck для каждого телефона будет генерироваться уникальный URL запроса, поэтому проблем с кешированием не будет.
Если клиент вводит тот же телефон, вероятно, кеширование предотвратит пересылку запроса на сервер Break Neck... но в этом случае сервер должен каждый раз возвращать те же данные клиента (тот же адрес). Таким образом, в этой ситуации кеширование действительно экономит время.
Приложение не должно загромождаться лишним кодом. В приложении Break Neck нет необходимости вносить изменения в уже написанный код.
Шаг 4 выполнен! Переходим к шагу 5...

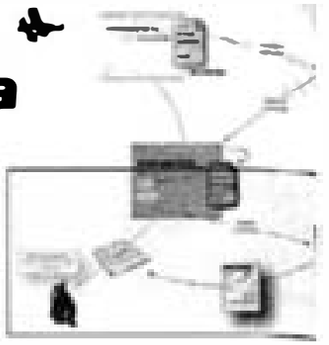
Шаг 5: Регистрация заказа

Итак, клиент ввел описание заказа на форме; его адрес был автоматически поставлен на форму функцией JavaScript, написанной нами на шаге 4. Остается лишь дать пользователю возможность отправить форму, и можно приступать к приготовлению пиццы

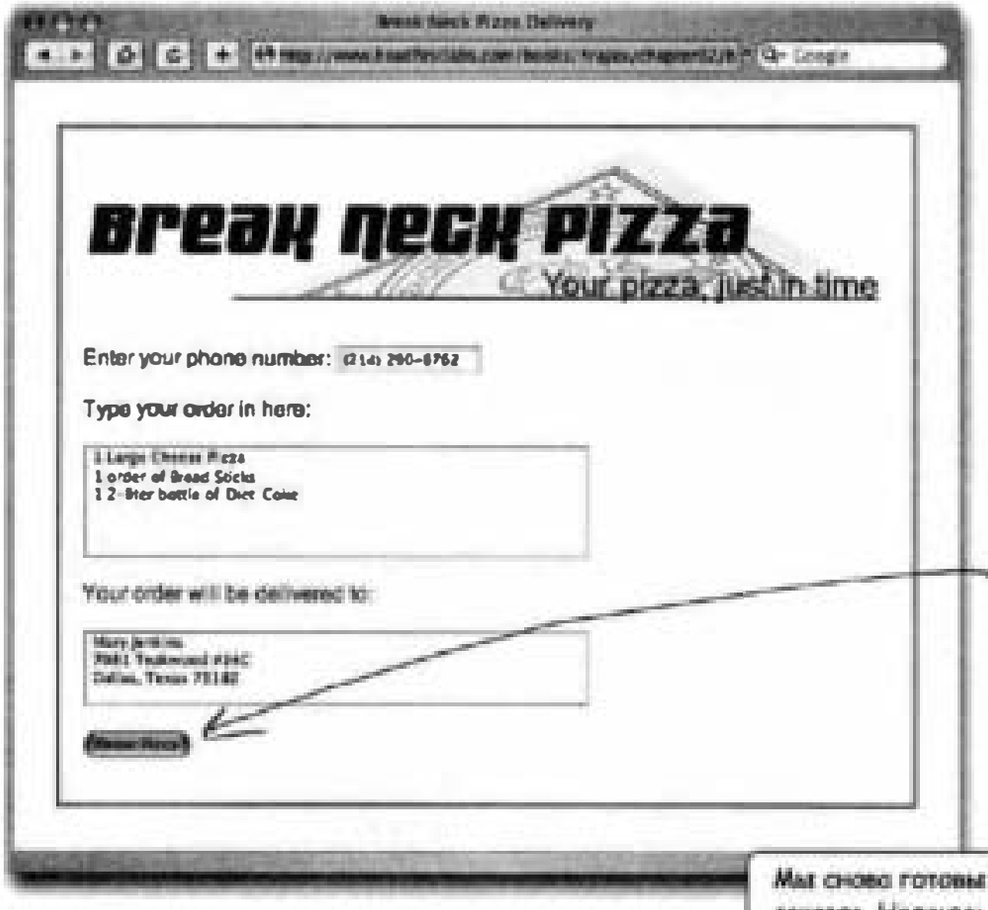




Возвращаемся к форме заказа

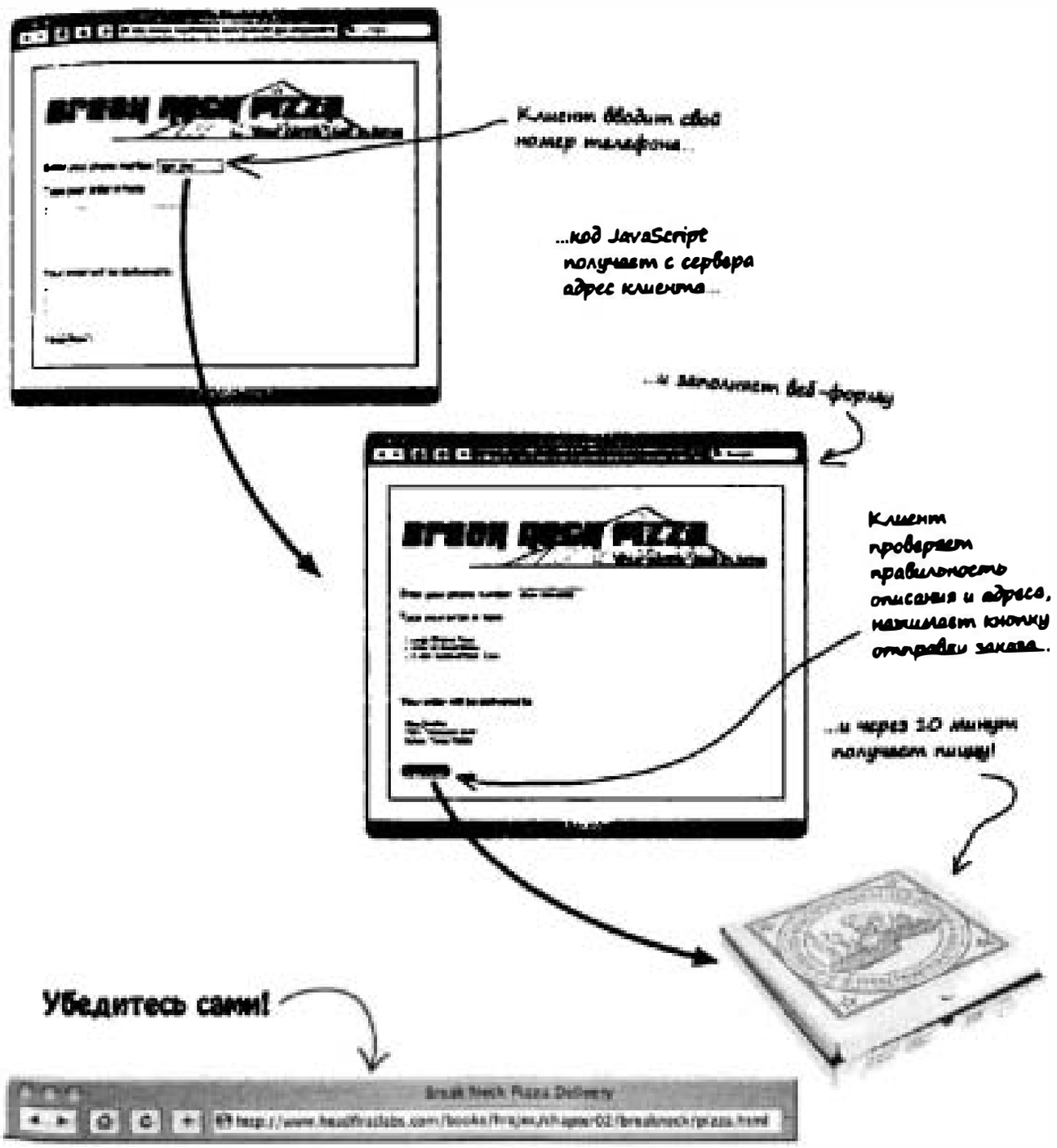


Разобравшись с кодом JavaScript, вернемся к форме заказа. После того как клиент введет свой телефон и описание заказа, а также проверит свой адрес, остается лишь отправить заказ.

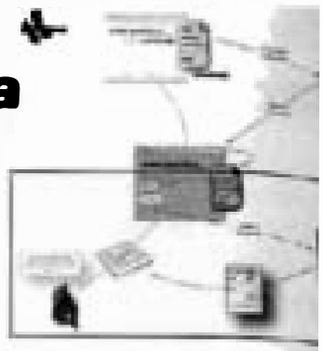


На самом деле эта часть уже готова.. Клиент отправляет свой заказ на сервер Break Neck, щелкнув на кнопке Order Pizza

Пробный запуск



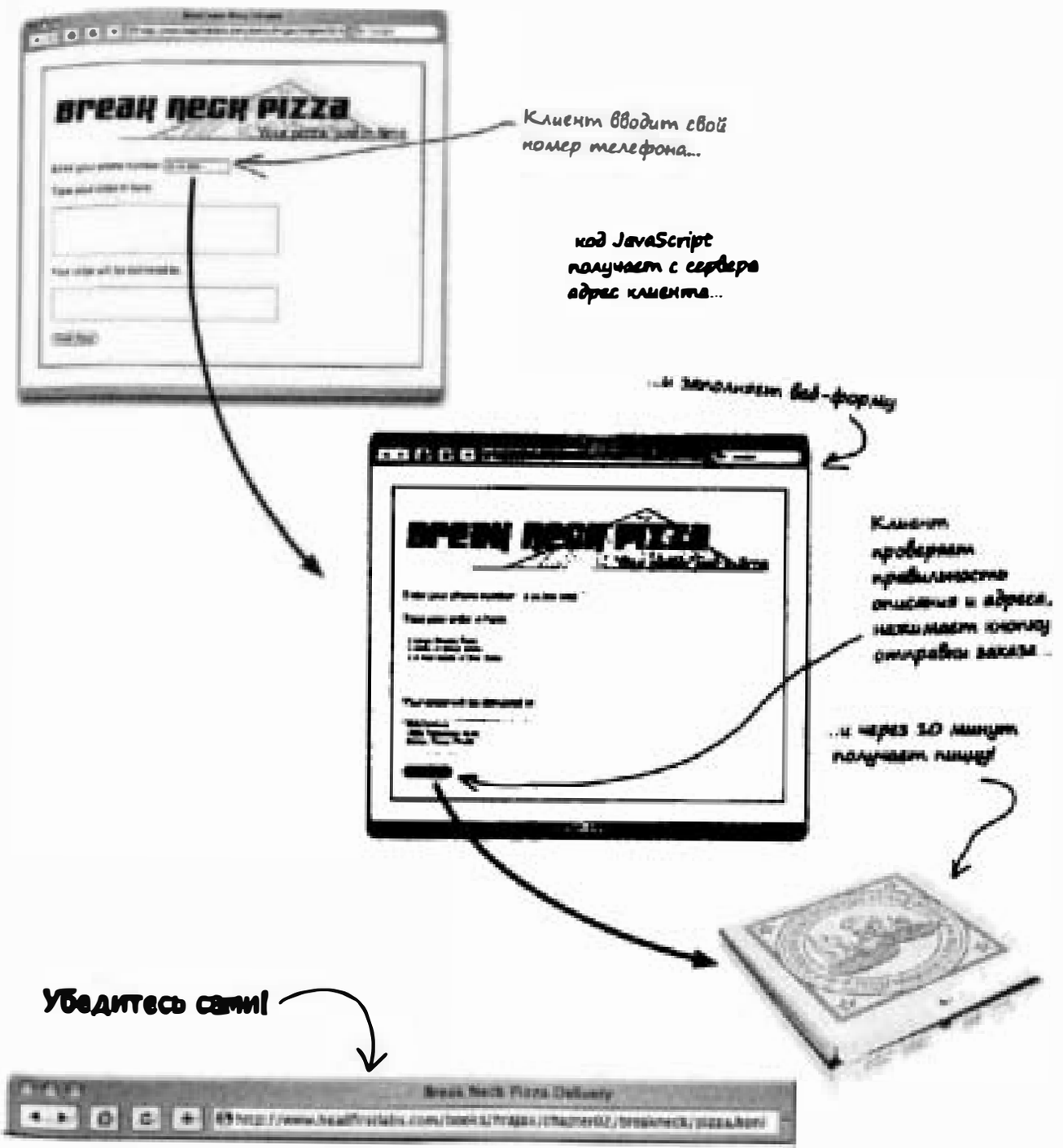
Возвращаемся к форме заказа



Разобравшись с кодом JavaScript, вернемся к форме заказа. После того как клиент введет свой телефон и описание заказа, а также проверит свой адрес, остается лишь отправить заказ.

На самом деле эта часть уже готова... Клиент отправляет свой заказ на сервер Break Neck, щелкнув на кнопке Order Pizza

Пробный запуск



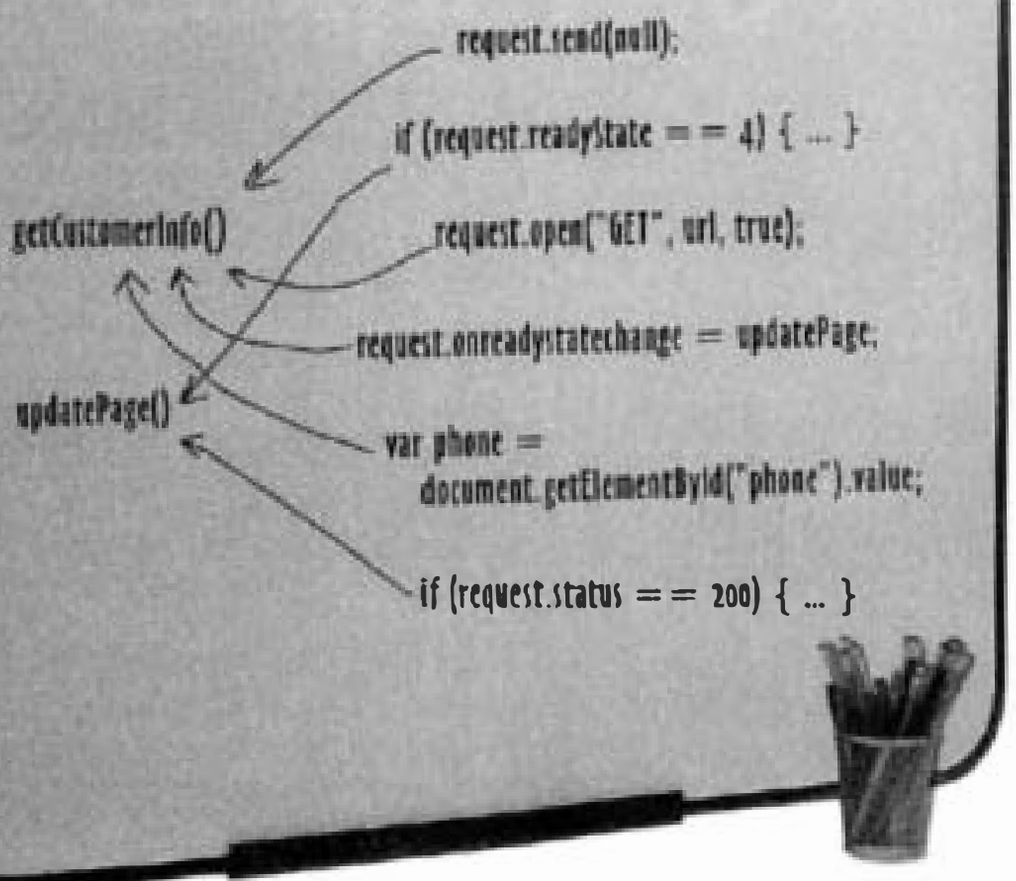
Обзор на 60 секунд

- В браузерах Microsoft запросы Ajax представлены объектом ActiveXObject, при этом в качестве типа объекта указывается строка Msxml2.XMLHTTP или Microsoft.XMLHTTP.
- В альтернативных браузерах, включая FireFox, Safari и Opera, запросы Ajax представлены объектом XMLHttpRequest.
- Статический код JavaScript не принадлежит ни одной функции и выполняется браузером при загрузке страницы.
- Используя статический код JavaScript, можно обеспечить выполнение некоторых фрагментов кода до того, как пользователь начнет работать с веб-страницей.
- Состояние готовности запроса определяет, на какой стадии находится его обработка: запрос инициализируется, устанавливается связь с сервером, сервер завершил обработку и т. д.
- Если состояние готовности запроса равно 4, сервер завершил обработку запроса, а данные ответа могут безопасно использоваться программным кодом.
- При каждом изменении состояния готовности запроса браузер запускает функцию обратного вызова, зарегистрированную для запроса.
- Обновление текста в визуальных элементах HTML (таких, как и) осуществляется с применением модели DOM; для изменения текста элементов полей форм (таких, как <input> и <textarea>) используется свойство value.
- Следите за тем, чтобы порядок следования полей соответствовал коду JavaScript приложения, а не противоречил ему.
- Чтобы решить проблему кэширования в таких браузерах, как Opera и Internet Explorer, необходимо использовать различающиеся URL запросов.

Решение упражнений

Сделайте это... — Ответ

Как и в главе 7, нам понадобится несколько функций JavaScript, обеспечивающих работу приложения Break Neck. В левом столбце приведены имена двух функций JavaScript, а в правом — несколько строк кода JavaScript. Соедините каждую строку с именем функции, которой принадлежит данная строка.



Совершенно секретно

Перед прочтением сжечь

Копирование строго запрещено

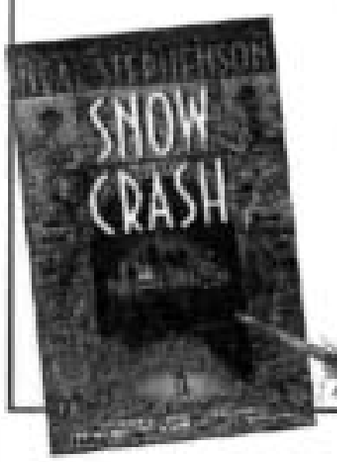
Добро пожаловать в проект «Хаос»

Поздравляем! Благодаря своим выдающимся практическим знаниям и опыту в построении веб-приложений вы являетесь основным кандидатом для участия в проекте «Хаос». Хотя мир и не подозревает о существовании нашей службы, мы следим за тем, чтобы ошибки в программах заставляли программистов допоздна засиживаться на работе, прогулка по берегу моря в лунном свете прерывалась срочным звонком по сотовому телефону, а вечер, начинающийся с розы и поцелуя, завершался в офисе, а не в спальне.

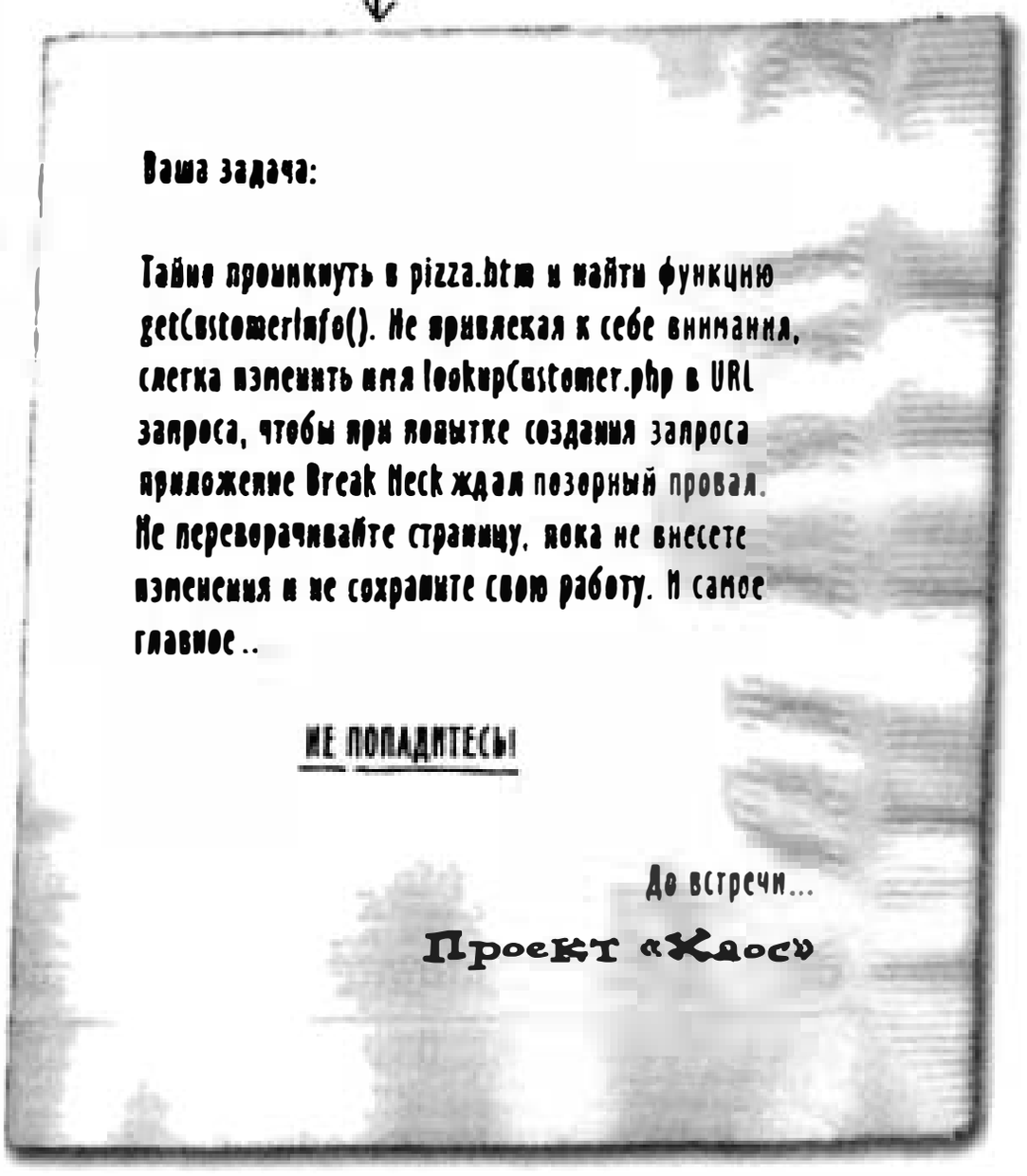
Нам удалось выяснить, что интерактивная форма заказа Break Neck обеспечит беспрецедентный уровень у пользователей и существенно сократит недовольство любителей пиццы в данном регионе. Более того, возможно дальнейшее распространение идей асинхронности приложений, в результате чего Интернет станет динамической, удобной для пользователя средой. Это совершенно неприемлемо!

Чтобы получить задание и понять, как лично вы можете противодействовать появлению приложений «нового поколения», займите столик на двоих ровно в 22:14 в ближайшей кофейне Starbuzz. Заложите синее перо между страниц книги Нила Стивенсона «Лавина».

С вами свяжется наш представитель; не пытайтесь связаться с нами самостоятельно, или последствия будут более серьезными, чем вы можете представить.



Записка, которую вам передал странный человек с пистолетами



Ваша задача:

Тайно проникнуть в pizza.htm и найти функцию getCustomerInfo(). Не привлекая к себе внимания, слегка изменить имя lookupCustomer.php в URL запроса, чтобы при попытке создания запроса приложение Break Neck ждал позорный провал. Не переворачивайте страницу, пока не внесете изменения и не сохраните свою работу. И самое главное..

НЕ ПОПАДИТЕСЬ!

До встречи...

Проект «Хаос»

Проблемы в Break Neck

Похоже, с приложением Break Neck что-то не так. Попробуйте открыть файл pizza.html в браузере и введите номер телефона. Что происходит?

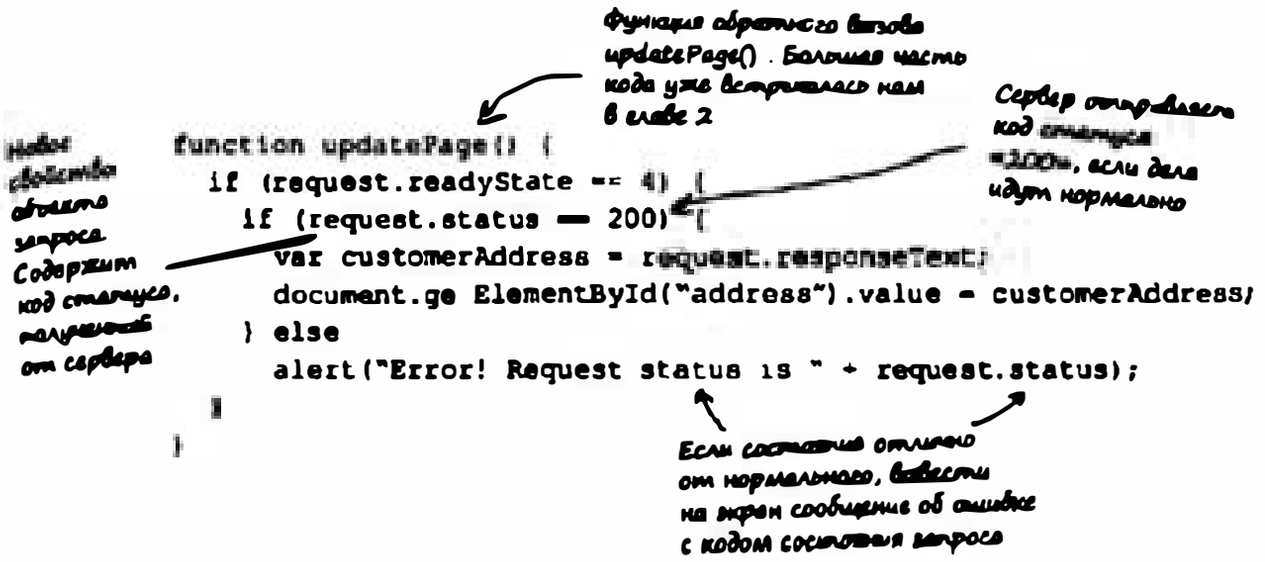
Хотя мы ввели телефон в поле адреса ничего не появилось

Но что еще хуже, страница даже не сообщила об ошибке!

Проверка статуса запроса

На самом деле браузер сообщил о возникшей проблеме — скорее всего, вы просто не обратили на это внимания. Для этой цели браузер устанавливает свойство объекта запроса с именем **status**.

Проверка состояния запроса выполняется следующим образом:



Просто сделайте это

Включите приведенный код в функцию **updatePage()** файла pizza.html. Перезагрузите страницу в браузере, введите телефон. Что происходит при переходе от телефона к следующему полю? Запишите код статуса, полученный от сервера:

_____

Теперь исправьте URL запроса в **getCustomerInfo()** так, чтобы он ссылался на правильный сценарий на сервере Break Neck. Перезагрузите страницу и посмотрите, что происходит. Код статуса изменился или остался прежним? Запишите код статуса для правильного URL:

_____

Один момент. Если URL запроса содержит ошибку, сервер получит наш запрос или нет?

URL запроса является относительным

Помните первую часть URL запроса из `lookupCustomerInfo()`?

```
lookupCustomer.php
```

Параметр запроса опущен для краткости

Этот URL является *относительным*; он не содержит доменного имени сервера, которому отправляется запрос. Какой же домен выбирает браузер при отправке запроса? Он автоматически использует домен, с которого запрашивалась веб-страница pizza.html. Следовательно, когда вы вводите URL вида http://www.breakneckpizza.com/pizza.html для просмотра формы заказа Break Neck, браузер преобразует относительный URL в *абсолютный*:

```
http://www.breakneckpizza.com/lookupCustomer
```

Запрос отправляется на тот же сервер, с которого браузер загрузил форму заказа. Даже если имя программы, работающей на сервере, задано неверно, запрос все равно будет направлен браузером на нужный сервер.

Программа может быть сервлетом, сценарий PHP, компонентом JSP или любой другой код, выполняемый на сервере

Это абсолютный URL. Он содержит имя домена

...и путь к программе или файлу на сервере

```
http://www.boardsrus.com/getUpdatesBoardSales.php
```

Это относительный URL. Он не содержит имени домена...

...а только путь к программе или файлу на сервере

```
/getUpdatesBoardSales.php
```

Браузер всегда запускает функцию обратного вызова... и он сообщил об ошибке.

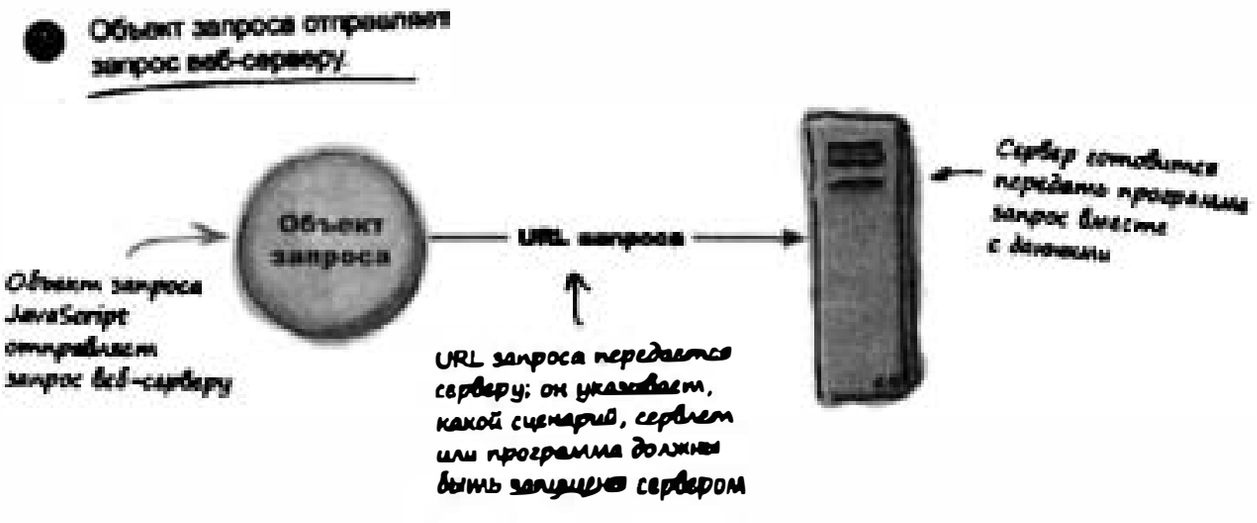
Браузер всегда запускает функцию обратного вызова, потому что это позволяет вам среагировать на действия сервера по обработке запроса. Поскольку мы используем асинхронный запрос, код обработки ответа сервера можно написать только одним способом — в виде функции обратного вызова. А чтобы помочь вам с обработкой ответа, браузер сообщает как состояние готовности запроса, так и его статус.

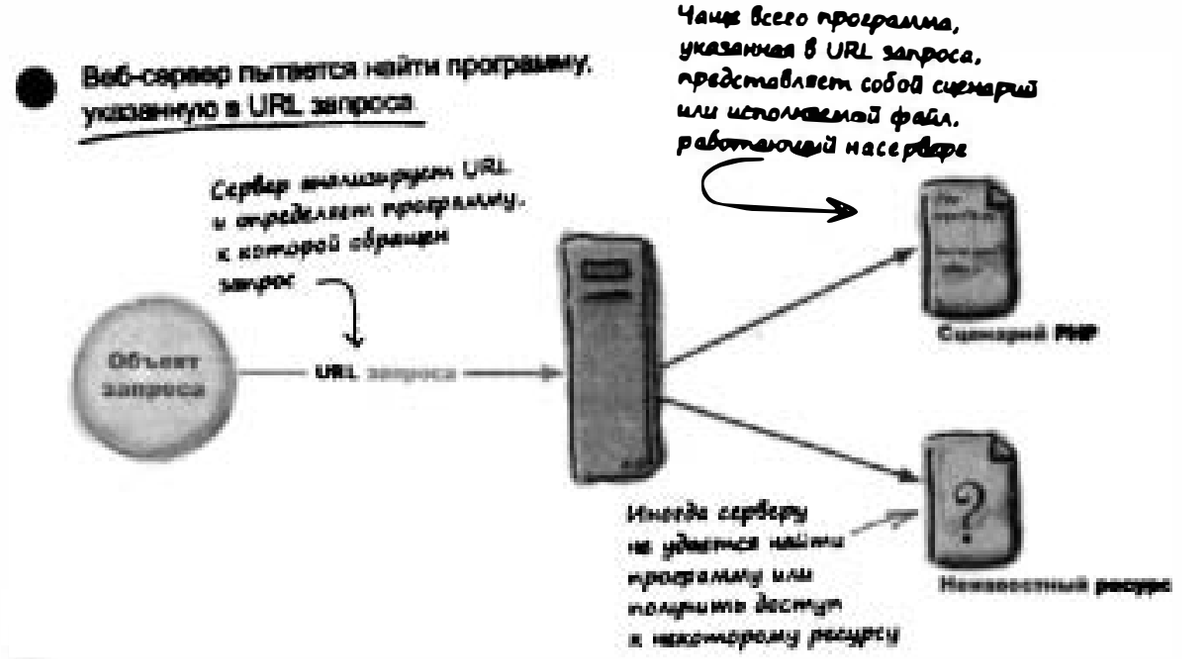
Не путайте состояние запроса с его статусом. Состояние готовности запроса указывает браузеру, на какой стадии находится обработка запроса: инициализация, обработка, завершение и т. д. Однако сам факт завершения еще не означает, что запрос завершился успешно... за это отвечает *статус* запроса

Сервер сообщает о любых проблемах с запросом при помощи кода статуса. Статус указывает, что произошло *во время запроса*, и развивались ли события именно так, как предполагалось. Следовательно, даже если запрос завершился, мы все равно должны проверить код статуса и убедиться в том, что все прошло нормально.

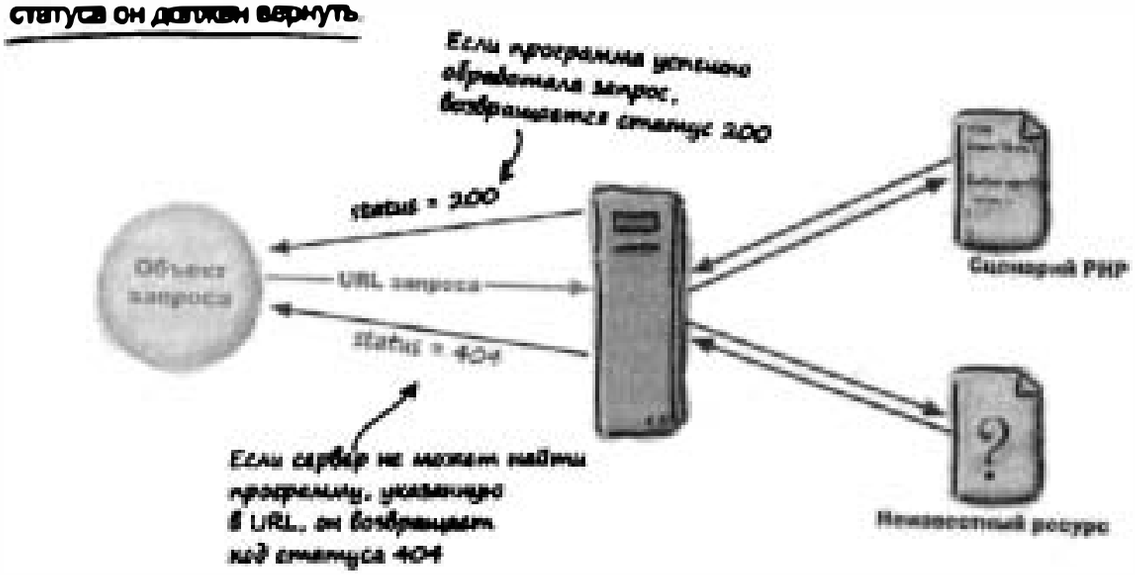
Сервер возвращает состояние готовности и код статуса

В своем ответе сервер возвращает довольно много полезной информации... вероятно, больше, чем вы предполагали. Следующая схема показывает, как организован возврат кодов сервером.

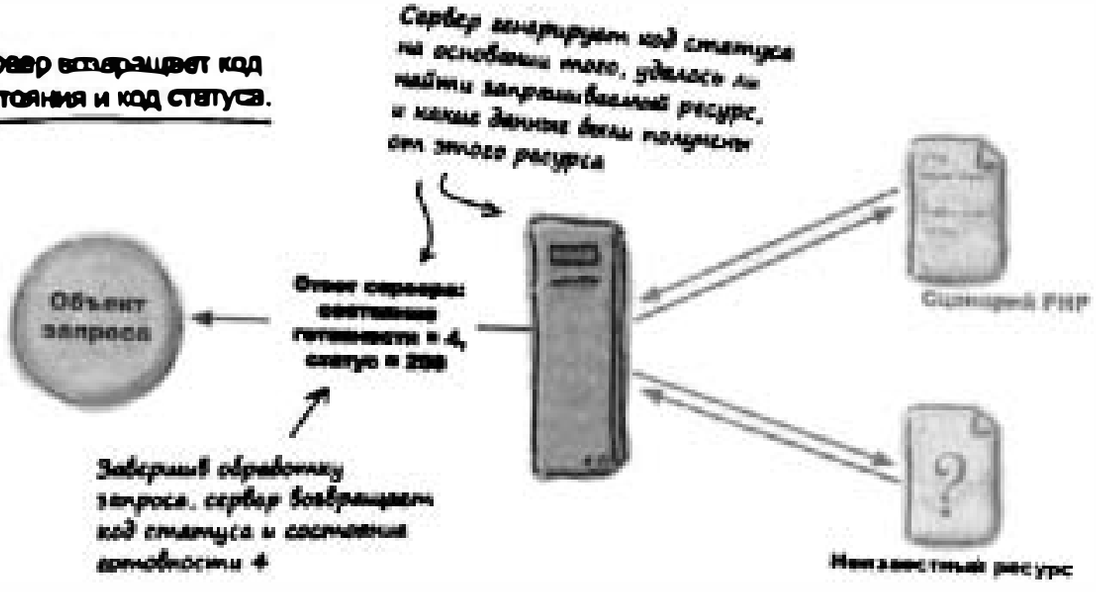




● **Сервер определяет, какой код статуса он должен вернуть.**



● **Сервер возвращает код состояния и код статуса.**



Часто задаваемые вопросы

В: Я думал, состояние готовности, полученное от сервера, говорит о том, что обработка запроса завершена. Почему мы должны дополнительно проверять код статуса?

О: Состояние готовности указывает лишь на то, что сервер завершил обработку запроса; именно по этой причине его необходимо проверить перед выполнением какого-либо кода в функции обратного вызова. Если проверка показывает, что обработка запроса завершена, необходимо убедиться в отсутствии ошибок; эта информация хранится в коде статуса. Чтобы быть уверенным в том, что запрос был обработан, а страницу можно обновить данными, полученными от сервера, необходимо как состояние готовности, так и статус.

В: Значит, состояние готовности должно быть равно 4, а статус — 200, верно?

О: Верно. Состояние готовности 4 означает, что запрос был полностью обработан, а статус 200 указывает на отсутствие ошибок.

В: Есть ли другие коды статуса, о которых необходимо знать?

О: Мы уже видели код статуса 404, означающий, что сервер не может найти запрашиваемую программу. Другой популярный код статуса — 403 — означает, что доступ к запрашиваемой программе или ресурсу либо запрещен, либо требует аутентификации.

Также существует множество других кодов статуса. Полный список находится по адресу http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

В: Придется ли мне писать код JavaScript для обработки всех кодов статуса?

О: В большинстве случаев достаточно проверить, равен ли статус 200, и вывести сообщение об ошибке при любом другом значении.

Конечно, можно попытаться написать хитроумный код JavaScript для повторной отправки запроса, но сбой при обработке запроса обычно свидетельствует об ошибке в веб-странице или коде JavaScript. Вывод сообщения на экран только собьет с толку пользователей.

В: Даже если URL запроса содержит ошибку, сервер ответит на мой вопрос и запустит функцию обратного вызова?

О: Мы используем относительный URL запроса, поэтому имя домена в запросе совпадает с именем сервера, предоставляющего вашу веб-страницу. Если веб-страница находится по адресу http://www.breakneckpizza.com/pizza.html, то по умолчанию для URL запроса будет использоваться имя домена http://www.breakneckpizza.com. После имени домена может следовать каталог, имя сценария или веб-приложения, и т. д.

Помните о различиях между сервером и программой, работающей на сервере. Если для вашего запроса известен домен веб-сервера, вы получите ответ — даже если ответ всего лишь сообщит, что запрашиваемая программа отсутствует или недоступна.

Выпираны проекта «Хаос» расшифровывают коды статуса HTTP по этой книге.

Возвращаемся к Break Neck...

Чтобы покинуть ряды проекта «Хаос», проследите за тем, чтобы в функцию **updatePage()** был включен код проверки статуса запроса. Также исправьте все ошибки в URL запроса в функции **getCustomerInfo()**. Сохраните изменения, снова загрузите файл **pizza.html** и убедитесь в том, что все работает правильно.

Убедитесь в том, что форма заказа пиццы работает правильно, ваш запрос находит адрес клиента, а заказ успешно отправляется

Молниеносная асинхронность

Где можно подождать? Извините, ждать некогда. Это Веб, а не вокзал, и никто не желает листать старые журналы, пока сервер занимается своим делом. Вы уже знаете, как Ajax избавляет от перезагрузки страниц, но сейчас настало время включить *восприимчивость* в список основных свойств веб-приложений. В этой главе вы узнаете, как отправлять запросы пользователей серверу, и как дать пользователю возможность продолжать работу, пока он ждет ответа. А впрочем... забудьте. В этой главе ждать вообще не придется.

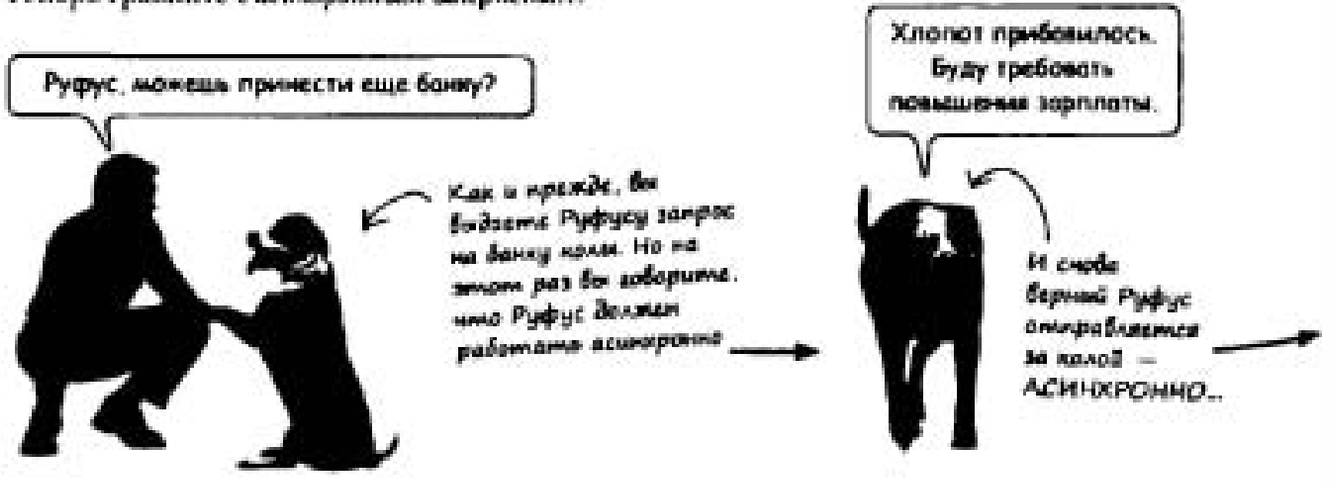
Что такое асинхронность на самом деле?

Термин **асинхронность** означает, что вам не придется **дожидаться**, пока веб-сервер отвечает на ваш запрос. Вы не проводите время впустую, а можете делать все, что захотите; когда сервер завершит обработку запроса, он вам об этом сообщит. Давайте попробуем взглянуть на происходящее «с высоты птичьего полета»; для начала стоит разобраться, что такое синхронный запрос, и сравнить его с асинхронным запросом.

Синхронный запрос

Асинхронный запрос

Теперь сравните с асинхронным запросом...

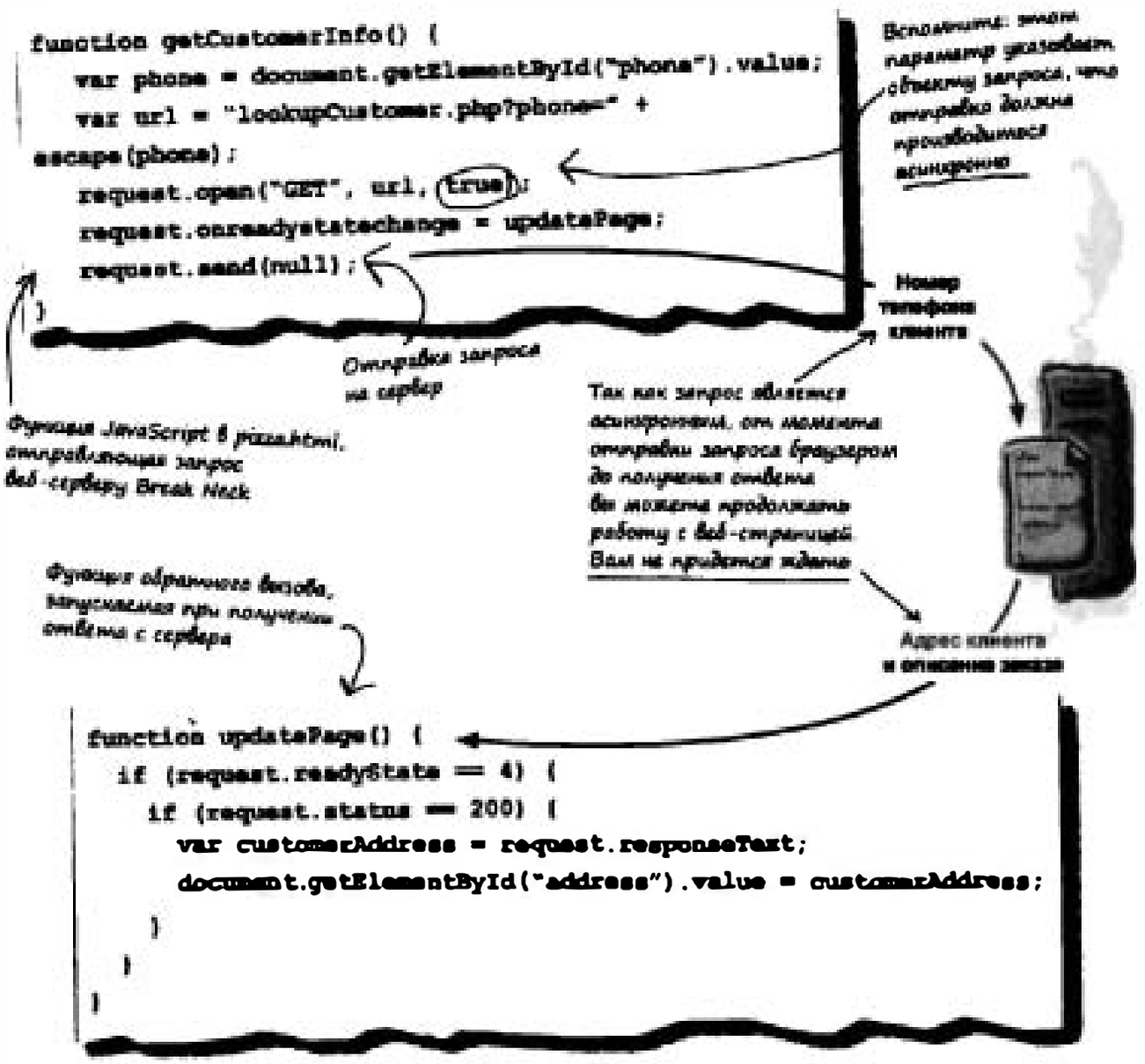



Ой, моя спина!
но так как вопрос является синхронным, вы полностью парализованы до получения ответа!
Наконец, Руфус возвращается к колодцу.
Молодец, Руфус! Но дальше работаем асинхронно... иначе моя спина больше не выдержит.
к моменту получения ответа вам уже пора к дантисту.


Пока он не вернется, вы можете заниматься чем угодно. Вам уже не придется торчать на одном месте, как в синхронном режиме.
Руфус приносит колу как раз в тот момент когда вы собираетесь отдохнуть. Гольф и прохладительные напитки: идеальное сочетание!
Результат обнадеживает: вы получаете колу. Разница в том, что вы не теряете время за пассивным ожиданием.

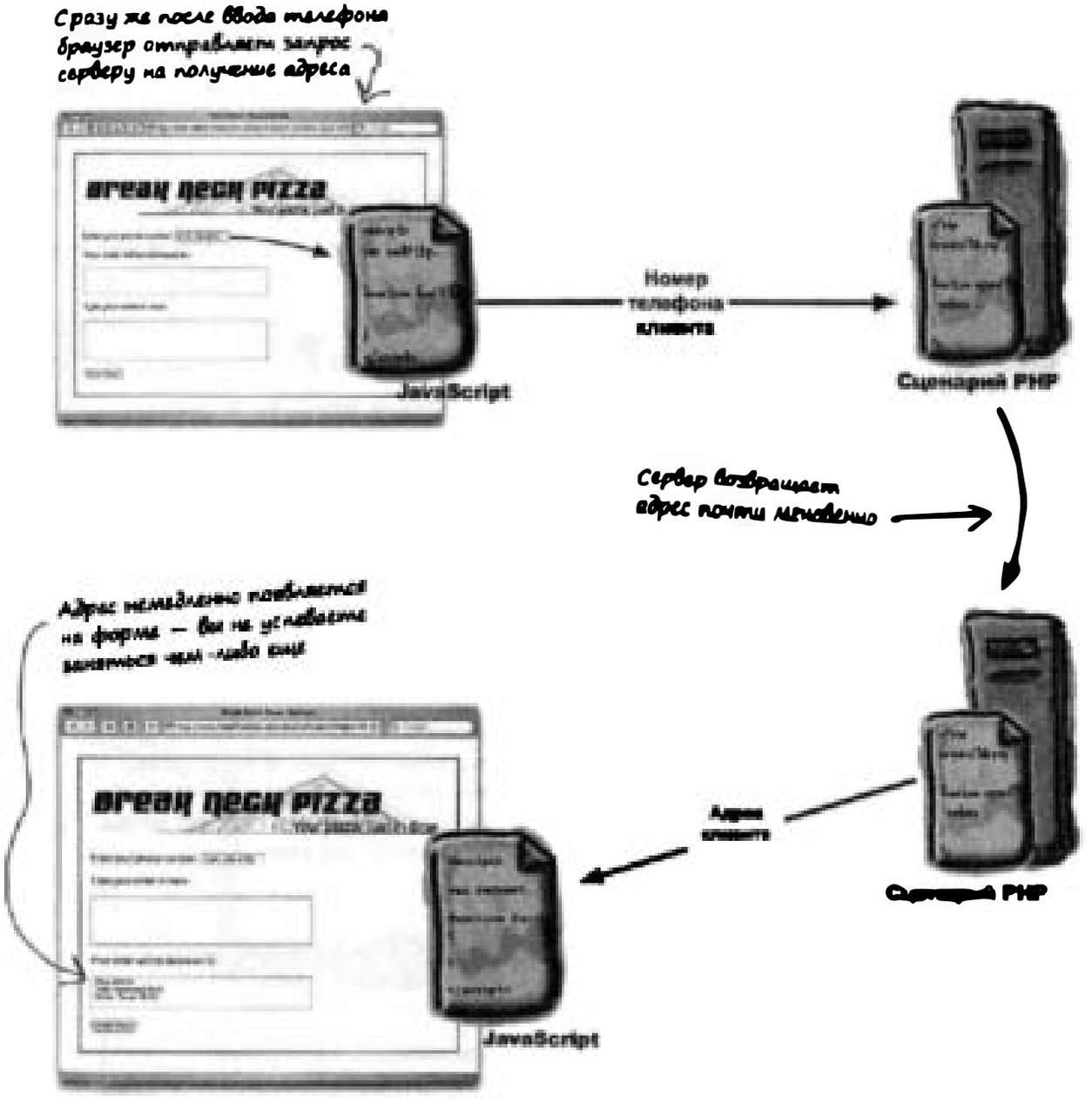
Break Neck Pizza — асинхронное приложение

Взглянем еще раз на Ajax-приложение Break Neck. Вы вводите на форме телефон, после чего переходите к описанию заказа или адресу. Код JavaScript страницы читает введенный номер с формы и отправляет веб-серверу запрос на получение адреса. Если поиск адреса займет много времени, вы даже можете отправить заказ, *не дожидаясь получения адреса от сервера.*



Вероятно, вы даже не заметили ...

Служба доставки Break Neck не зря называется «молниеносной». Отправка запроса и получение ответа от сервера происходят так быстро, что вы, вероятно, не успели приступить к вводу описания заказа, когда на форме уже появился адрес.



Что нам дает асинхронность?

Оба приложения, написанные нами, были асинхронными. Однако сервер отвечает на запросы настолько быстро, что вы, вероятно, не заметили никаких преимуществ от асинхронности.

Но что произойдет, если получение данных с сервера занимает действительно много времени? Или если вам действительно *необходимо* выполнять две задачи одновременно? Ведь вам хотелось бы продолжать работу с приложением во время ожидания ответа, не так ли?

Именно в таких ситуациях проявляются преимущества «асинхронности» Ajax. В этой главе мы создадим еще одно приложение — кофеварку на базе Ajax, и на этот раз асинхронность сыграет очень важную роль. Почему? Сейчас увидите

ШЕВЕЛИМ МОЗГАМИ

Какие приложения, с которыми вы работаете в Сети, выиграли бы от асинхронности?

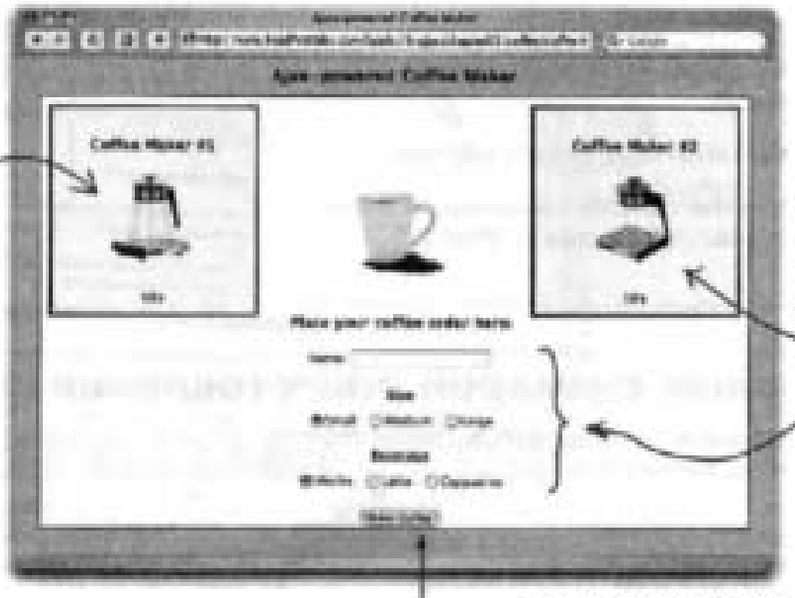
Кофеварка на базе Ajax

Даже две кофеварки...
...и целая контора кофеманов

Вы являетесь «ответственным за кофе» в своей организации. Ваша задача — позаботиться о том, чтобы потребности ваших коллег в кофеине удовлетворялись... и как можно быстрее. Все коллеги являются заядлыми кофеманами, и если под рукой у них не окажется чашки свежесваренного кофе, это их раздражает. К счастью, в офисе установлены две кофеварки; даже если одна из них занята, можно воспользоваться второй кофеваркой.

Мы написали Ajax-приложение, позволяющее заказывать кофе по Интернету.

Наше приложение должно заказывать кофе и отслеживать текущее состояние обеих кофеварок. Вот как оно будет выглядеть:



Первая (основная) кофеварка. Если при получении нового заказа она свободна, то заказ выполняется этой кофеваркой. Тем не менее на приготовление чашки кофе требуется время, а в любой момент времени готовится может только одна чашка

Вторая кофеварка. Если при получении нового заказа кофеварка №1 занята, заказ передается второй кофеварке. Для приготовления кофе ей также требуется определенное время

Здесь вводится информация о заказе: имя, размер порции и тип напитка

Когда вся необходимая информация будет введена, пользователь щелкает на кнопке Order Coffee. Заказ передается свободной кофеварке.

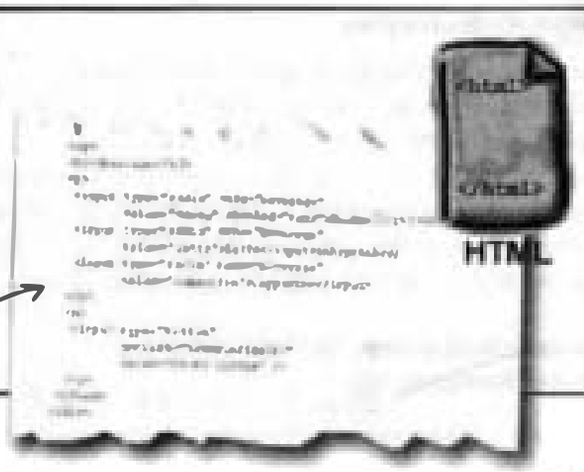
Три ингредиента асинхронного кофе

Программное обеспечение кофеварки на базе Ajax состоит из трех основных частей:

1 Код HTML

Прежде всего нам потребуется код HTML для веб-страницы. Страница принимает заказы и выводит информацию о состоянии двух кофеварок.

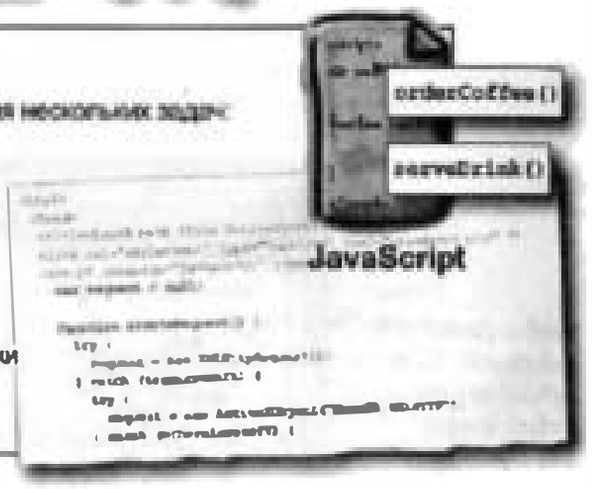
Элементы для ввода информации о заказе на странице HTML



2 Код JavaScript

Нам потребуется код JavaScript для решения нескольких задач:

- код создания объекта запроса;
- функция передачи заказа сценарию кофеварки;
- функция выдачи сваренного напитка;
- обработчики событий, связывающие кнопки веб-формы с функциями JavaScript.



3 Серверный сценарий приготовления кофе

Сценарий работает на стороне сервера и варит кофе при получении нового запроса.

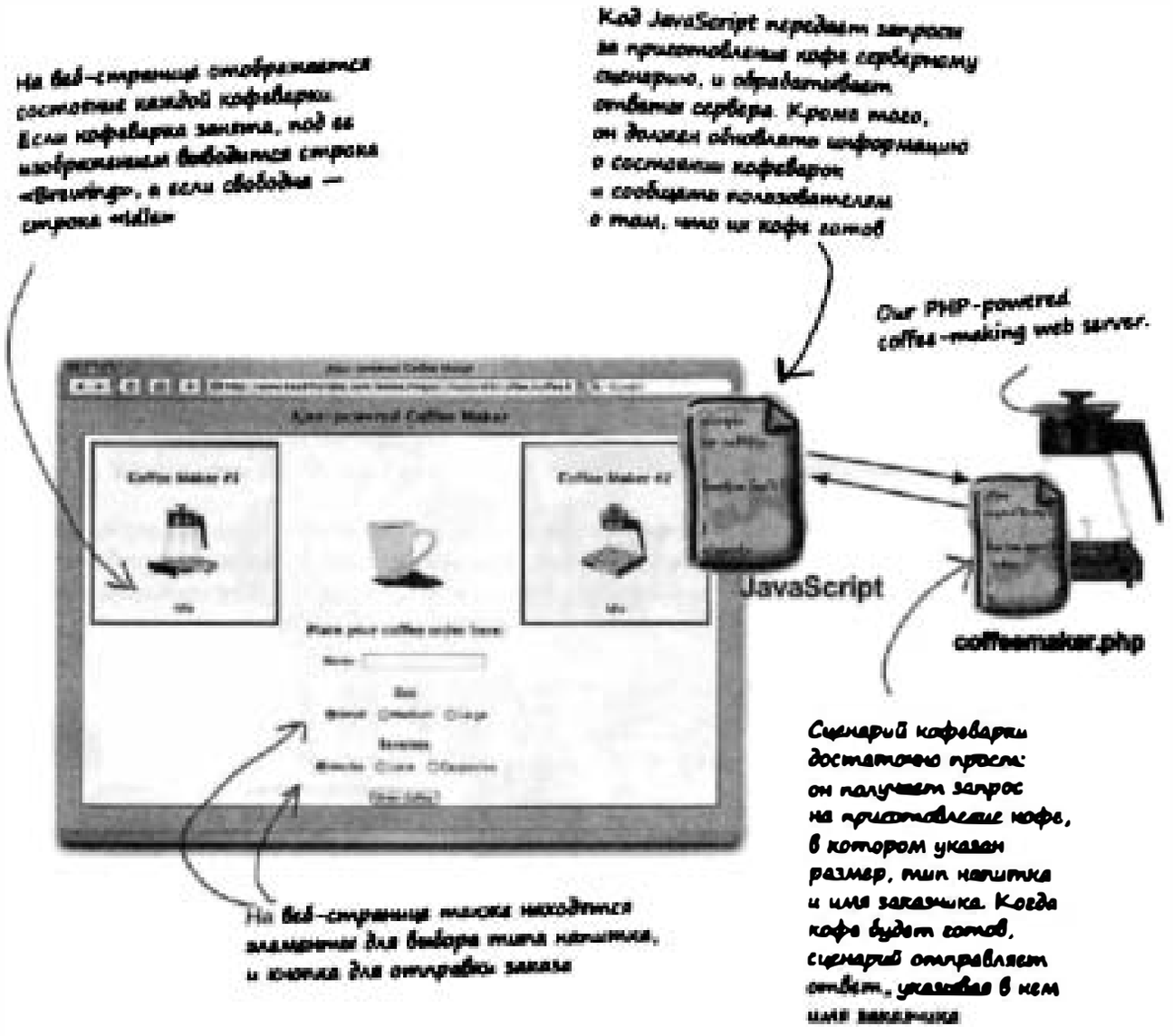
Сценарий PHP для приготовления кофе. Все заказы передаются этому сценарию для обработки

Сценарий обслуживает обе кофеварки, так как в нашем приложении могут одновременно готовиться две чашки

Сценарий PHP

Соединяем отдельные части

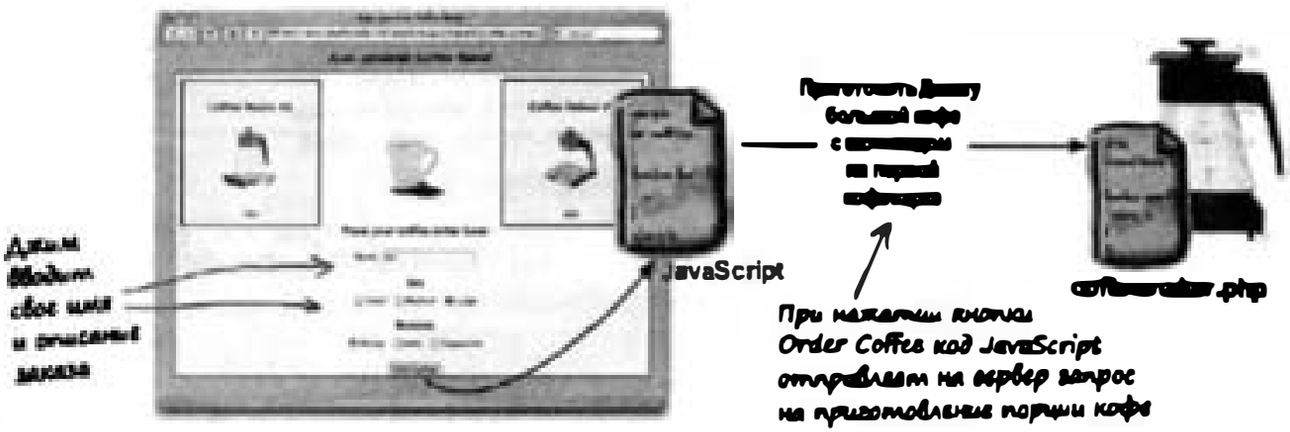
Теперь вы знаете, из каких частей будет состоять наше приложение. Но давайте сделаем следующий шаг и посмотрим, как эти части объединяются в технологическое чудо. Итак, имеется форма HTML., код JavaScript и серверный сценарий приготовления кофе. Как организовано взаимодействие этих компонентов?



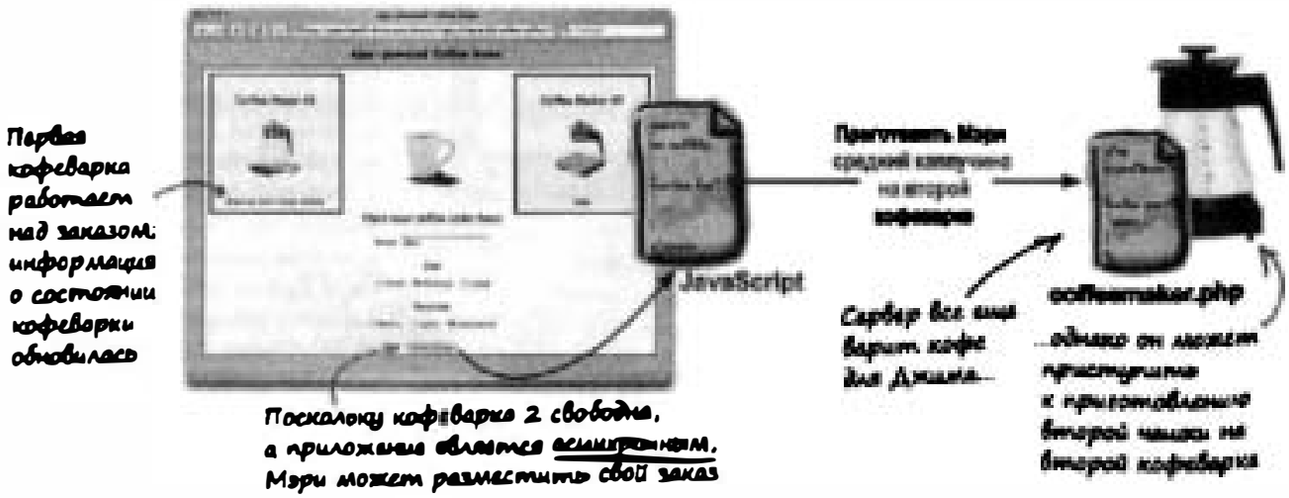
Как работает Ajax-кофеварка?

Итак, вы достаточно хорошо себе представляете, что понадобится для построения нашего приложения. Остается лишь также хорошо понять, что оно должно делать. Прежде чем погружаться в анализ кода HTML и JavaScript, давайте разберемся, как в нашем приложении будет организован запрос на приготовление чашки кофе.

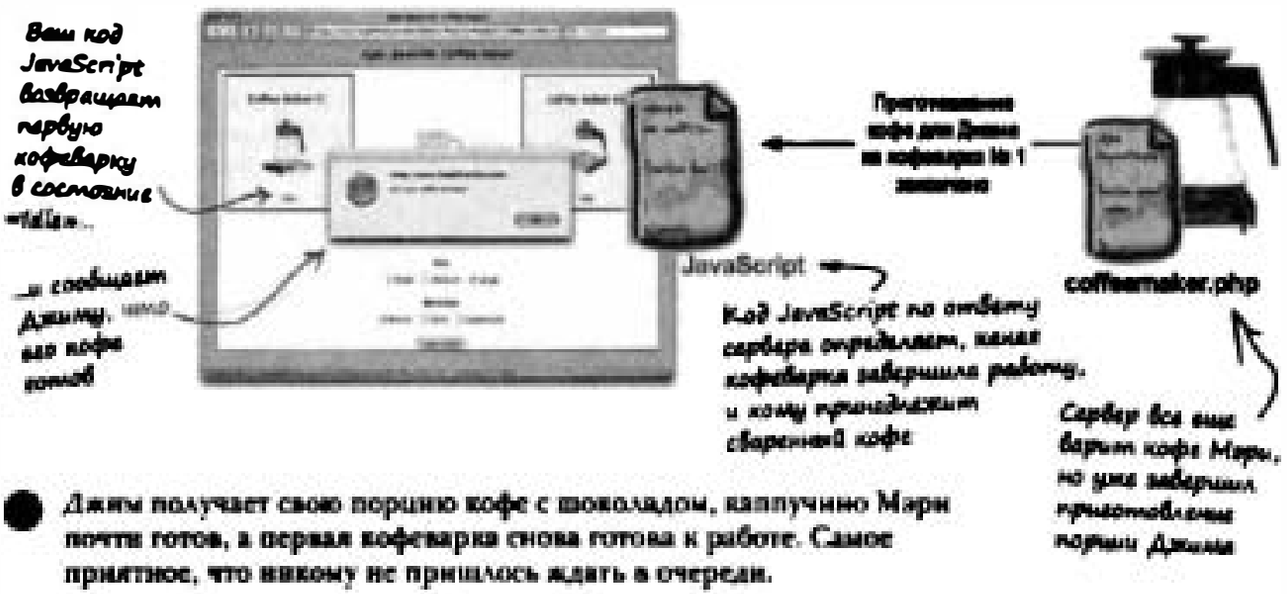
- Предположим, Джим желает получить свою порцию кофеина. Он вводит свое имя на веб-форме, выбирает большую порцию (Large) с шоколадом (Mocha), после чего щелкает на кнопке Order Coffee. Это действие приводит к запуску кода JavaScript, отправляющего запрос первой кофеварке



- Пока первая кофеварка готовит кофе для Джима, Мэри решает, что для поддержания настроения ей необходима порция капучино. К счастью, наше приложение работает асинхронно — Мэри может заказать напиток на второй кофеварке, хотя сервер все еще занят приготовлением кофе в ответ на более ранний запрос Джима.



- Первая кофеварка завершает приготовление порции Джима и возвращает ответ вашей странице. Браузер запускает функцию обратного вызова JavaScript, которая обновляет информацию о состоянии кофеварки и сообщает Джиму, что его заказ готов.



- Джим получает свою порцию кофе с шоколадом, каппучино Мэри почти готов, а первая кофеварка снова готова к работе. Самое приятное, что никому не пришлось ждать в очереди.

Просто сделайте это

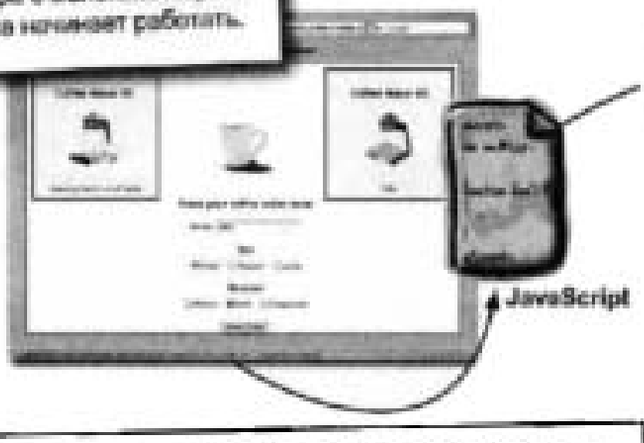
Найдите в архиве примеров каталог chapter3/coffee. Готовый код PHP хранится в файле coffeemaker.php. Вы также найдете файл HTML coffee.htm и файл coffee.css с таблицей стилей CSS. Ваша задача — доделать файл coffee.html и написать код JavaScript, который будет приводить кофеварку в действие. Но сначала переверните страницу — вас ждет маленькая головоломка...

Кофейная головоломка

Как вы думаете, что произойдет, если реализовать приложение-кофеварку в синхронном, а не в асинхронном режиме? Представьте следующую ситуацию и посмотрите, удастся ли вам найти правильный ответ.

Правильный ответ будет приведен далее в этой главе, так что вы узнаете его не сразу!

1 Сара заказывает маленькую чашку кофе с молоком. Первая кофеварка начинает работать.



Сказать Сара маленькую кружку кофе с молоком на первой кофеварке

coffeemaker.php

2 Вашей коллеге нужен кофе. Может ли она воспользоваться второй кофеваркой, пока первая кофеварка выполняет заказ Сары? Помните: речь идет о синхронном приложении (по крайней мере сейчас).

Первая кофеварка продолжает готовить кофе для Сары

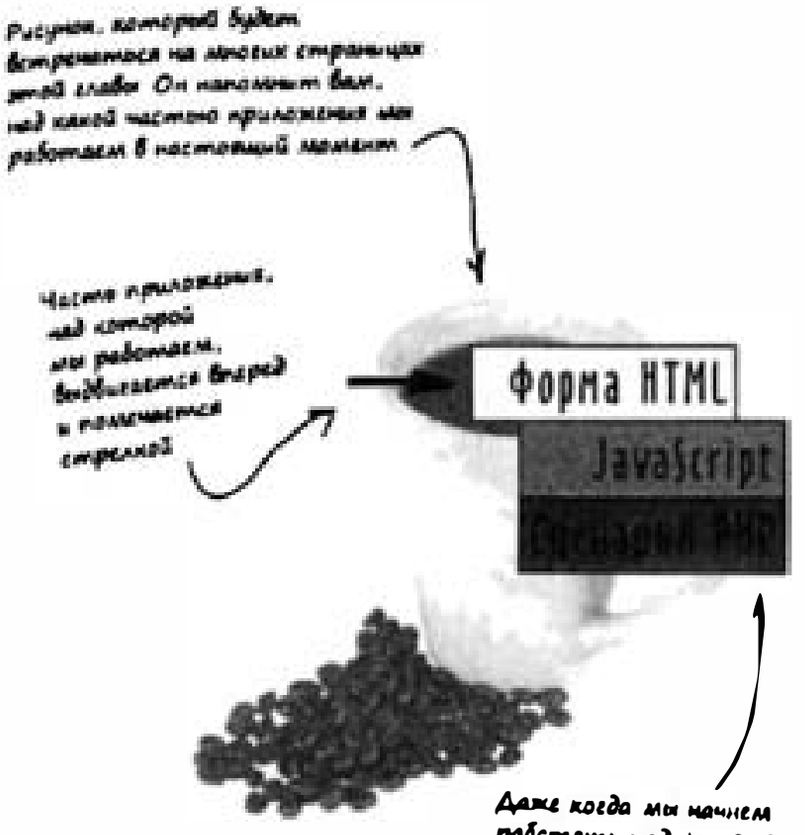
Вопрос: сможем ли Джанис использовать вторую кофеварку, пока первая выполняет заказ Сары?

Джанис хочет кофе

coffeemaker.php

Итеративная разработка Ajax-приложений

В главах 1 и 2 мы начали работать над приложением и создавали его последовательно, шаг за шагом, пока работа не была завершена. В этой главе мы будем переключаться между тремя составляющими нашего приложения в процессе работы, возвращаясь к ним снова и снова. Такой подход, при котором вы снова и снова перерабатываете одни и те же фрагменты, называется **итеративной разработкой**.



Большинство приложений разрабатывается в итеративном режиме: разработчик снова и снова возвращается к тем компонентам, над которыми он уже работал, дополняя и совершенствуя их.

Код HTML

Впрочем, довольно разговоров о кофе; пора приступать к построению асинхронного приложения. Начнем с рассмотрения кода HTML приложения, хранящегося в файле coffee.html. Далее приводится код HTML для размещения заказа и обслуживания первой кофеварки (второй кофеваркой мы займемся позже). Давайте посмотрим, с чем нам предстоит работать.

```
<html>
 <head>
  <title>Ajax-powered Coffee Maker</title>
  <link rel="stylesheet" type="text/css" href="coffee.css" />
 </head>
 <body>
  <div id="header">
   <h1>Ajax powered Coffee Maker</h1>
  </div>

  <div id="wrapper">
   <div id="coffeemaker1">
    <h2>Coffee Maker #1</h2>
    <p><img src="images/CoffeeMaker1.gif"
            alt="Coffee Maker #1" /></p>
    <div id="coffeemaker1-status">Idle</div>
   </div>

   <div id="coffeeorder">
    <p><img src="images/coffeeMugWithBeans.jpg" alt="Coffee Pot 1"
/></p>
    <h2>Place your coffee order here:</h2>
    <div id="controls1">
     <form>
      <p>Name: <input type="text" name="name" id="name" /></p>
      <h3>Size</h3>
      <p>
       <input type="radio" name="size"
              value="small" checked="true">Small</input>
         
       <input type="radio" name="size"
              value="medium">Medium</input>
         
       <input type="radio" name="size"
              value="large">Large</input>
      </p>
```

Стандартный заголовок. Мы подключаем внешнюю таблицу стилей и включаем название приложения

Код HTML первой кофеварки. Последний элемент показывает, свободна ли кофеварка, или занята изготовлением чего-то заказа. Обратите внимание на применение элемента для хранения текста состояния

Элемент для заказа кофе

Здесь вводится имя, чтобы кофеварка знала, для кого она готовит напиток

Здесь выбирается размер порции: малой, средний, большой

продолжение на следующей странице

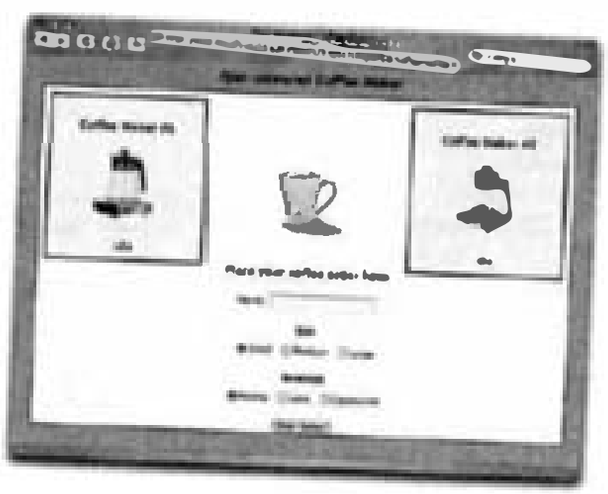
```
      <h3>Beverage</h3>
      <p>
       <input type="radio" name="beverage"
              value="mocha" checked="true">Mocha</input>
         
       <input type="radio" name="beverage"
              value="latte">Latte</input>
         
       <input type="radio" name="beverage"
              value="cappucino">Cappucino</input>
      </p>
      <p>
       <input type="button"
              onClick="_____"
              value="Order Coffee" />
      </p>
     </form>
    </div>
   </div>

   <div id="_____">
    <h2>Coffee Maker #2</h2>
    <p><img src="images/CoffeeMaker2.gif" alt="Coffee Maker #2"
/></p>
    <div id="_____">Idle</div>
   </div>

   <p id="clear"></p>
  </div>
 </body>
</html>
```

Последняя секция выбора заказа: здесь выбирается нужный тип кофе

Кнопка для отправки заказа. Мы еще вернемся к ней и заполним обработчик события onClick

Позднее в этой главе мы дополним код HTML для второй кофеварки

Маленький трюк HTML/CSS, который гарантирует, что все элементы для выбора заказа будут размещаться в основной части формы



Пока мы не начали работать над JavaScript, я подумал... А не лучше ли будет разместить весь код JavaScript в отдельном файле, не смешивая его с HTML?

Давайте разместим код JavaScript в отдельном файле.

Вместо того чтобы размещать весь код JavaScript прямо в файле HTML, давайте выделим его в отдельный файл. Ссылка на файл с кодом включается в элемент `<script>` в секции `<head>` файла HTML. Всем известно, почему HTML следует отделять от CSS; по тому же принципу JavaScript желательно отделить от HTML: логика приложения (JavaScript) отделяется от структуры приложения и его представления (HTML и CSS).

Вероятно, вам порядком надоело снова и снова вводить одинаковый код JavaScript во всех приложениях. Чтобы избавить вас от этой необходимости, мы создадим два файла с кодом JavaScript. Первый файл будет содержать код, постоянный для всех приложений, а второй — специфический код конкретного приложения. Файл с общим кодом будет называться ajax.js, а файл с кодом, специфическим для приложения CoffeeMaker — coffee.js. Включите в файл HTML два элемента `<script>` со ссылками на два файла JavaScript, которые мы собираемся создать:

Пожалуй, нам бы следовало создать два файла — для кода JavaScript, входящего во все приложения Ajax, и для кода JavaScript, специфического для кофеварки. Тогда общий код JavaScript можно будет использовать во всех новых приложениях.

```
<html>
 <head>
  <title>Ajax-powered Coffee Maker</title>
   <link rel="stylesheet" type="text/css"
         href="coffee.css" />
   <script type="text/javascript" src="ajax.js"> </script>
   <script type="text/javascript" src="coffee.js"> </script>
 </head>
<body>
  ...
</body>
</html>
```

Два элемента `<script>` со ссылками на файлы JavaScript. Обязательно включите пробел и тег `</script>` — без этого некоторые браузеры не загружают код JavaScript, на который указывает ссылка

Просто сделайте это

Ниже приведена структура кода JavaScript, который мы собираемся включить в приложение Coffee Maker. Укажите для каждого фрагмента JavaScript, в каком файле должен находиться код — ajax.js или coffee.js.

Напишите имя файла, в котором должен находиться каждый из представленных фрагментов

```
try {
  request = new XMLHttpRequest();
} catch (trymicrosoft) {
  try {
    request = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (othermicrosoft) {
    try {
      request = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (failed) {
      request = null;
    }
  }
}
```

?

```
function getBeverage() {
   // Figure out what beverage was selected
}
```

```
function serveDrink() {
   // When the coffee maker is done, serve the drink
}
```

```
if (request == null)
  alert("Error creating request object!");
```

```
function orderCoffee() {
   // Take an order from the web form
}
```

```
function sendRequest(url) {
   // Send a request to the Coffee Maker
}
```

```
var request = null;
```

```
function getSize() {
   // Figure out what size cup was selected
}
```

Часто задаваемые вопросы

В: Я не могу понять, в какой файл поместить sendRequest() — в ajax.js или coffee.js. С одной стороны, эта функция должна присутствовать в каждом Ajax-приложении; с другой стороны, она содержит код, специфический для конкретного приложения. В каком файле должна находиться эта функция?

О: Вы правы, это трудный вопрос. Наверное, возможны оба варианта. Мы решили разместить функцию sendRequest() в файле coffee.js, хотя sendRequest(), как вы убедитесь через несколько страниц, не содержит специфического кода. Мы выбрали этот вариант, потому что нам кажется логичным хранить функции sendRequest() и serveDrink() вместе. Так как функция serveDrink() содержит специфический код приложения, обе функции были размещены в файле coffee.js

В: Зачем нужны отдельные функции для определения размера и типа напитка?

О: Как вы вскоре увидите, для чтения значения кнопки-переключателя (элемента, используемого для выбора размера порции и типа напитка) требуется несколько строк кода. Мы предпочитаем хранить их отдельно от остального кода функции orderCoffee(), в котором строится URL запроса и обновляется текст с состоянием кофеварок.

В: Похоже, объект запроса создается в статическом коде. Почему мы снова это делаем?

О: При размещении кода создания объекта запроса в статическом коде (то есть коде JavaScript, не принадлежащем ни одной функции), все сообщения об ошибках будут выводиться до того, как пользователь начнет работать с приложением. Так предотвращается ситуация, когда пользователь вводит заказ, а потом узнает о возникших проблемах.

Вот что мы сделали...

Весь код создания и проверки объекта запроса включается в файл ajax.js, а все специфические функции — в coffee.js. Если вы выберете другой способ, это тоже нормально. Главное — помните, где находится каждая функция, чтобы вы знали, какой файл следует редактировать во время работы над приложением.

ajax.js

```
var request = false;
try {
  ...
}

if (request == null)
{
  ...
}
```

coffee.js

```
function sendRequest(url)
{...}
function serveDrink() {...}
function orderCoffee() {...}
function getSize() { ... }
function getBeverage() {...}
```

Добро пожаловать в мир интерактивной разработки... Сейчас мы займемся доработкой HTML, зная начинки с кода JavaScript

Просто сделайте это

Зная имена функций JavaScript, мы можем написать оставшийся код HTML.

Файл coffee.html

```
    <h3>Beverage</h3>
    <p>
     <input type="radio" name="beverage"
            value="mocha"
            checked="true">Mocha</input>
       
     <input type="radio" name="beverage"
            value="latte">Latte</input>
       
     <input type="radio" name="beverage"
            value="cappucino">Cappucino</
input>
    </p>
    <p>
     <input type="button"
            onClick="_____"
            value="Order Coffee" />
    </p>
   </form>
  </div>
</div>

<div id="_____">
 <h2>Coffee Maker #2</h2>
 <p><img src="images/CoffeeMaker2.gif"
         alt="Coffee Maker #2" /></p>
 <div id="_____">Idle</div>
</div>
```

1. Заполните пробел в коде HTML — подставьте имя функции, которая должна запускаться, когда пользователь щелкает на кнопке Order Coffee

2. Допишите код HTML второй кофеварки. Помните, что атрибуты id всех элементов должны быть уникальными. От этого зависит работа кода JavaScript, поэтому сверьте свой ответ с нашим (на следующей странице) перед тем, как двигаться дальше.

Если у вас возникнут затруднения, вернитесь к коду HTML первой кофеварки и возьмите все за образец для заполнения пропусков в коде HTML второй кофеварки

Просто сделайте это — решение

Зная имена функций JavaScript, мы можем написать оставшийся код HTML.

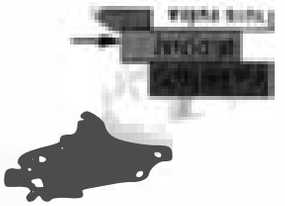
```
    <h3>Beverage</h3>
    <p>
     <input type="radio" name="beverage"
            value="mocha"
            checked="true">Mocha</input>
       
     <input type="radio" name="beverage"
            value="latte">Latte</input>
       
     <input type="radio" name="beverage"
            value="cappucino">Cappucino</input>
    </p>
    <p>
     <input type="button"
            onClick="orderCoffee();"
            value="Order Coffee" />
    </p>
   </form>
  </div>
</div>

<div id="coffeemaker2">
 <h2>Coffee Maker #2</h2>
 <p><img src="images/CoffeeMaker2.gif"
         alt="Coffee Maker #2" /></p>
 <div id="coffeemaker2-status">Idle</div>
</div>
```

функция JavaScript для отправки нового заказа сценарию PHP (мы ее еще напишем)

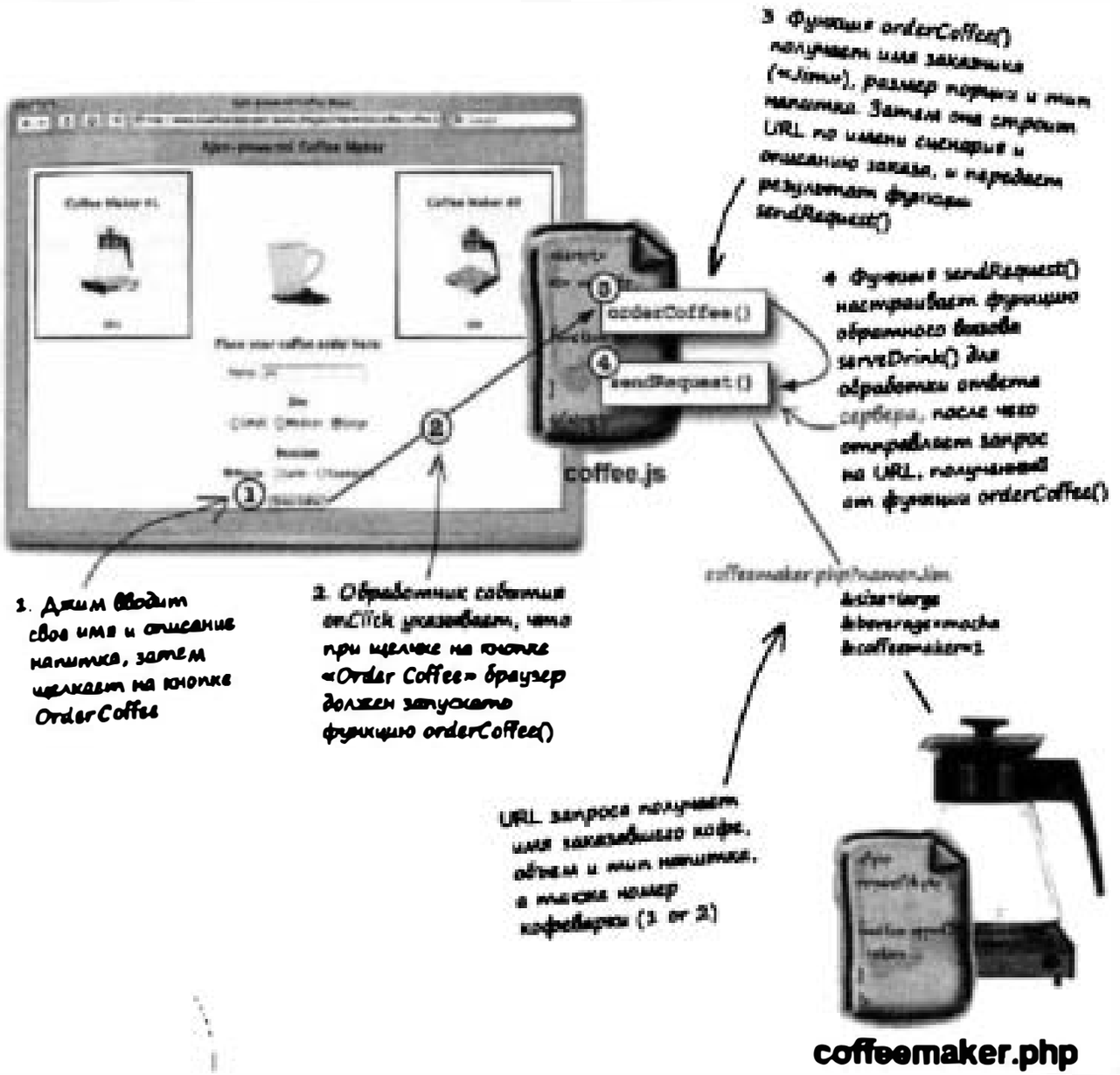
В коде первой кофеварки эти идентификаторы были равны «coffeemaker1» и «coffeemaker1-status»...

...поэтому для второй кофеварки мы просто заменяем «1» на «2»

Отправка запроса на приготовление кофе



Код HTML завершен, возвращаемся к JavaScript. Сейчас мы напишем код JavaScript для отправки серверу запроса на приготовление кофе. Вероятно, к настоящему моменту вы уже стали настоящим профессионалом по выдаче запросов, так что с функцией **orderCoffee()** справитесь легко, не так ли? Для начала подумаем, что должна делать эта функция.



Написание кода JavaScript для отправки запроса

Весь данный код должен находиться в данной книги coffee.js

```
function sendRequest(url) {
  request.onreadystatechange = serveDrink;
  request.open("GET", url, true);
  request.send(null);
}

function orderCoffee() {
  var name = document.getElementById("name").value;
  var beverage = getBeverage();
  var size = getSize();

  var url = "coffeemaker.php?name=" + escape(name) +
                          "&size=" + escape(size) +
                          "&beverage=" + escape(beverage) +
                          "&coffeemaker=1";

  sendRequest(url);
}
```

Функция sendRequest() задает функцию обратного вызова для ответа сервера (serveDrink) и отправляет запрос GET по URL, полученному от orderCoffee()

Функции для получения нужного объема и напитка мы напишем через пару страниц

Функция orderCoffee() строит URL запроса по имени сценария coffeemaker.php и информации, принятой с формы заказа кофе. В URL указано, что для приготовления кофе должна использоваться первая кофеварка

coffee.js

Одну минуту! Что такое? Вы кое-что забыли — а если первая кофеварка уже занята? И как насчет второй кофеварки? Когда мы будем ее использовать?

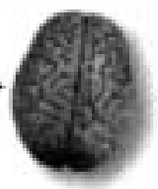
Какую кофеварку использовать?

Каждая кофеварка может готовить только одну порцию; что если первая кофеварка уже занята? Перед отправкой нового запроса мы должны каким-то образом проверить, равно ли состояние первой кофеварке «Idle». Если кофеварка занята, мы проверяем вторую. Если занята и вторая... что же, Джиму придется немного подождать.

Как организовать проверку состояния кофеварок? Вот как выглядит код HTML для первой кофеварки... Как вы думаете, что нужно сделать?

```
<div id="coffeemaker1">
  <h2>Coffee Maker #1</h2>
  <p><img src="images/CoffeeMaker1.gif"
          alt="Coffee Maker #1" /></p>
  <div id="coffeemaker1-status">Idle</div>
</div>
```

Код HTML для второй кофеварки выглядит почти так же



ШЕВЕЛИМ МОЗГАМИ

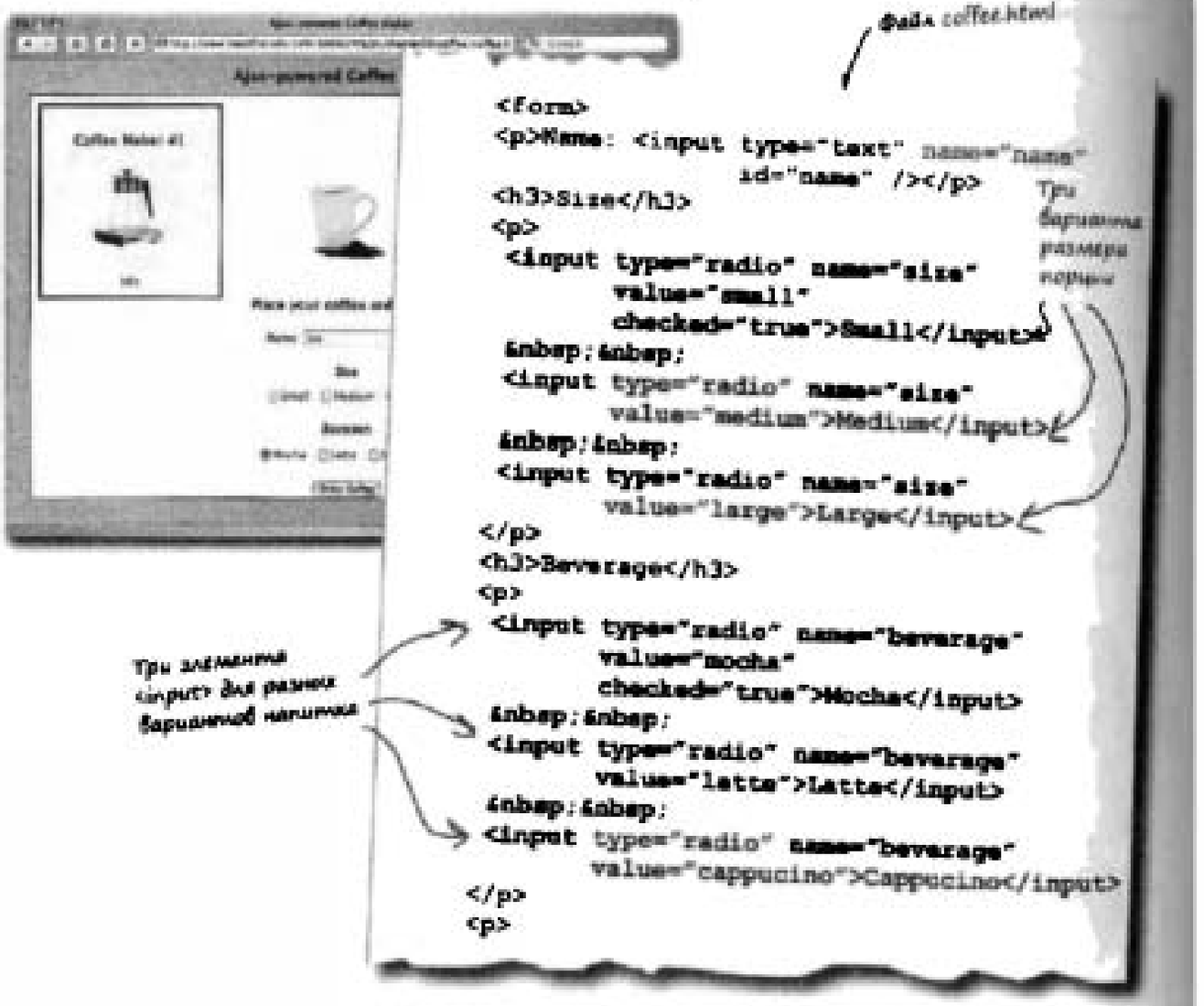
Как проверить состояние кофеварок? И как обновить состояние, когда кофеварка приступает к приготовлению кофе, или завершает его?

Подумайте над этими вопросами. Мы вернемся к ним позднее в этой главе

Определение размера порции и типа напитка

Прежде чем заняться получением и обновлением состояния двух кофеварок, мы должны написать код функций **getSize()** и **getBeverage()**.

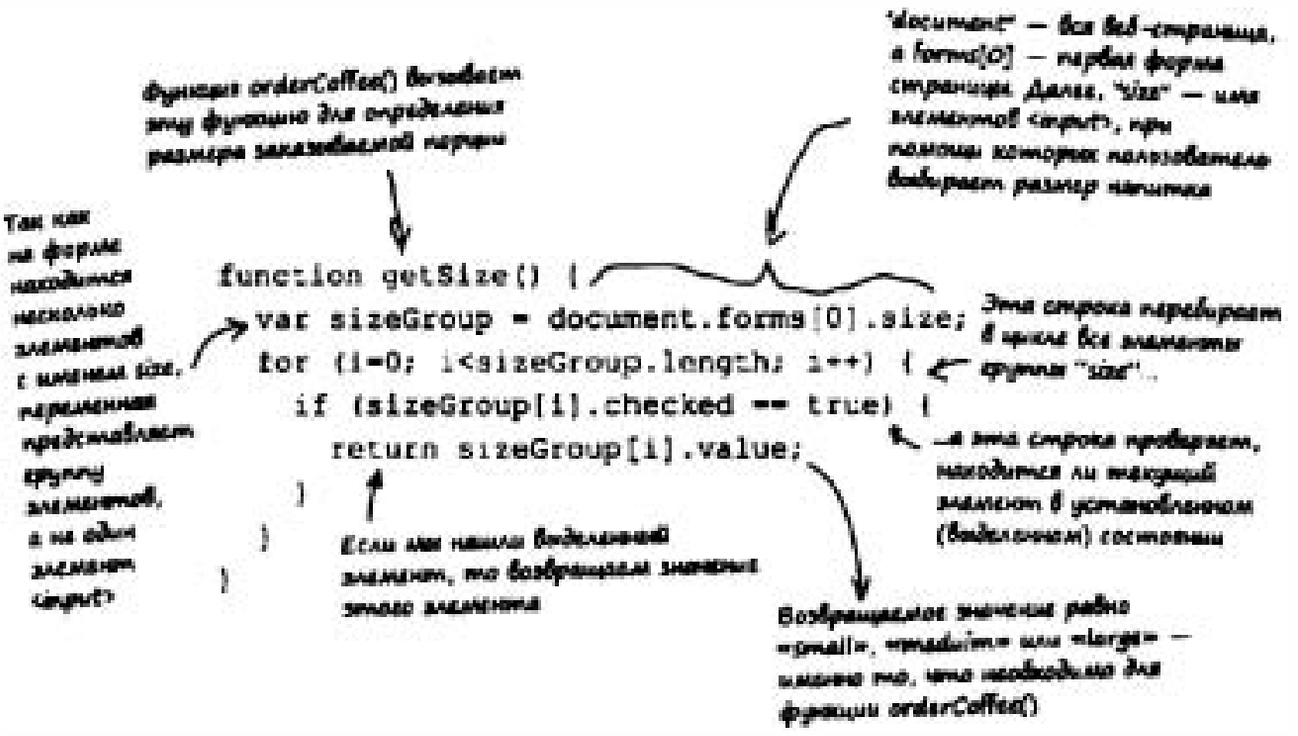
Вернемся к фирме заказа и посмотрим, о чем идет речь.



Но как узнать, какой размер порции и тип напитка выбрал Джим?

Определение значения группы переключателей

Чтобы определить, какой размер и тип напитка выбрал Джин, необходимо понять один принципиальный момент: на форме находятся три разных элемента **<input>** с одинаковыми именами **size** (и **beverage**). Следовательно, нам необходимо выяснить, какой из этих трех элементов установлен в настоящий момент.



Просто сделайте это

Теперь ваша очередь поработать с элементами **<input>**. Откройте файл **coffee.js** и добавьте в него функции **orderCoffee()**, **sendRequest()** и **getSize()**, рассматривавшиеся на нескольких последних страницах.

Затем необходимо написать функцию **getBeverage()**. Вероятно, вы уже представляете себе ее устройство после знакомства с кодом HTML из файла **coffee.html** и описанной ранее функцией **getSize()**. Когда и это будет сделано, сохраните файл coffee.html и перезагрузите страницу.

Беседа за чашкой кофе

Наши гости: Асинхронное Приложение (АП) и Синхронное Приложение (СП).

СП: Привет, АП... Давно не виделись.

АП: Ну еще бы... Каждый раз, когда я звоню получаю сигнал «занято». Можно подумать, что мы не родственники.

СП: У меня много дел, ты в курсе? И я не хочу, чтобы кто-нибудь отвлекал меня от обслуживания пользователей.

АП: Это понятно. А как насчет других пользователей, ожидающих в очереди?

СП: И до них очередь дойдет. Поймите, я учился у самой модели Клиент-Сервер. Она была просто великолепна; ее все любили за простоту и четкость, как меня.

АП: Странно. А мне кажется, что многие пользователи жалуются на долгое ожидание.

СП: По-моему, ты забываешь, кто здесь старший. Я живу на свете гораздо дольше тебя.

АП: Да, и твои пользователи .. Которые все идут и идут ...

СП: Никто не уделяет пользователям больше внимания, чем я. Просто я стараюсь полностью обслужить каждого из них. А ты тем временем бегаешь от одного к другому, и пытаешься обслужить всех сразу. Не выйдет!

АП: А мне кажется, что ты пытаешься обслужить своего ПОЛЬЗОВАТЕЛЯ.. одного. Всем остальным в это время приходится ждать.

СП: Иногда бывает лучше доделать одну работу до конца, а потом браться за следующую.

АП: Конечно, если кто-нибудь отправляет 200 блоков данных сразу. Но ведь большую часть времени ты фактически простаиваешь.

СП: Только потому, что я не разрешаю людям прерывать себя во время работы.

АП: Я умею слушать и говорить одновременно, а ты одержим одной мыслью.

СП: Я одержим? Просто я всегда завершаю начатое.

АП: Похвально, но сколько для этого потребуется времени? Десять секунд? Или десять минут? А может, час? Ты действительно думаешь, что пользователю нравится смотреть, как на экране вращаются песочные часы?

СП: Никто не жалуется.

АП: Еще бы. Ладно, я бы с удовольствием просидел здесь весь день, но мои пользователи не любят меня ждать. Это скорее по твоей части.

СП: Давай, наслаждайся своими 15 минутами славы, братец. Я уже миллион раз видел, как однодневки вроде тебя приходят и уходят.

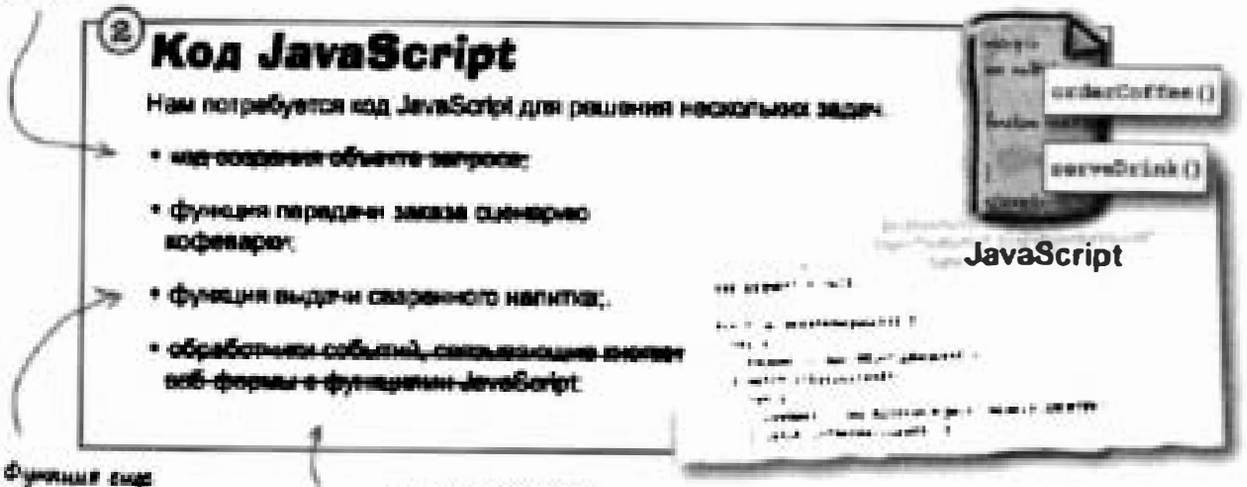
АП: Могу поспорить, что ты и про группу «U2» думал то же самое. Пока я никуда уходить не собираюсь. Увидимся, когда я захочу удалить тебя вызовом, братец...

Какой код JavaScript осталось написать?

Чтобы кофеварка работала так, как задумано, нам предстоит написать немало кода JavaScript. Давайте перечислим все функции и разберемся, что необходимо сделать в каждом случае.

Код уже написан и находится в index.js. Этот же код использовался в главе 2. Для создания объекта запроса в статическом коде JavaScript.

Мы уже знаем, как получить описание меню, но нам еще осталось проверить состояние кофеварки и определить, какой из них следует передать заказ.



Функция еще не написана. Она должна получить ответ сервера и вернуть кофеварку, завершившую работу, в состояние «idle»

Этот код уже сделан. Щелчок на кнопке «Order Coffee» запускает нашу функцию orderCoffee()

* Не забывайте — нам еще потребуется сценарий, выполняемый на сервере. Скоро доберемся и до него.

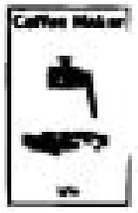
Чтение и запись текстового содержания в

Вернемся к функции **orderCoffee()**. Мы должны прочитать состояние первой кофеварки из элемента ** ** «coffeemaker1-status», и если оно представляет собой строку «Idle» — отправить запрос на приготовление кофе этой кофеварке. Если первая кофеварка занята, мы проверяем вторую кофеварку и элемент «coffeemaker2-status».

Вдобавок мы должны изменять состояние используемой кофеварки и показывать, что она занята выполнением заказа. Для получения или чтения текстового содержания элемента ** ** используется модель Document Object Model, или сокращенно DOM. Модель DOM значительно более подробно рассматривается в следующей главе, а пока мы просто воспользуемся [redacted] и завершим наше приложение с кофеварками.

Что мы хотим: получить текст элемента «coffeemaker1-status»

```
<div id="coffeemaker1">
 <h2>Coffee Maker #1</h2>
 <p><img src="images/CoffeeMaker1.gif"
         alt="Coffee Maker #1" /></p>
 <div id="coffeemaker1-status">Idle</div>
</div>
```



Чтобы узнать состояние кофеварки, следует получить содержимое элемента «coffeemaker1-status»

То же самое необходимо сделать для второй кофеварки, используя элемент «coffeemaker2-status»

Снова используем файл text-utils.js

Вспомните: значком [redacted] обозначается код, готовый к использованию. Просто введите его, а позднее мы все объясним — честное слово!

В главе 1 мы использовали файл JavaScript text-utils.js для обновления веб-отчета Кати. Сейчас мы воспользуемся тем же файлом для приложения, обслуживающего кофеварки. Для начала включите в заголовок файла coffee.html еще один элемент **<script>**, ссылающийся на text-utils.js:

После включения этой строки все вспомогательные функции из файла text-utils.js можно будет использовать в функциях JavaScript

```
<html>
 <head>
  <title>Ajax-powered Coffee Maker</title>
  <link rel="stylesheet" type="text/css" href="coffee.css" />
  <script type="text/javascript" src="ajax.js"> </script>
  <script type="text/javascript" src="text-utils.js"> </script>
  <script type="text/javascript" src="coffee.js"> </script>
 </head>
```

Проверка состояния кофеварки

Получив возможность пользоваться служебными функциями из файла text-utils.js, мы можем легко проверить текущее состояние кофеварки функцией **getText()**:

```
function orderCoffee() {
  var name = document.getElementById("name").value;
  var beverage = getBeverage();
  var size = getSize();

  var coffeemakerStatusDiv1 =
   document.getElementById("coffeemaker1-status");
  var status = getText(coffeemakerStatusDiv1);
  if (status == "Idle") {
    // Update the coffee maker's status
    var url = "coffeemaker.php?name=" + escape(name) +
                              "&size=" + escape(size) +
                              "&beverage=" + escape(beverage) +
                              "&coffeemaker=1";
    sendRequest(url);
  }
```

Получение элемента , содержащего информацию о состоянии первой кофеварки

Функция getText() возвращает текст, хранящийся в элементе coffeemaker1-status

Если первая кофеварка не занята, мы отправляем ей заказ на приготовление кофе

Нам еще предстоит определить, как обновлять информацию о состоянии кофеварки

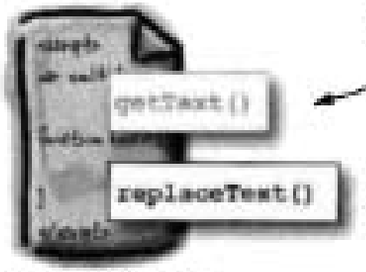
Запись текста в элемент

Итак, вы знаете, как получить текст с состоянием кофеварки из элемента ; остается лишь разобраться, как записать нужный текст в этот элемент. Для этой цели можно воспользоваться другой вспомогательной функцией из файла text-utils.js, которая называется replaceText().

```
function makeCoffee() {
  var name = document.getElementById("name").value;
  var beverage = getBeverage();
  var size = getSize();

  var coffeemakerStatusDiv =
    document.getElementById("coffeemaker-status");
  var status = getText(coffeemakerStatusDiv);
  if (status == "idle") {
    replaceText(coffeemakerStatusDiv, "Brewing " +
                name + "'s " +
                size + " " + beverage);
    var url = "coffeemaker.php?name=" + escape(name) +
                                "&size=" + escape(size) +
                                "&beverage=" + escape(beverage) +
                                "&coffeemaker=1";
    sendRequest(url);
```

Если первая кофеварка свободна, эта команда обновляет ее состояние и показывает, что кофеварка занята приготовлением напитка.

А эти команды отправляют запрос сценарию PHP на сервере



text-utils.js

Функция getText() возвращает текстовое содержимое или любого другого элемента, который передается ей при вызове

Функция replaceText() получает элемент и текст, который должен быть в него помещен. При этом из элемента удаляется все текущее содержимое, так что переданный вами текст становится единственным содержимым элемента

Мы еще не рассматривали модель DOM, поэтому постарайтесь хотя бы в общих чертах представьте, что делает код JavaScript из файла text-utils.js. Модели DOM будет подробно описана в следующей главе.

Javascript ...с первого взгляда

Посмотрите, как выглядит код text-utils.js. Еще раз напомним, что все подробности будут объяснены в главе 4

```
function replaceText(el, text) {
  if (el != null) {
    clearText(el);
    var newNode = document.createTextNode(text);
    el.appendChild(newNode);
  }
}

function clearText(el) {
  if (el != null) {
    if (el.childNodes) {
      for (var i = 0; i < el.childNodes.length; i++) {
        var childNode = el.childNodes[i];
        el.removeChild(childNode);
      }
    }
  }
}

function getText(el) {
  var text = "";
  if (el != null) {
    if (el.childNodes) {
      for (var i = 0; i < el.childNodes.length; i++)
{
        var childNode = el.childNodes[i];
        if (childNode.nodeValue != null) {
          text = text + childNode.nodeValue;
        }
      }
    }
  }
  return text;
}
```

replaceText() — функция общего назначения, которая получает объект элемента (например,) и строку, и заменяет текстовое содержимое элемента полученным текстом

Функция replaceText() вызывает clearText() для очистки текущего содержимого элемента, а затем назначает новое текстовое содержимое элемента

Будьте внимательны! Функция clearText() стирает все содержимое элемента, включая вложенные элементы и текст

Функция getText() возвращает текстовое содержимое заданного элемента

getText() объединяет весь текст элемента в одну строку, которая и возвращается функцией

НЕ ЗАБЫВАЙТЕ: если что-то в этом коде осталось непонятным, это нормально

Разложите по полкам

С помощью файла text-util.js функция sendCoffee() проверяет, свободна ли первая кофеварка, и если свободна — передает на нее заказ. А если первая кофеварка занята? В этом случае код JavaScript должен проверить, свободна ли вторая кофеварка, и если свободна — передать заказ ей. В противном случае следует сообщить пользователю, что он должен подождать освобождения одной из двух занятых кофеварок.

Далее приводится код функции sendCoffee() с несколькими пропусками. Ваша задача — завершить код JavaScript для проверки и использования обеих кофеварок. Расставьте правильные фрагменты по пропускам в коде функции.

```
function orderCoffee() {
  var name = document.getElementById("name").value;
  var beverage = getBeverage();
  var size = getSize();

  var coffeemakerStatusDiv1 =
   document.getElementById("coffeemaker1-status");
  var status = getText(coffeemakerStatusDiv1);
  if (status == "Idle") {
    replaceText(coffeemakerStatusDiv1, "Brewing " +
                name + "'s " +
                size + " " + beverage);
    var url = "coffeemaker.php?name=" + escape(name) +
                              "&size=" + escape(size) +
                              "&beverage=" + escape(beverage) +
                              "&coffeemaker=1";
    sendRequest(url);
  } else {
```

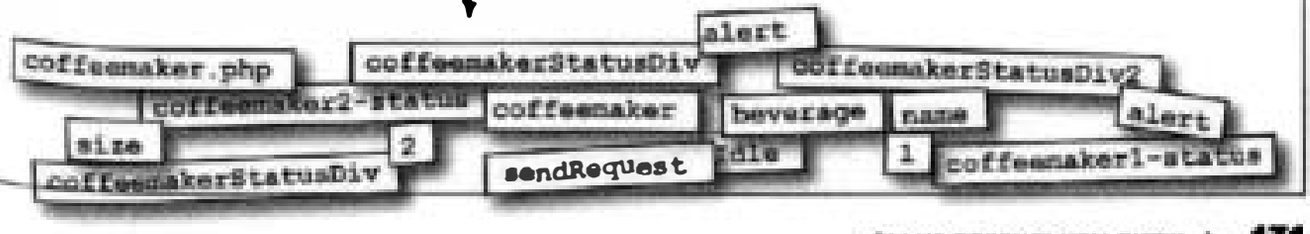
Весь код на соседней странице является частью блока «else»

Секция «else» выполняется в том случае, если состояние первой кофеварки отлично от «Idle»

Если вы не уверены во фрагментах кода, которые надо здесь использовать, то взгляните на код JavaScript для первой кофеварки. Код для второй кофеварки гораздо проще

```
var _____ =
  document.getElementById("_____");
status = getText(_____);
if (status == "_____") {
  replaceText(_____,
              "Brewing " + _____ + "'s " +
              _____ + " " + _____);
  var url = "_____?_____=" + escape(_____) +
                          "&_____=" + escape(_____) +
                          "&_____=" + escape(_____) +
                          "&_____=___";
  _____(url);
} else {
  _____("Sorry! Both coffee makers are busy. " +
        "Try again later.");
}
}
}
```

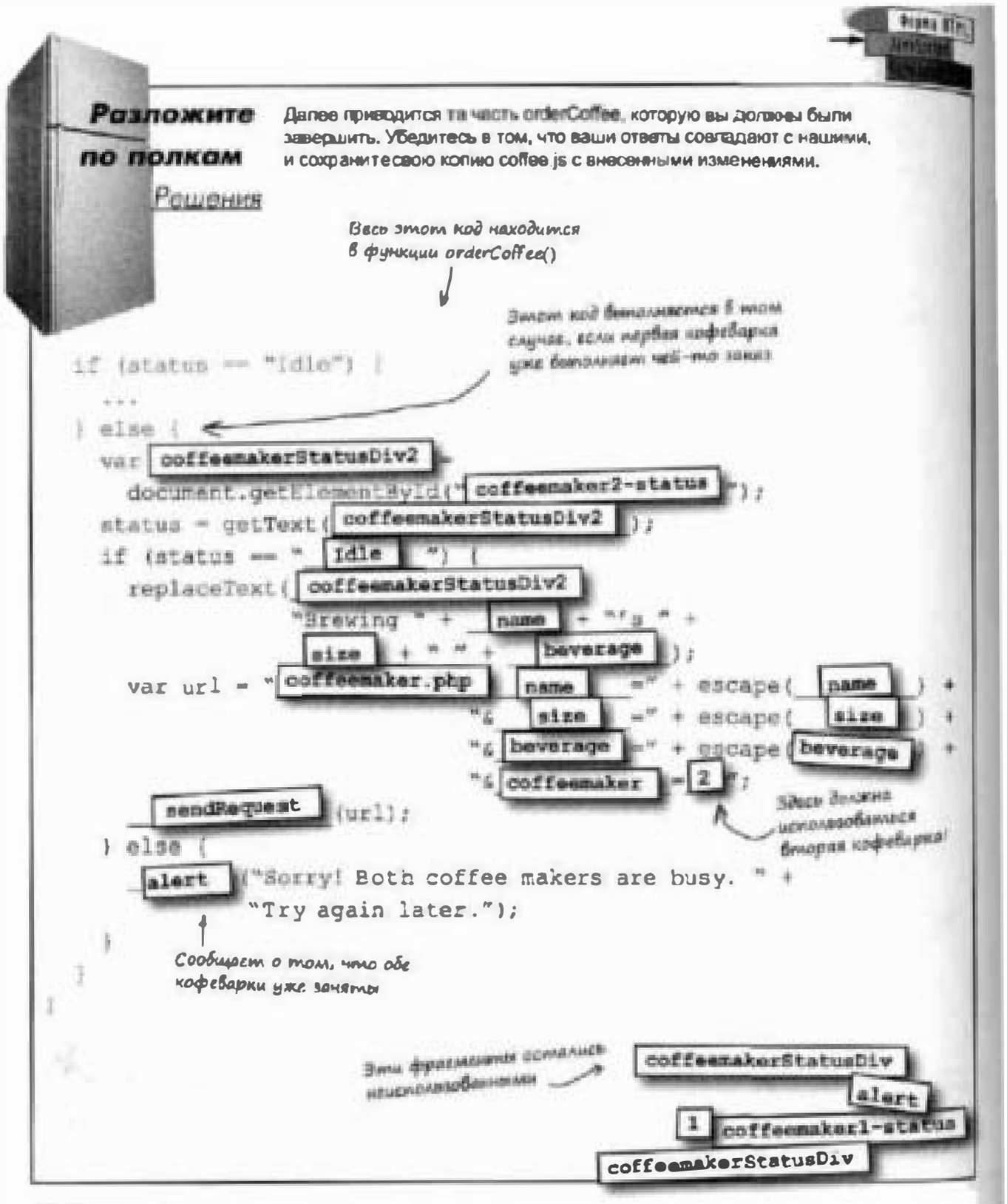
При необходимости фрагменты можно использовать многократно



Разложите по полкам

Решения

Далее приводится та часть orderCoffee, которую вы должны были завершить. Убедитесь в том, что ваши ответы совпадают с нашими, и сохраните свою копию coffee.js с внесенными изменениями.



Пробный запуск

Нам еще предстоит проделать немалую работу, и все же давайте попробуем запустить наше приложение. Убедитесь в том, что функции **sendRequest()**, **getSize()**, **getBeverage()** и **orderCoffee()** были включены в файл **coffee.js**, и загрузите **coffee.html** в браузере.

Введите заказ и посмотрите, что при этом происходит. Затем введите другой заказ... Используются ли обе кофеварки?

ШЕВЕЛИМ МОЗГАМИ

Обратите внимание: после размещения заказа имя заказчика и описание напитка остаются на форме. Может ли это создать проблемы? Как вы думаете, что следует сделать для предотвращения путаницы?

По-моему, нехорошо, когда во время приготовления напитка описание заказа продолжает отображаться на форме. Нельзя ли очистить форму перед размещением заказа?

Очистка формы при размещении заказа

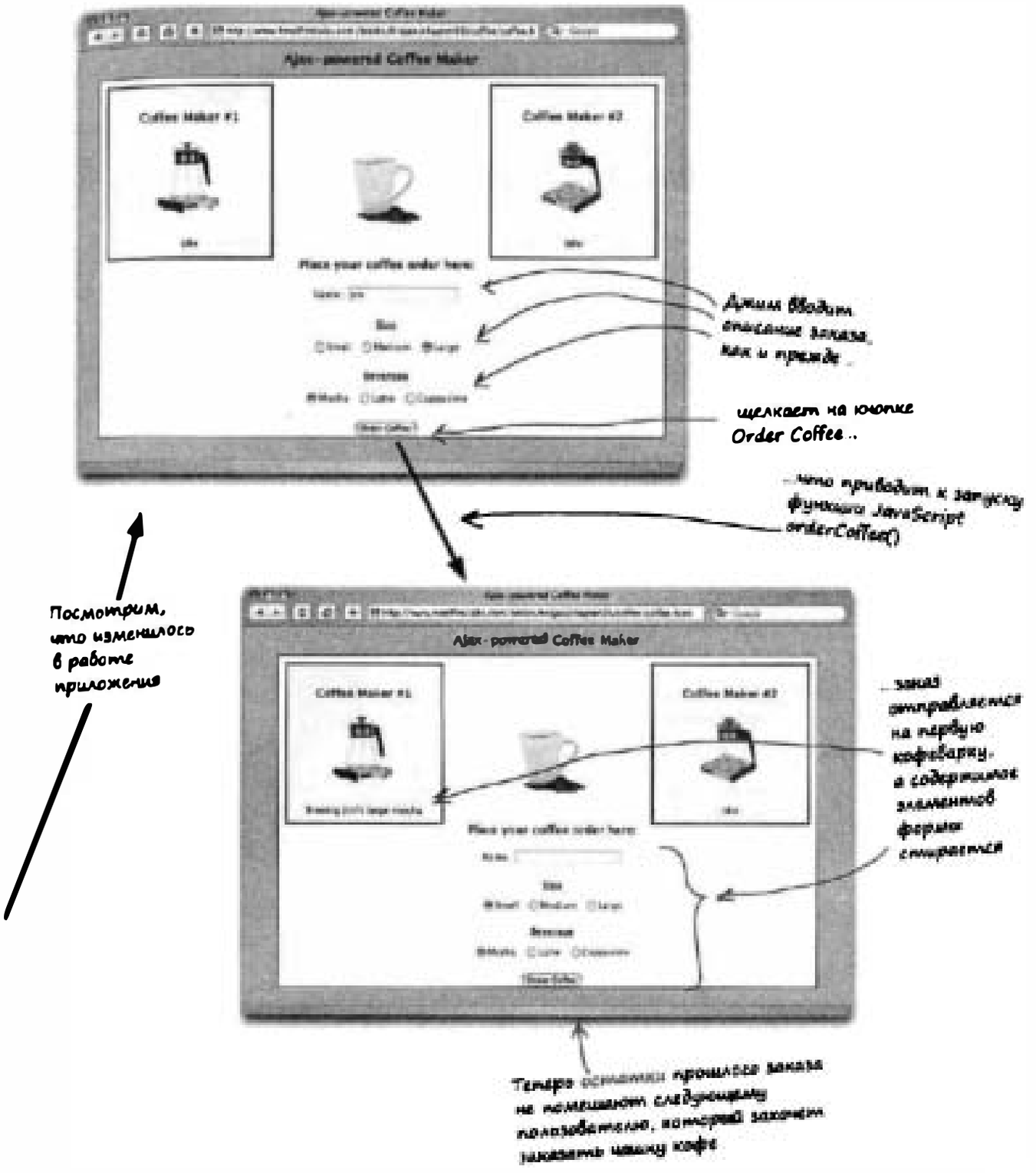
Сейчас ваши коллеги вводят описание заказа и щелкают на кнопке Order Coffee. Заказ отправляется на кофеварку, и начинается приготовление напитка. Но описание заказа остается на форме, и это может сбить с толку других пользователей. Было бы гораздо удобнее, если бы после отправки заказа на кофеварку его описание стиралось с формы.

Для очистки формы можно воспользоваться методом `reset()`. Давайте дополним функцию `orderCoffee()` в файле *coffee.js* парой команд, очищающих форму после отправки заказа:

```
var status = getText(coffeemakerStatusDiv1);
if (status == "Idle") {
  replaceText(coffeemakerStatusDiv1,
              "Brewing " + name + "'s " +
              size + " " + beverage);
  document.forms[0].reset();
  var url = "coffeemaker.php?name=" + escape(name) +
                                "&size=" + escape(size) +
                                "&beverage=" + escape(beverage) +
                                "&coffeemaker=1";
  sendRequest(url);
} else {
  var coffeemakerStatusDiv2 =
    document.getElementById("coffeemaker2-status");
  status = getText(coffeemakerStatusDiv2);
  if (status == "Idle")
    replaceText(coffeemakerStatusDiv2,
                "Brewing " + name + "'s " +
                size + " " + beverage);
  document.forms[0].reset();
  var url = "coffeemaker.php?name=" + escape(name) +
                                "&size=" + escape(size) +
                                "&beverage=" + escape(beverage) +
                                "&coffeemaker=2";
```

Мы обращаемся с вопросом на приготовление кофе в 26-й строке листинга coffee.js

В обоих случаях для обновления первой формы без-отправки вызывается метод reset()


Ajax-powered Coffee Maker
Coffee Maker #1
Coffee Maker #2
Place your coffee order here:
Name:
Size
Small
Medium
Large
Flavors
Mocha
Latte
Cappuccino
Order Coffee
Данные вводим описанные заказы, как и прежде.
щелкает на кнопке Order Coffee..
...что приводит к запуску функции JavaScript orderCoffee()
Посмотрим, что изменилось в работе приложения
Ajax-powered Coffee Maker
Coffee Maker #1
Coffee Maker #2
Place your coffee order here:
Name:
Size
Small
Medium
Large
Flavors
Mocha
Latte
Cappuccino
Order Coffee
...заказ отправляется на первую кофеварку - а содержимое заполненной формы очищается
Теперь [illegible] принимает заказы на напитки следующему пользователю, который захочет заказать чашку кофе

Научим сервер домовёнка кофе, чтобы организовать его бедному вашим уполномоченным коллегам

PHP...с первого взгляда

Сценарий PHP просто организует циклическую задержку, имитирующую процесс приготовления кофе. Вот как выглядит код в файле coffeemaker.php:

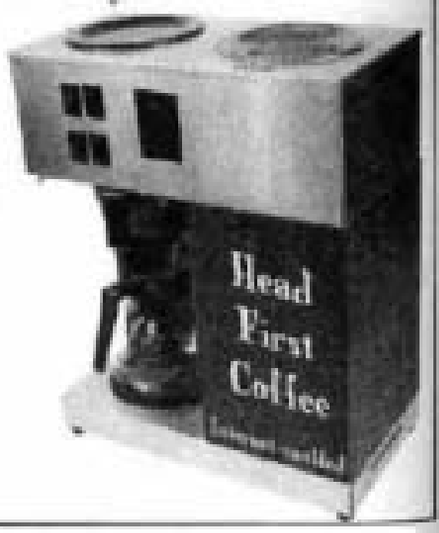
```
<?php

$name = $_REQUEST['name'];
$size = $_REQUEST['size'];
$drink = $_REQUEST['drink'];
$coffeemaker = $_REQUEST['coffeemaker'];

for ($i = 0; $i < 50000000; $i++) {
  // brewing
}

echo $coffeemaker . $name;
?>
```

Сначала сценарий получает все данные, переданные в составе URL запроса

Здесь вы можете подключить собственный код управления кофеваркой

Цикл просто организует задержку... и имитирует приготовление кофе

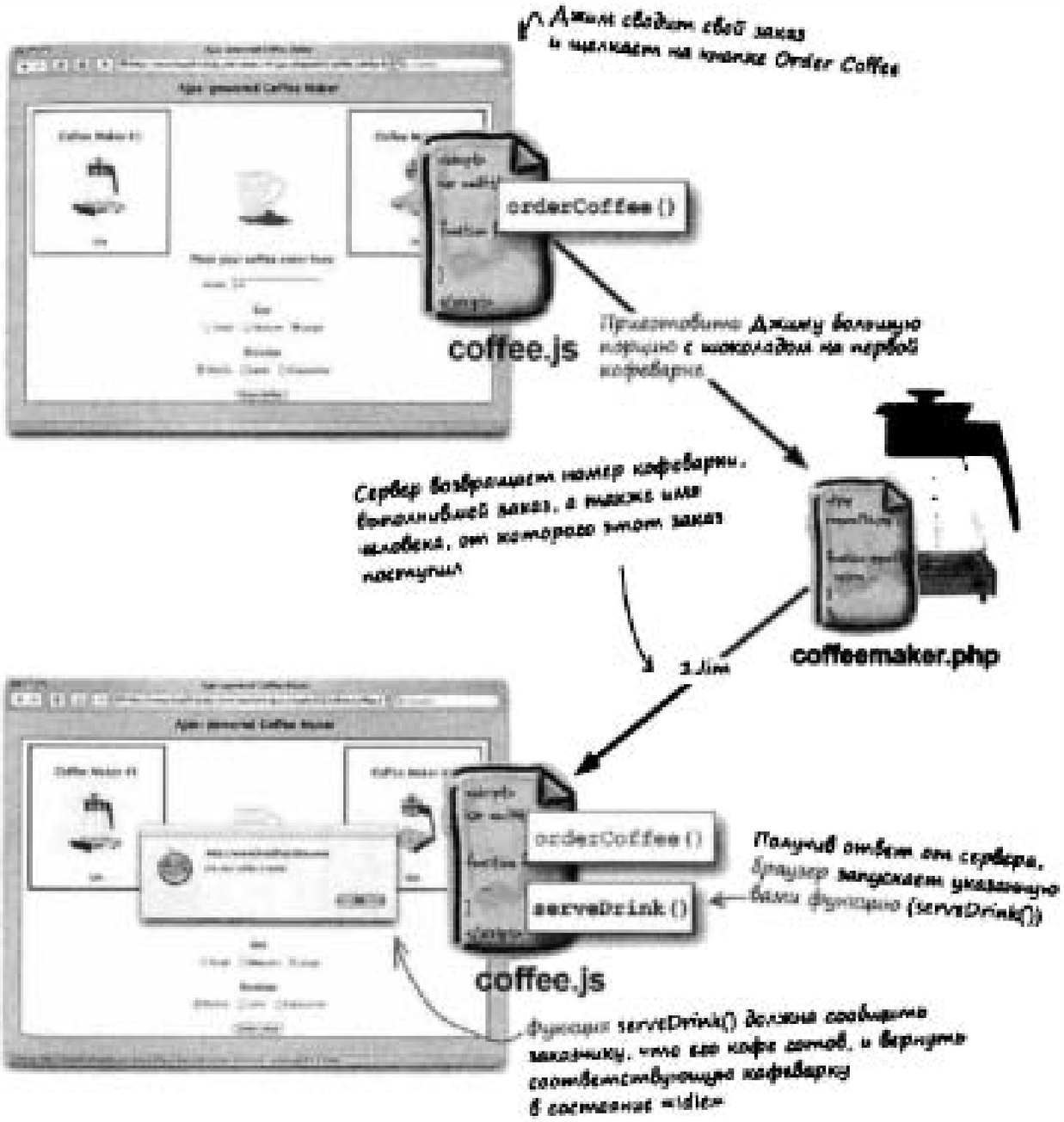
Когда напиток будет готов, сервер возвращает номер освободившейся кофеварки и имя человека, для которого поступил заказ. Например, ответ может иметь вид «1_Jim», если первая кофеварка приготовила кофе для Джима, или «2_Mary», если вторая кофеварка выполнила заказ Мэри



Загрузите сценарий coffeemaker.php с сайта, или обратитесь к листингу в приложении 3

Что делать с ответом сервера?

Функция **orderCoffee()** готова, как и сценарий coffeemaker.php. При отправке запроса на одну из кофеварок сценарий coffeemaker.php отрабатывает и возвращает выходные данные: номер кофеварки, завершившей заказ, а также имя заказчика.



Пишем функцию обратного вызова

Нам осталось написать только функцию **serveDrink()**; эта функция получает ответ от сервера и определяет, кто отправил заказ, и какая кофеварка использовалась для его выполнения. Затем **serveDrink()** переводит кофеварку в состояние «Idle» и сообщает заказчику о том, что его кофе готов.

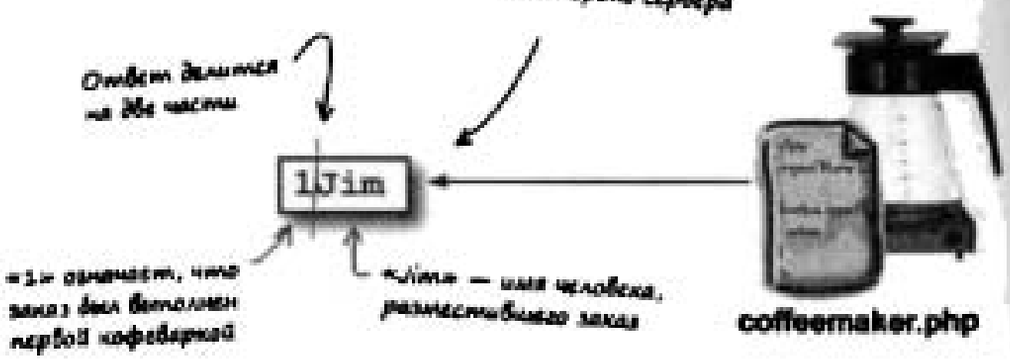
Начнем с проверки состояния готовности, статуса HTTP и получения ответа от сервера.

```
function serveDrink() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      var response = request.responseText;

      // Figure out who placed the order, and
      //   which coffee maker was used
    } else
      alert("Error! Request status is " + request.status);
  }
}
```

Вероятно, этот код был уже хорошим знаком. Он будет присутствовать почти во всех функциях обратного вызова, которые вам предстоит написать

Интерпретация ответа сервера

Ответ сервера, принятый функцией обратного вызова **serveDrink()**, выглядит примерно так:



функция JavaScript substring()

В JavaScript имеется функция, которая идеально подходит для разбиения возвращаемой строки на составляющие. Вот как эта функция используется на практике:

```
var newString = myString.substring(startIndex, endIndex);
```

substring() — функция JavaScript, при помощи которой можно разбить строку на несколько подстрок

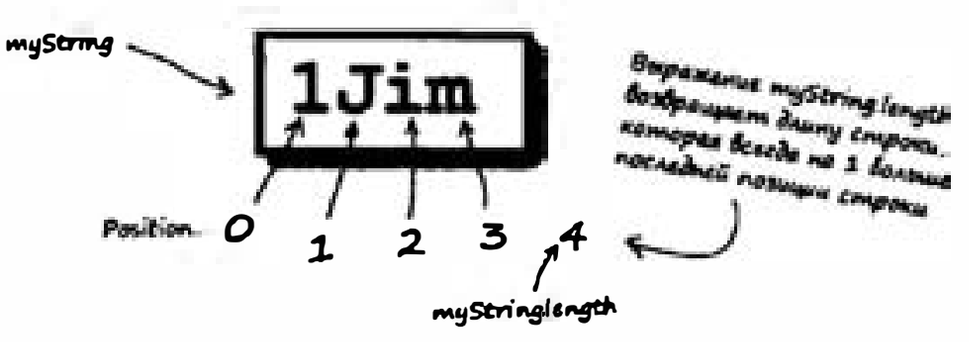
startIndex — позиция myString, с которой должна начинаться подстрока. Помните, что первому символу соответствует позиция "0", а не "1".

newString — новая подстрока, возвращаемая JavaScript

Переменная myString содержит строку, разбиваемую на подстроки

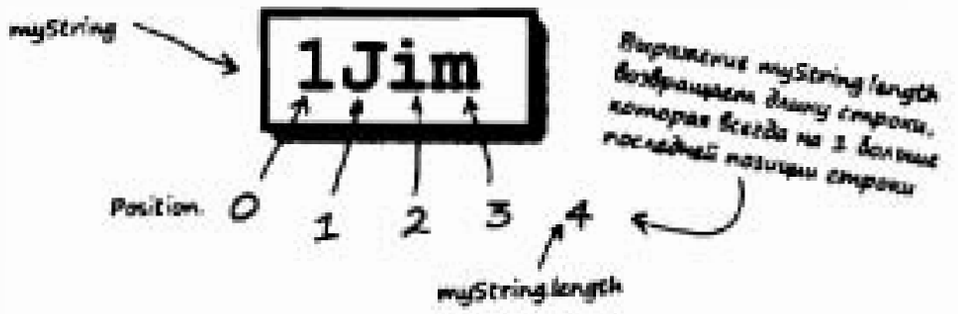
endIndex — конечная позиция подстроки в myString. Символ, находящийся в этой позиции, не включается в подстроку

Возможно, вам мы будем удобнее считать, что substring() создаст подстроку от позиции startIndex до позиции (endIndex − 1)



Применение substring()

Давайте немного потренируемся в использовании метода JavaScript **substring()** для строки, возвращаемой сервером.



Подстрока начинается в позиции 1, завершается в позиции 3 и не включает символ в позиции 3

Подстрока начинается в позиции 1 («J») и первый в позиции (3–1), то есть 2 («i»)

```
myString.substring(1, 3);
```

→ Ji

Поскольку myString.length на 1 больше последней позиции, подстрока распространяется до конца строки

```
myString.substring(2, myString.length);
```

→ im

Позиция 0 соответствует первому символу в строке

myString.length захватывает все символы до конца строки

Следовательно, этот код JavaScript всегда возвращает всю исходную строку

```
myString.substring(0, myString.length);
```

→ 1Jim

Просто сделайте это

Сейчас вы уже вполне готовы самостоятельно дописать оставшийся код JavaScript-функции **serveDrink()**. Ниже приводится соответствующая часть кода; ваша задача — заполнить пропуски и сделать так, чтобы программа заработала.
Для этого необходимо получить ответ с сервера и разбить его на две составляющие: номер кофеварки и имя заказчика. Затем функция должна перевести кофеварку, завершившую выполнение заказа, в свободное состояние («Idle»), и сообщить заказчику о том, что его напиток готов.

```
function serveDrink() {
  if (request.readyState == _____) {
    if (request.status == _____) {
      var response = request.responseText;
      var whichCoffeemaker = response.substring(___, ___);
      var name = response.substring(_____, _____);
      if (whichCoffeemaker == "1") {
        var coffeemakerStatusDiv1 =
          document.getElementById("_____");
        replaceText(_____, "Idle");
      } else {
        var coffeemakerStatusDiv2 =
          document.getElementById("_____");
        replaceText(_____, "_____");
      }
      _____(name + ", your coffee is ready!");
    } else
      alert("Error! Request status is " + request.status);
  }
}
```

Воспользуйтесь тем, что вы узнали об использовании функции substring()

coffee.js

Доделываем функцию serveDrink()

Потратим еще немного времени на освоение метода **substring()** JavaScriptmethod, используя строки с последних двух страниц.

```
function serveDrink() {
  if (request.readyState == [illegible]) {
    if (request.status == [illegible]) {
      var response = request.responseText;
      var whichCoffeemaker = response.substring([illegible], [illegible]);
      var name = response.substring([illegible], [illegible]);
      if (whichCoffeemaker == "1") {
        var coffeemakerStatusDiv1 =
          document.getElementById("[illegible]");
        replaceText([illegible], "Idle");
      } else {
        var coffeemakerStatusDiv2 =
          document.getElementById("[illegible]");
        replaceText([illegible], "[illegible]");
      }
      [illegible](name + ", your coffee is ready!");
    } else
      alert("Error! Request status is " + request.status);
  }
}
```

Эти две строки уже можно считать стандартными

Чтобы узнать номер кофеварки, достаточно отделить один символ от начала строки

Имя, возвращаемое сервером, начинается со второй позиции («1») и следует до конца строки

Код обновляет состояние кофеварки, завершившей работу

Команда сообщает отправителю о том, что его заказ готов

Часто задаваемые вопросы

В: Почему мы не отображаем состояние кофеварок в элементах <form>? Разве это не проще, чем работа с моделью DOM?

О: Конечно, для отображения состояния кофеварок можно было воспользоваться элементами форм. Однако элементы формы предполагают, что пользователь может вводить в них данные. Информация о состоянии кофеварки предназначена исключительно для вывода — пользователь не должен изменять ее самостоятельно, поэтому в данном случае логичнее воспользоваться полями, для которых редактирование запрещено (то есть предназначенными только для вывода текста).

А это означает, что нам придется прибегнуть к модели DOM... впрочем, как вы увидите в следующей главе, модель DOM не так уж сложна. Стоит понять, как браузер на самом деле воспринимает разметку HTML, и вы вскоре начнете писать собственный код HTML. Так что оставайтесь с нами...

В: Не существует ли более простого способа чтения и записи текста в элементах ? Я читал о свойстве innerHTML, которое позволяет записать в произвольный код HTML. Нельзя ли воспользоваться этим свойством?

О: Свойство innerHTML не рекомендуется использовать для чтения и записи содержимого элементов. Оно не входит в спецификацию DOM, и консорциум W3C объявил его устаревшим — возможно, в будущем оно даже не будет поддерживаться некоторыми браузерами. Но что еще хуже, некоторыми браузерами оно не поддерживается даже сейчас!

Гораздо безопаснее использовать код DOM, как в нашем приложении. В следующей главе мы в деталях рассмотрим работу кода DOM. Написав несколько функций, использующих модель DOM, вы увидите, как это просто. А самое лучшее — что эта модель поддерживается на любой платформе, на которой существуют веб-браузеры!

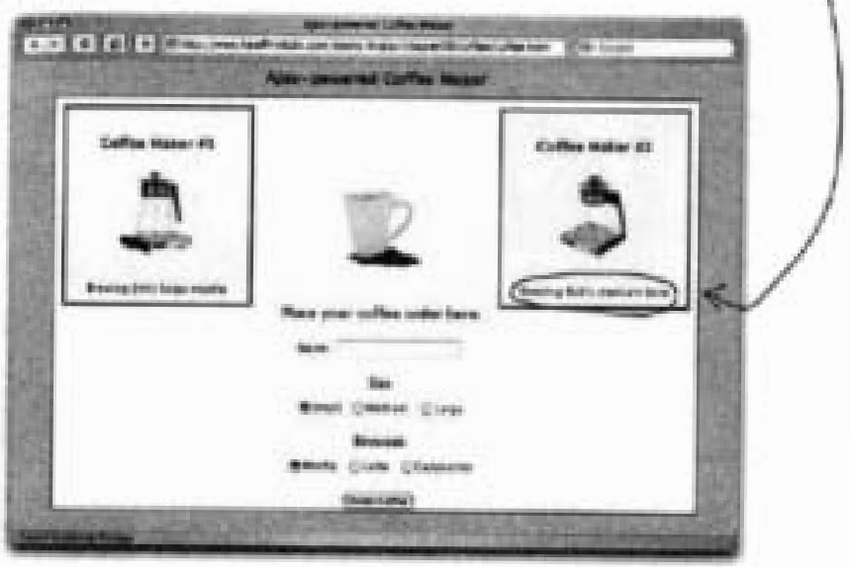
Последняя проверка (или все-таки...?)

Похоже, мы успешно написали весь код JavaScript для наших кофеварок. Проверьте, что вы следовали всем описанным примерам и сохранили свои изменения в файлах coffee.html и coffee.js. Загрузите файл coffee.html в браузере и приготовьтесь испытать приложение в деле.

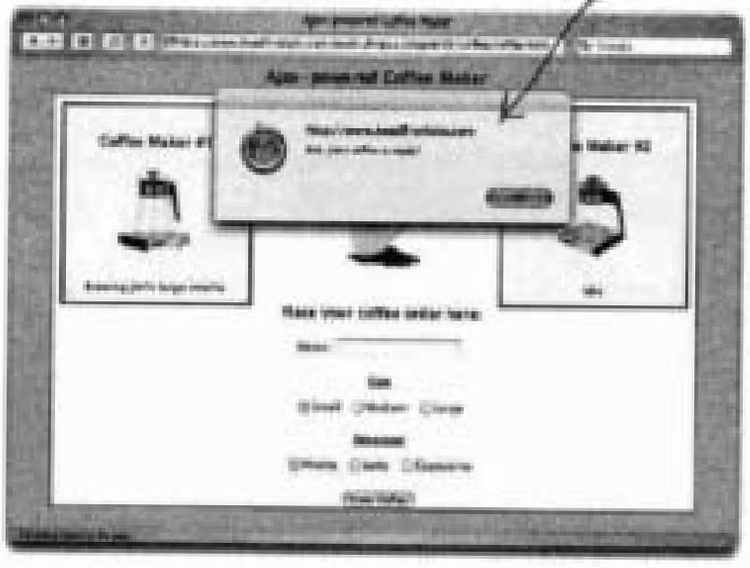
- Введите заказ для Джима: он хочет большой кофе с шоколадом. Щелкните на кнопке Order Coffee.

- Заказ Джима начинает выполняться на первой кофеварке.

- Теперь введите следующий заказ — Боб хочет средний кофе с молоком. Щелкните на кнопке Order Coffee; заказ Боба начинает выполняться на второй кофеварке



Заказ Боба был выполнен первым, хотя поступил вторым

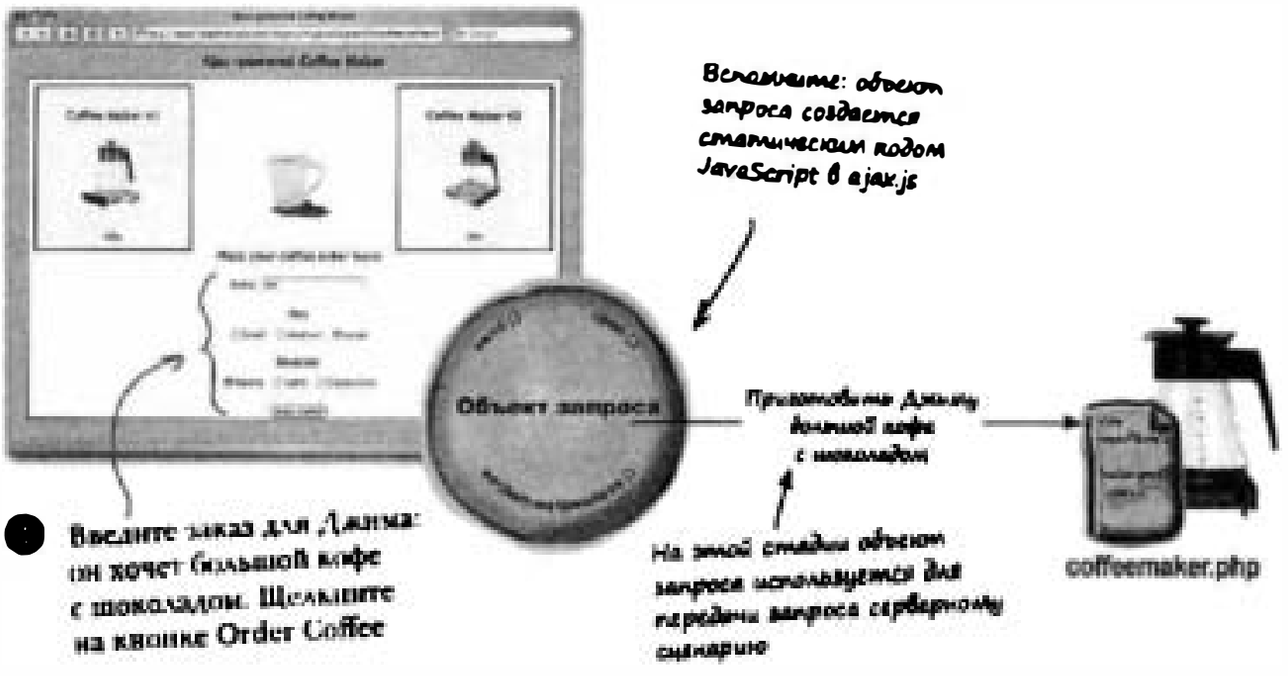


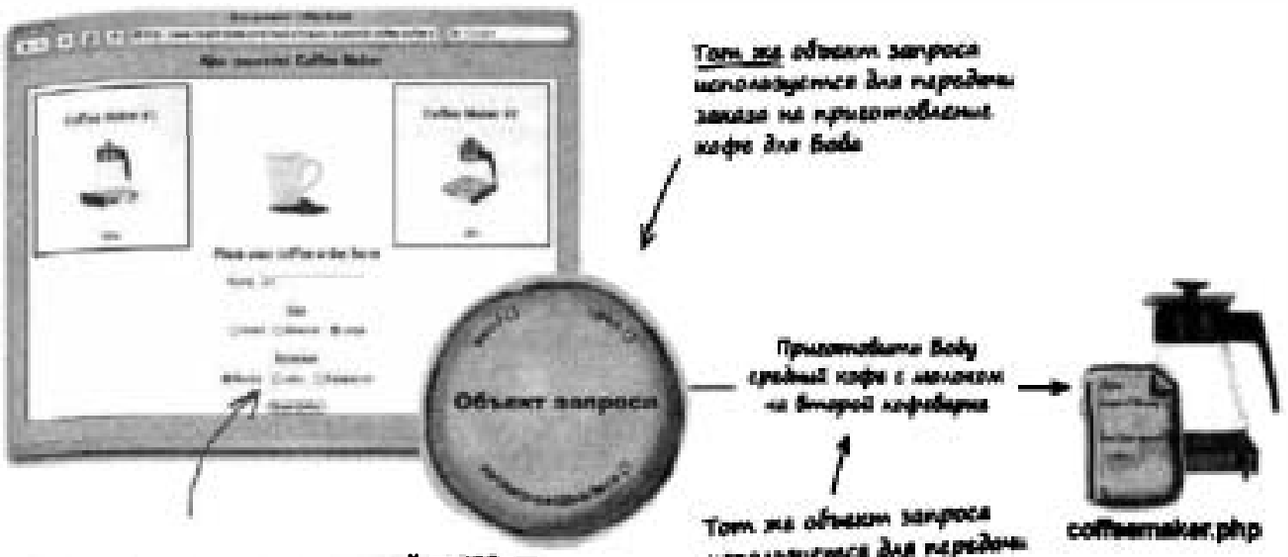
- Подождите, пока кофеварки завершат работу. Чей заказ был выполнен первым? Завершились ли оба заказа? Соответствует ли результат вашим предположениям?

Что же случилось с заказом Джима?

Давайте присмотримся к объекту запроса

Чтобы понять, что произошло с заказом Джима, давайте повнимательнее проанализируем каждый шаг нашего тестового запуска, и разберемся, что же в действительности происходит с объектом запроса.

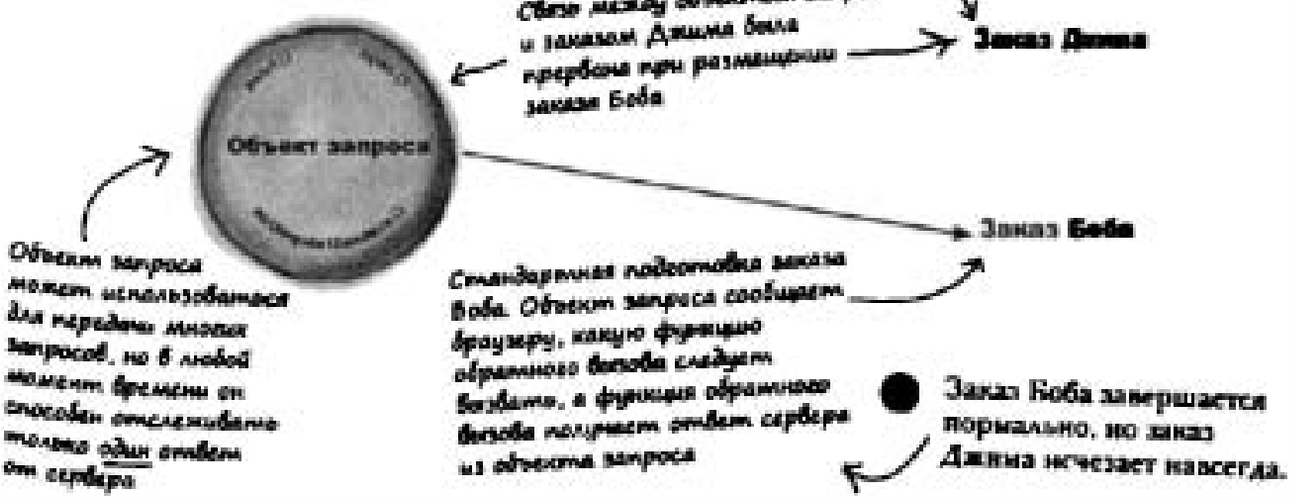




● Теперь введите следующий заказ — Боб хочет средний кофе с молоком. Щелкните на кнопке Order Coffee; заказ Боба начинает выполняться на второй кофеварке.

Но как насчет заказа Джима?

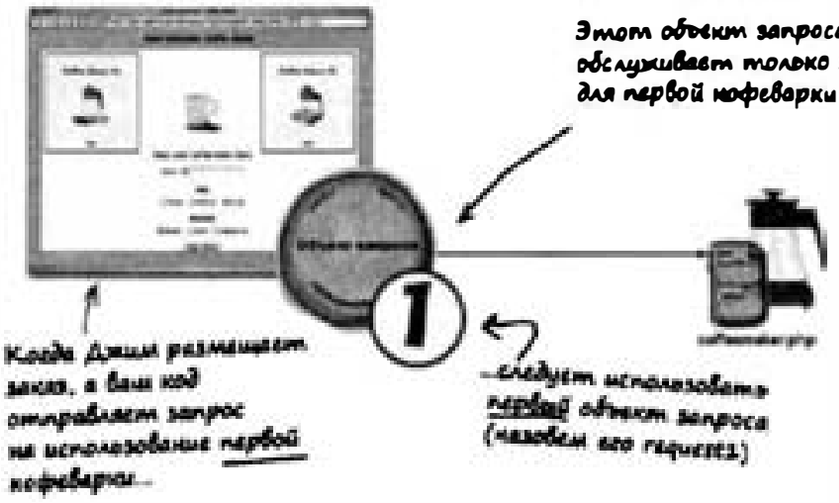
Видите, что произошло? Кофе для Джима все еще готовится, но с запросом теперь не связан объект! Заказ Боба заменил в объекте запроса информацию, относящуюся к заказу Джима:

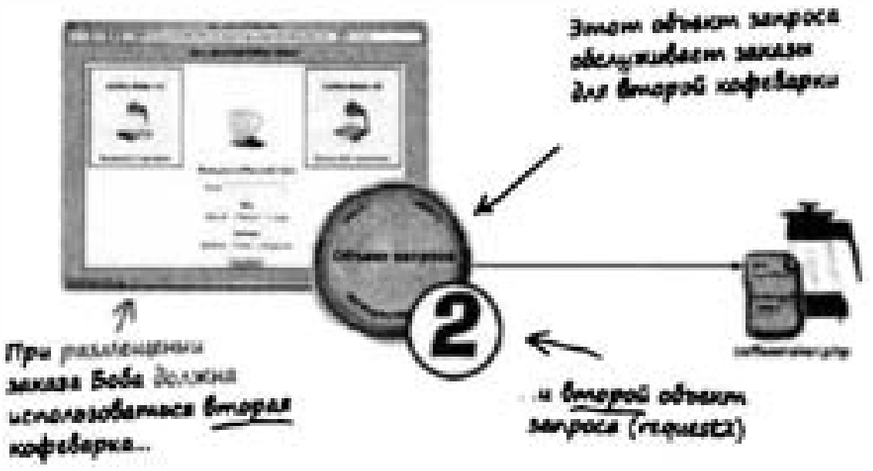


Пора снова браться за код JavaScript

Нам нужны два объекта запроса!

Похоже, с одним объектом запроса мы не сможем организовать обработку двух заказов одновременно. Посмотрим, нельзя ли решить проблему, с которой мы столкнулись, посредством использования двух разных объектов:





Создание двух объектов запроса

Для начала необходимо создать оба объекта. Откройте файл ajax.js и внесите в него следующие изменения, чтобы вместо одного объекта запроса создавались два:

Файл ajax.js

```
var request = null;

function createRequest() {
  var request = null;
  try {
    request = new XMLHttpRequest();
  } catch (trymicrosoft) {
    try {
      request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (othermicrosoft) {
      try {
        request = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (failed) {
        request = null;
      }
    }
  }

  if (request == null) {
    alert("Error creating request object!");
  } else {
    return request;
  }
}

var request1 = createRequest();
var request2 = createRequest();
```

Мы откажемся от создания одного объекта запроса. Удалите эту строку

Теперь преобразуйте этот код в функцию, чтобы его было удобно вызывать повторно

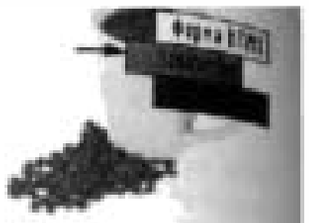
Если объект запроса создан успешно, мы возвращаем его как результат работы функции

Напоследок в статическом коде JavaScript создаются два объекта запроса. Каждому объекту присваивается возвращаемое значение функции createRequest()

После выполнения этого фрагмента мы имеем два объекта запроса, созданных и готовых к использованию — request1 и request2

ajax.js

Использование двух объектов запроса



После создания объектов запросов необходимо внести изменения в функцию **orderCoffee()** так, чтобы она работала с двумя объектами вместо одного. Первый объект **request1** будет использоваться для отправки всех запросов первой кофеварке, а второй объект **request2** — для отправки всех запросов второй кофеварке...

Я думаю, перед изменением orderCoffee() необходимо изменить sendRequest(). Разве функция sendRequest() не должна использовать оба объекта запроса?

Да, начнем с обновления sendRequest()

Давайте изменим функцию **sendRequest()** так, чтобы в одном из параметров передавался объект запроса; этот объект должен использоваться для передачи запроса заданному URL. В этом случае нам не придется дважды программировать отправку запроса... достаточно вызвать **sendRequest()** и передать ей нужный объект.

Отредактируйте свою версию функции sendRequest() в файле coffee.js

В sendRequest() ничего больше изменять не придется

```
funct ion sendRequest(request, url) {
  request.onreadystatechange = serveDrink;
  request.open("GET", url, true);
  request.send(null);
}
```

В этих строках используется объект запроса, переданный sendRequest() при вызове. Не забывайте, что теперь используются два объекта запроса

Оба объекта запроса инициализируются одинаковыми параметрами подключения и функцией обратного вызова. Различаются только URL запросов

Обновление orderCoffee()

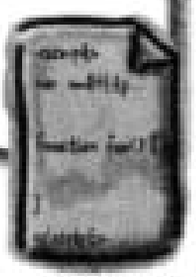
Функция **sendRequest()** теперь обеспечивает доставку запроса нужной кофеварке; остается лишь передать ей правильный объект запроса в функции **orderCoffee()**. Для этого достаточно внести два небольших изменения:

Внесите оба изменения в свою версию orderCoffee() в функции coffee.js

```
var status = getText(coffeemakerStatusDiv1);
if (status == "Idle") {
  replaceText(coffeemakerStatusDiv1,
              "Brewing " + name + "'s " +
              size + " " + beverage);
  document.forms[0].reset();
  var url = "coffeemaker.php?name=" + escape(name) +
                                "&size=" + escape(size) +
                                "&beverage=" + escape(beverage) +
                                "&coffeemaker=1";
  sendRequest(request1, url);
} else {
  var coffeemakerStatusDiv2 =
    document.getElementById("coffeemaker2-status");
  status = getText(coffeemakerStatusDiv2);
  if (status == "Idle") {
    replaceText(coffeemakerStatusDiv2,
                "Brewing " + name + "'s " +
                size + " " + beverage);
    document.forms[0].reset();
    var url = "coffeemaker.php?name=" + escape(name) +
                                  "&size=" + escape(size) +
                                  "&beverage=" + escape(beverage) +
                                  "&coffeemaker=2";
    sendRequest(request2, url);
  } else {
    alert("Sorry! Both coffee makers are busy. " +
          "Try again later.");
  }
```

При отправке заказа первой кофеварке данные должны использоваться объектом request1

Запросы ко второй кофеварке должны передаваться при помощи объекта request2



coffee.js

Просто сделайте это

Мы почти завершили работу над нашим техническим чудом — приложением, управляющим несколькими кофеварками... осталось лишь изменить функцию обратного вызова `serveDrink()`, чтобы она работала с двумя объектами запросов вместо одного. В начале работы над функцией мы составили весь код JavaScript для работы с первым объектом запроса. Сейчас мы завершим функцию и включим в нее код для работы со вторым объектом запроса.

```
function serveDrink() {
  if ([redacted].readyState == 4) {
    if ([redacted].status == 200) {
      var response = [redacted].responseText;
      var whichCoffeemaker = response.substring(0, 1);
      var name = response.substring(1, response.length);
      if (whichCoffeemaker == "1") {
        var coffeemakerStatusDiv1 =
          document.getElementById("coffeemaker1-status");
        replaceText(coffeemakerStatusDiv1, "Idle");
      } else {
        var coffeemakerStatusDiv2 =
          document.getElementById("coffeemaker2-status");
        replaceText(coffeemakerStatusDiv2, "Idle");
      }
      alert(name + ", your coffee is ready!");
      [redacted] = createRequest();
    } else
      alert("Error! Request status is " + [redacted].status);
  } else if ([redacted].readyState == 4) {
    // All your code goes here
  }
}
```

Вся первая половина кода работает с объектом request1

После завершения работы с объектом запроса его необходимо заново инициализировать... для этого мы просто создаем объект заново

Ваша задача — написать весь код, который должен находиться здесь

Часто задаваемые вопросы

В: Мы используем два объекта запроса, чтобы иметь возможность отправлять запросы двум кофеваркам, верно?

О: Вообще говоря, для отправки запросов обеим кофеваркам достаточно одного объекта. Проблема в другом: с одним объектом запроса мы не сможем получать ответы от обеих кофеварок.

Когда вы отправляете запрос, а сервер работает над отправленным запросом, браузер через объект запроса связывает ответ сервера с вашим кодом JavaScript. Но если отправить другой запрос с использованием того же объекта до того момента, как будет завершен первый запрос, связь будет разорвана — первый запрос заменяется вторым.

В: Именно поэтому заказ Боба был выполнен, а заказ Джима — нет? Потому что его заказ был размещен вторым, и заменил предшествующий заказ Джима?

О: Точно. И хотя сценарий на сервере успешно выполнил заказ Джима и вернул ответ, браузер не сможет связать этот ответ с кодом JavaScript. Объект запроса уже был использован заново для обработки заказа Боба.

Использование двух объектов запроса гарантирует, что мы сможем использовать обе кофеварки, а браузер всегда передаст ответ сервера вашей функции обратного вызова JavaScript.

В: Выходит, в асинхронных приложениях всегда необходимо использовать два объекта запроса?

О: Вовсе нет. Более того, для большинства асинхронных приложений вполне достаточно одного объекта запроса. Два и более объекта необходимы только в одном случае — если ваше приложение должно выдавать два и более асинхронных запроса одновременно.

В: А почему мы снова преобразовали в функцию код, в котором создаются объекты запросов? Разве в главе этот код не был вынесен за пределы функции?

О: Действительно, был. Но как вы помните, в главе 2 этот код должен был выполняться только один раз для одного объекта запроса. Поскольку для кофеварок необходимы два объекта запроса, код создания объекта удобнее разместить в функции, а затем вызвать эту функцию дважды: по одному разу для каждого объекта запроса.

Впрочем, статическое оформление кода можно было бы сохранить (т. е. не включать его в функцию), но тогда пришлось бы сначала заменить все ссылки на request ссылками на request1, а затем продублировать код для объекта request2. В таком решении нет ничего плохого, но мы предпочли выделить код создания объекта в отдельную функцию, а затем просто вызвать эту функцию столько раз, сколько потребуется.

В: Зачем мы снова вызываем createRequest() в serveDrink()?

О: Когда ответ сервера будет готов, состояние готовности запроса задается равным 4. Однако использование объекта не приводит к сбросу его состояния готовности, а это значит, что условие в первой строке serveDrink() (request1.readyState == 4) всегда будет оставаться истинным после приготовления кофе на первой кофеварке.

Это создает проблемы при вызове serveDrink() после того, как кофе будет сварен на второй кофеварке. Хотя нас интересует вторая кофеварка (и второй объект запроса), первый объект запроса все еще будет оставаться в состоянии готовности 4. Чтобы избежать возни с такими проблемами, необходимо сбросить состояние объекта запроса.

В: А не проще будет просто обнулить свойство readyState объекта запроса в функции serveDrink()?

О: Нет, не проще... свойство readyState доступно только для чтения, и его значение не может задаваться напрямую из кода JavaScript. Только браузер может изменять свойство readyState объекта запроса.

Чтобы обойти это ограничение, мы просто вызываем функцию createRequest(), создающую новый объект. Трюк выглядит немного неестественно, но это самый простой способ заново инициализировать запрос и гарантировать, что состояние готовности будет равно 4 только у активного объекта запроса.

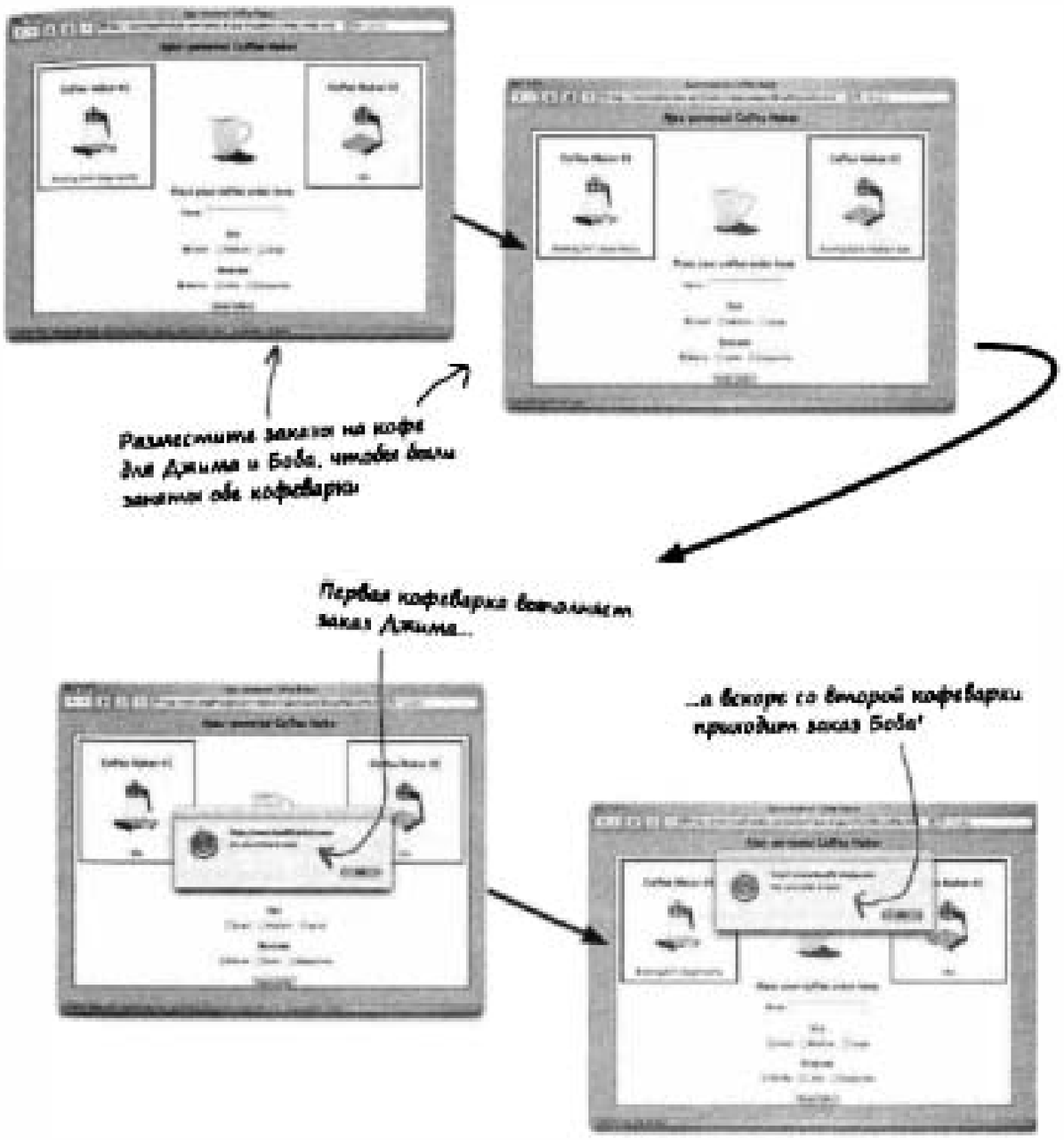
Просто сделайте это

Вот как должен выглядеть код части `serveDrink()`, работающий со второй кофеваркой. Убедитесь в том, что ваш код выглядит так же, и сохраните изменения в `coffee.js`.

```
    request = createRequest();
  } else
    alert("Error! Request status is " + request1.status);
} else if (request2.readyState == 4) {
  if (request2.status == 200) {
    var response = request2.responseText;
    var whichCoffeemaker = response.substring(0, 1);
    var name = response.substring(1, response.length);
    if (whichCoffeemaker == "1") {
      var coffeemakerStatusDiv1 =
        document.getElementById("coffeemaker1-status");
      replaceText(coffeemakerStatusDiv1, "Idle");
    } else {
      var coffeemakerStatusDiv2 =
        document.getElementById("coffeemaker2-status");
      replaceText(coffeemakerStatusDiv2, "Idle");
    }
    alert(name + ", your coffee is ready!");
    request2 = createRequest();
  } else
    alert("Error! Request status is " + request2.status);
}
```

Добро пожаловать в асинхронный мир!

Путешествие было долгим, но теперь у нас есть все необходимое для того, чтобы ваши коллеги могли получить свою порцию кофеина без малейшего ожидания. Убедитесь в том, что в файлы app.js и coffee.js были внесены все необходимые изменения, и в них используются два объекта запроса (по одному для каждой кофеварки).



Все еще сомневаетесь?

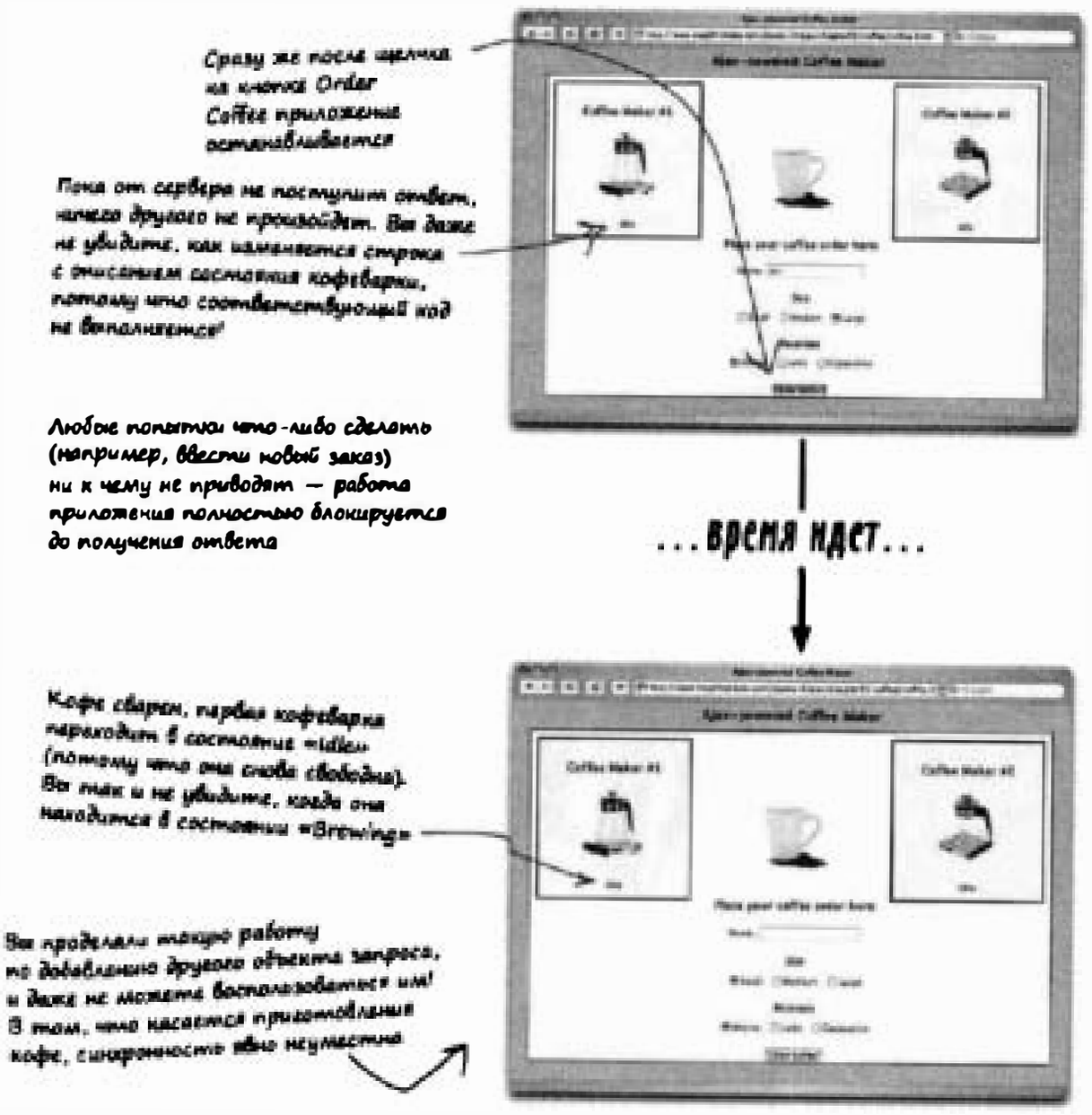
Хотите посмотреть своими глазами, как программа Coffee Maker работает в синхронном режиме? Это сделать несложно. Все, что для этого нужно, — заменить третий аргумент при вызове `request.open()` в функции `sendRequest()`: измените его с `true` на `false`

```
function sendRequest(request, url) {
  request.onreadystatechange = serveDrink;
  request.open("GET", url, false);
  request.send(null);
}
```

Замените этот аргумент на false, чтобы приложение отправляло запросы на сервер в синхронном режиме. Затем попробуйте запустить его. Что произойдет?

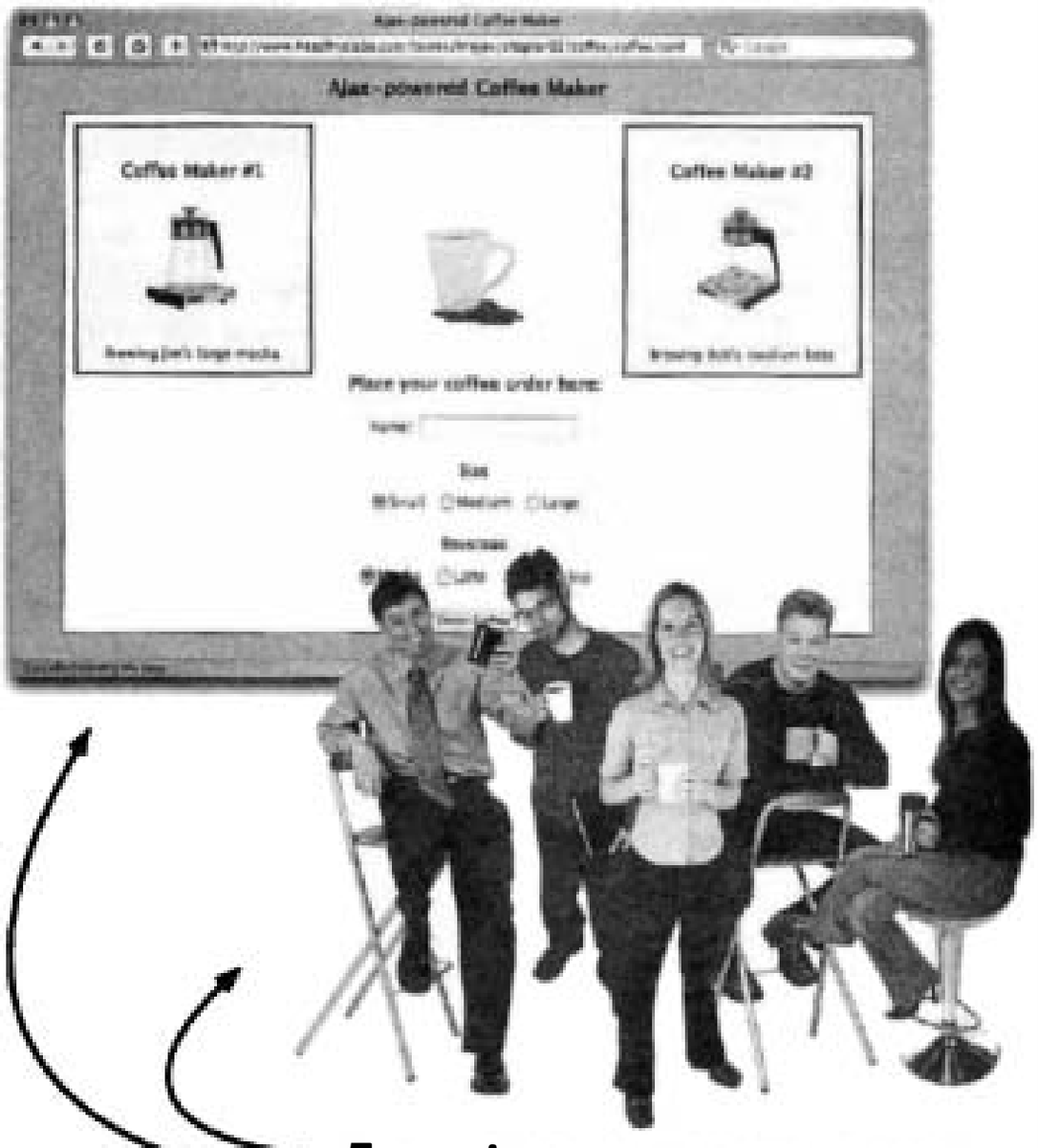
Проверка работы в синхронном режиме

При запуске синхронной версии приложения заметны существенные изменения. Как только вы щелкаете на кнопке Order Coffee, приложение «встает». Кнопка остается выделенной, а если и попытаться ввести другой заказ, вы увидите лишь вращающийся мячик (на Mac) или песочные часы (в Windows). Это означает: «Вам придется подождать — я занят!»



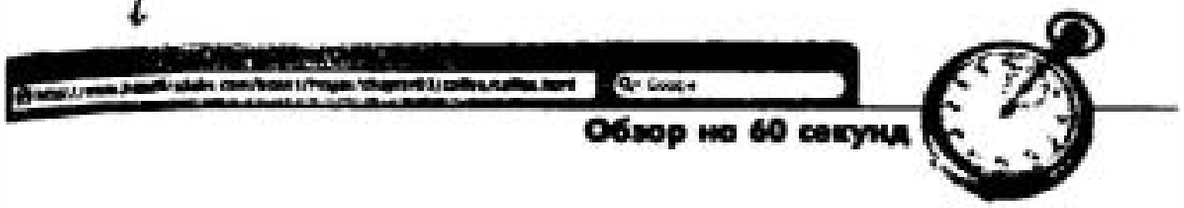
Верните нам асинхронность!

Не забудьте вернуть **sendRequest()** в файле **coffee.js** в асинхронный режим — никто не захочет ждать своей порции кофе



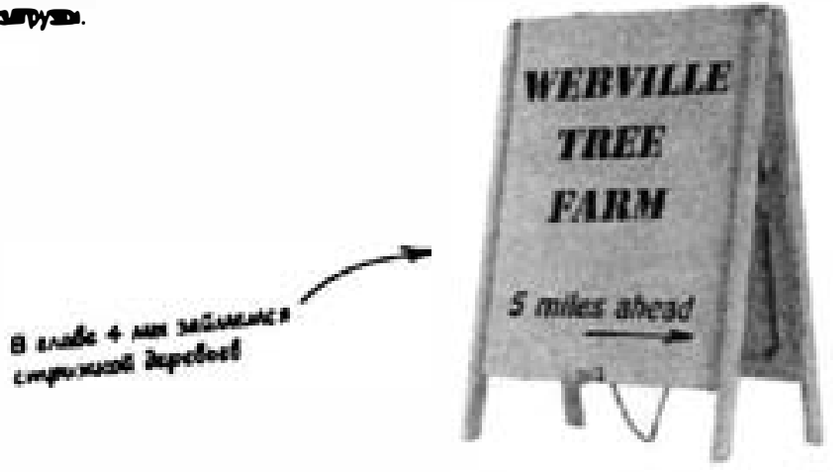
Две кофеварки, асинхронность — и коллектив доволен!

Проверьте сами как работает приложение!



Обзор на 60 секунд

- Синхронные запросы ожидают ответа от сервера; на это время работа приложения блокируется.
- Асинхронные запросы не ждут ответа, поэтому во время обработки запроса пользователь может продолжать работу с приложением.
- Многие приложения Ajax работают асинхронно, но если сервер возвращает ответ очень быстро — не так важно, является ли приложение синхронным или асинхронным.
- Асинхронные приложения лучше всего подходят для ситуаций, в которых получение ответа сервера требует определенного времени, или пользователи выполняют несколько операций одновременно на веб-странице или форме.
- При вызове метода open() для объекта запроса третий параметр задает тип запроса (асинхронный или нет). Для асинхронных запросов используется значение «true», а для синхронных — «false».
- Модель DOM позволяет изменять веб-страницу без перезагрузки.
- Использовать свойство innerHTML не рекомендуется. Такое решение не столь безопасно и надежно, как использование модели DOM для изменения веб-страницы из кода JavaScript.
- Асинхронная обработка запросов не требует никакого специального кода в приложении. Вы пишете код обычным образом, и он выполняется автоматически, без ожидания ответа сервера.
- Всегда полезно протестировать приложение в «синхронном режиме» и оценить, какую реальную пользу приносит асинхронность.



Зная имена функций JavaScript, мы можем написать оставшийся код HTML..

```
<h3>Beverage</h3>
<p>
 <input type="radio" name="beverage"
        value="mocha"
        checked="true">Mocha</input>
   
 <input type="radio" name="beverage"
        value="latte">Latte</input>
   
 <input type="radio" name="beverage"
        value="cappucino">Cappucino</input>
</p>
<p>
 <input type="button"
        onClick="orderCoffee();"
        value="Order Coffee" />
</p>
</form>
</div>
</div>

<div id="coffeemaker2">
 <h2>Coffee Maker #2</h2>
 <p><img src="images/CoffeeMaker2.gif"
         alt="Coffee Maker #2" /></p>
 <div id="coffeemaker2-status">Idle</div>
</div>
```

orderCoffee() — функция JavaScript для отправки нового заказа сценарию PHP

В коде первой кофеварки эти идентификаторы были равны «coffeemaker1» и «coffeemaker1-status»

...поэтому для второй кофеварки мы просто заменяем «1» на «2»

4 Модель DOM

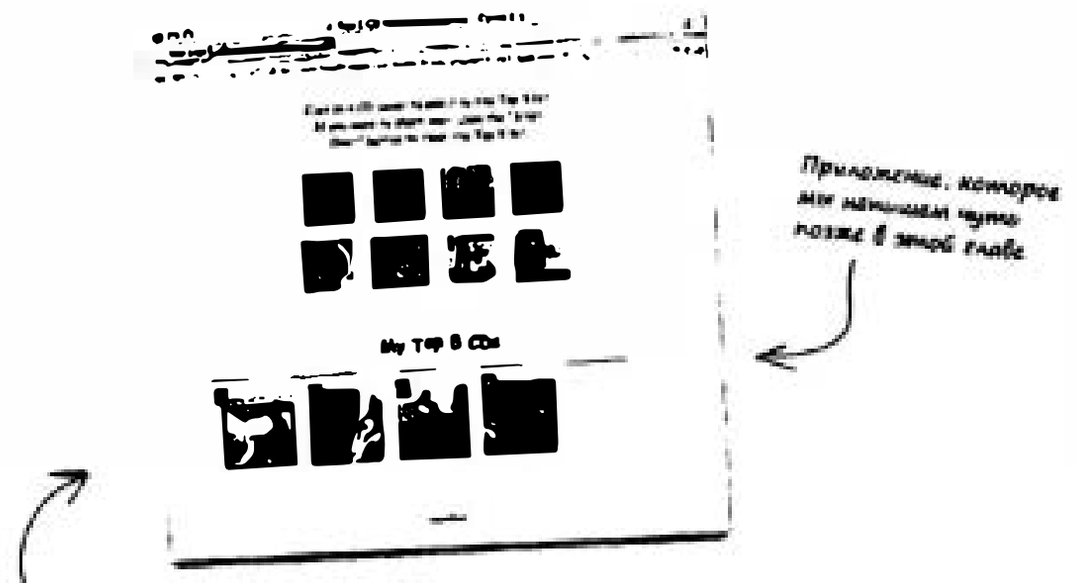
О стрижке деревьев

Наша цель: простое и удобное обновление веб-страниц.

Переходим к написанию кода, обновляющего веб-страницы «на ходу». С использованием модели DOM ваши страницы заживут новой жизнью, будут быстро реагировать на действия пользователя и навсегда избавятся от лишних перезагрузок. К концу главы вы научитесь добавлять, удалять и обновлять контент практически в любом месте веб-страницы.

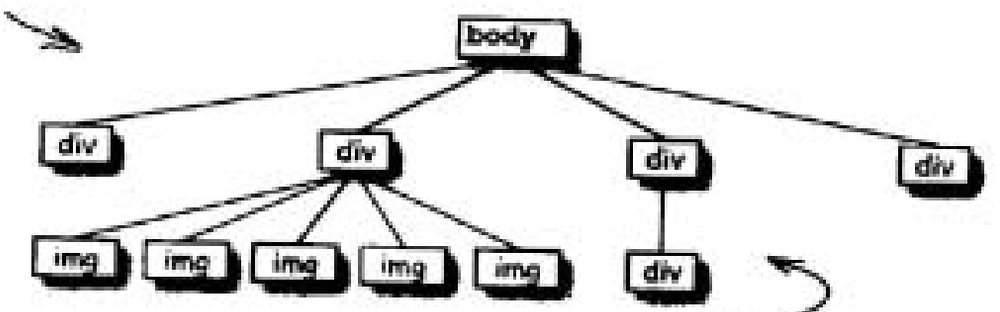

Вот какое дело: мне нравится разбираться в тонкостях асинхронного программирования... но все мои друзья помешаны на визуальных эффектах и интерактивности. Разве это не одно из составляющих популярности Ajax?
Netflix отображает удобное всплывающее окно, не требуя перезагрузки страницы
Джанна, начинающий Веб-программист
Flickr — замечательное Ajax-приложение... мгновенно реагирующее на действия пользователя, с превосходным интерфейсом
Google Maps — еще одно популярное Ajax-приложение.

отите написать динамическое риложение?



Воспользуйтесь моделью DOM

Браузер использует модель DOM (Document Object Model) для представления веб-страницы. Изменение модели в коде JavaScript приводит к автоматическому изменению веб-страницы



Знакомьтесь: модель DOM (снова)

Возможно, вы сами не сознавали этого, однако мы используем модель DOM, начиная с главы 1. Помните код из веб-отчета Кэти?

```
function updatePage() {
  var newTotal = request.responseText;
  var boardsSoldEl = document.getElementById("boards-sold");
  var cashEl = document.getElementById("cash");
  ...
}
```

Остальной код функции не приводится

Объект document предоставляет коду JavaScript доступ к дереву DOM, созданному браузером

А вот немного аналогичного кода из приложения Break Neck:

```
function getCustomerInfo() {
  var phone =
    document.getElementById("phone").value;
  var url = "lookupCustomer.php?phone=" +
    escape(phone);
  request.open("GET", url, true);
  request.onreadystatechange = updatePage;
  request.send(null);
}
```

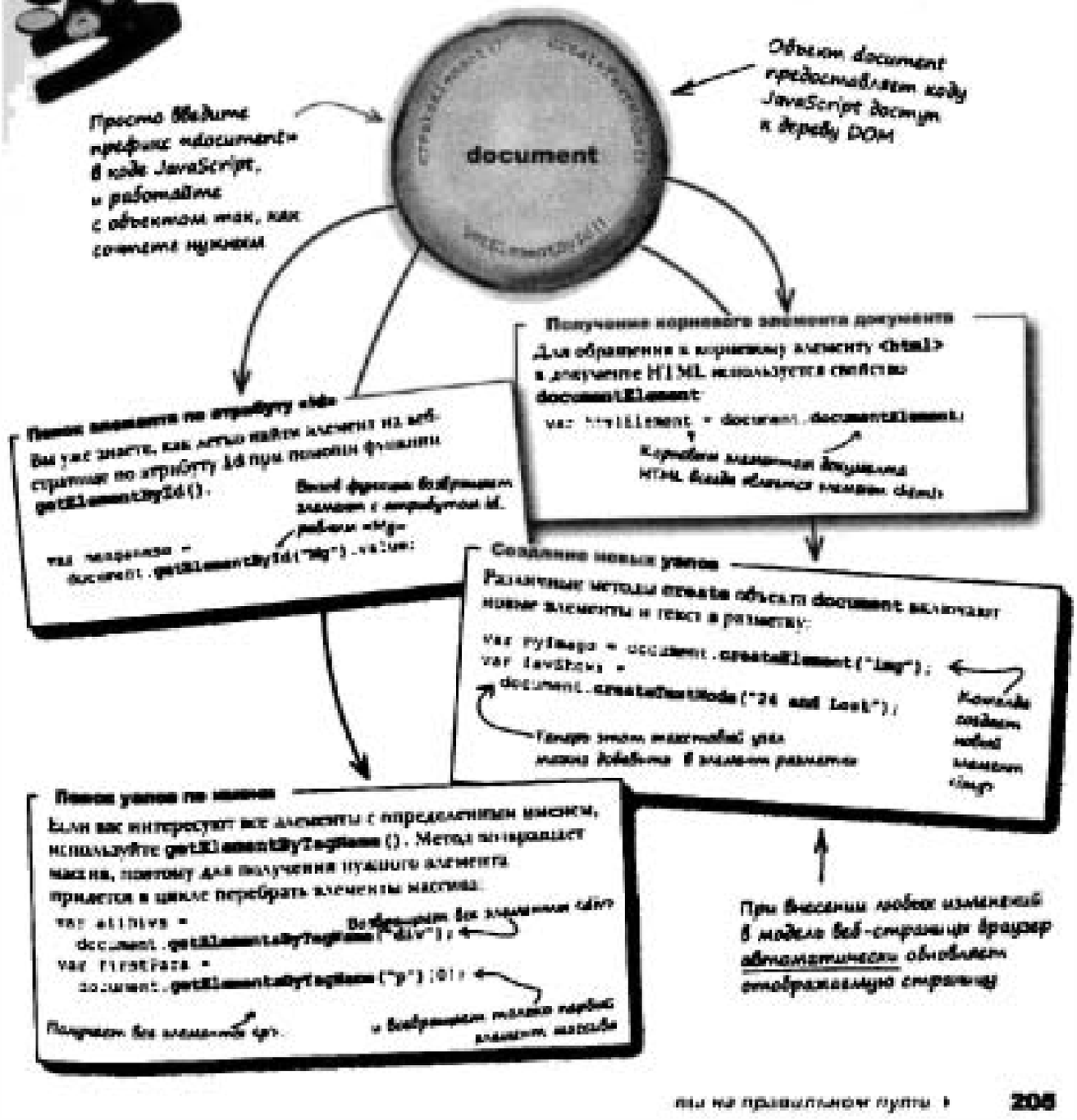
Это опять объект document

Объект «document» предоставляет коду JavaScript доступ к дереву DOM, используемому браузером.

Не упустите этот момент... Возможно, это самая важная информация во всей главе!

Под микроскопом: объект document

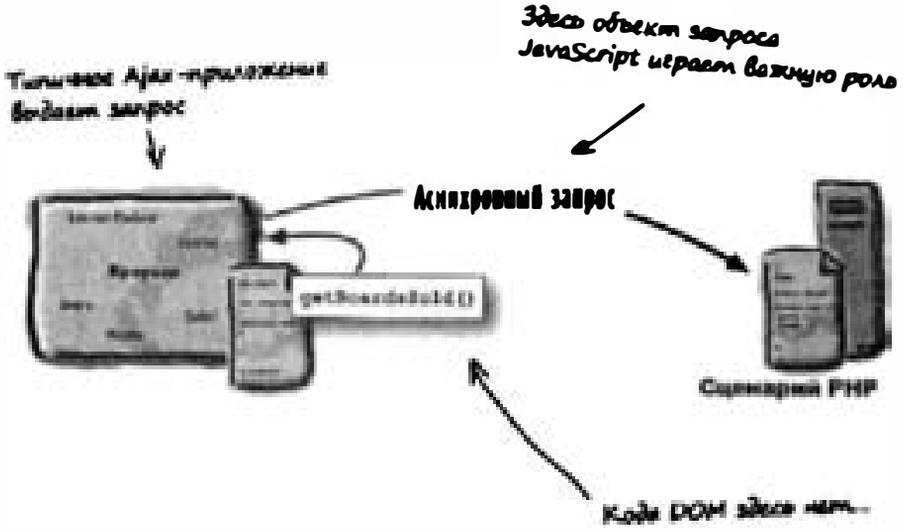
Все составляющие модели веб-страницы, используемой браузером, доступны через объект JavaScript с именем «document». Мы уже пользовались функцией getElementById(), но возможности объекта document этим отнюдь не исчерпаны.

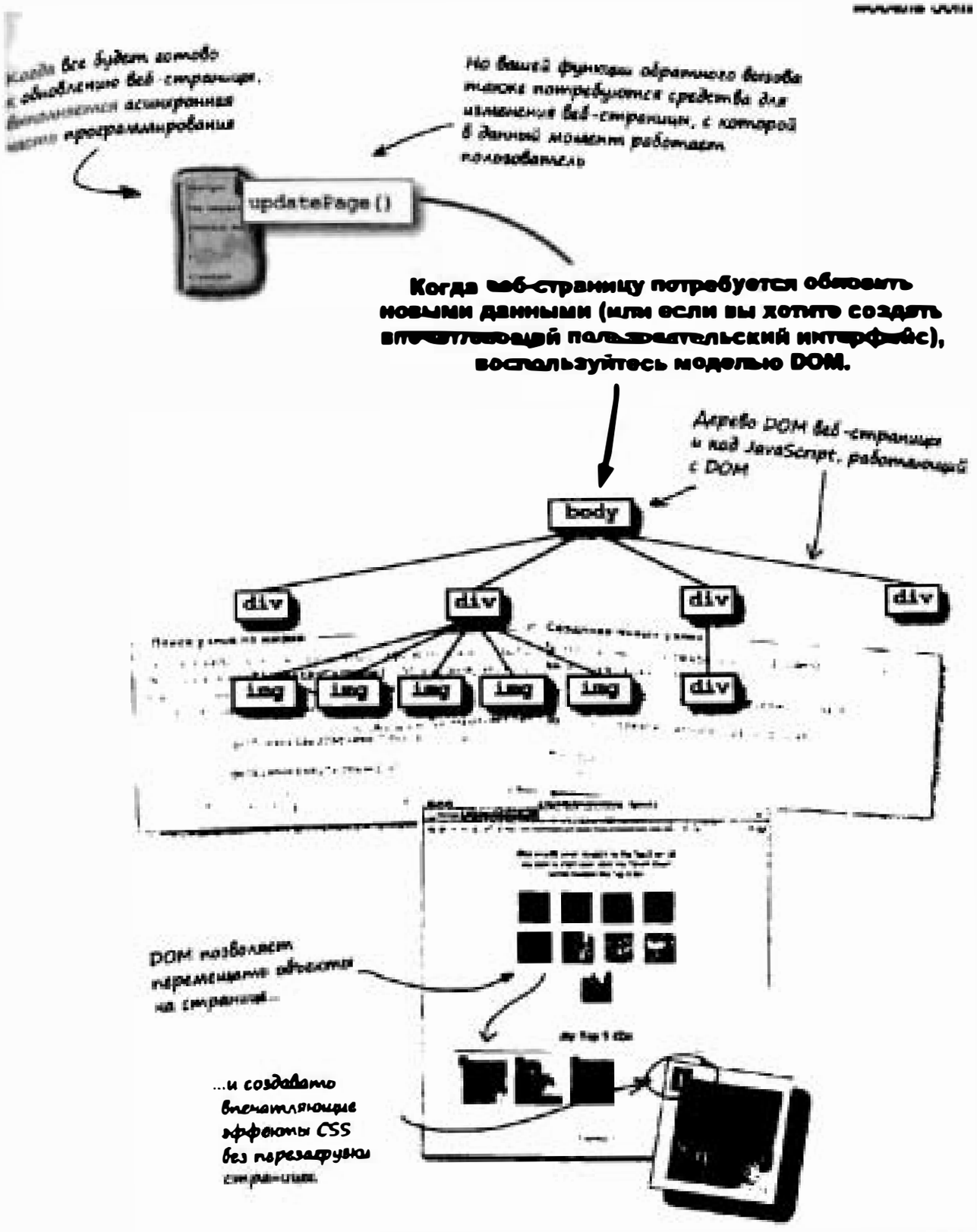


Похоже, это именно то, что мне нужно для оперативного обновления веб-страницы. Но причем здесь AJax? Я не вижу объекта запроса ни в одном из приведенных примеров.

DOM работает с Ajax...

...но DOM не является частью Ajax.




Когда все будет готово к обновлению веб-страницы, выполняется асинхронная часть программирования
updatePage()
На вашей функции обратного вызова также потребуются средства для изменения веб-страницы, с которой в данный момент работает пользователь
Когда веб-страницу потребуется обновить новыми данными (или если вы хотите создать впечатляющий пользовательский интерфейс), воспользуйтесь моделью DOM.
Дерево DOM веб-страницы и код JavaScript, работающий с DOM
body
div
div
div
div
img
img
img
img
img
div
DOM позволяет перемещать объекты на странице...
...и создавать впечатляющие эффекты CSS без перезагрузки страницы.

Использование DOM без Ajax

Так как модель DOM не связана с асинхронным программированием, ничто не мешает вам использовать ее *всех* веб-приложениях. Сейчас вы в этом убедитесь...

Великое испытание для главы 4

Написать веб-приложение, использующее модель DOM для создания динамического интерфейса пользователя, без применения кода Ajax.

Справившись с этой задачей, вы будете знать, что DOM поможет усовершенствовать работу любых приложений, в том числе и не использующих асинхронную загрузку.

Обратитесь в местный лесхоз...

Майк, владелец и основатель лесного хозяйства

Майк: Вы пришли как раз туда, куда нужно. Я могу помочь вам с моделью DOM. Сейчас у нас много заказов по этой части.

Дженни: Что? Наверное, вы не поняли. Я говорю о программировании веб-страниц, а у вас здесь просто какие-то деревья...

Майк: Вот именно. Деревья — это как раз то, что вам нужно.

Дженни: Это какое-то недоразумение. Я хочу программировать веб-страницы и изменять их внешний вид из кода JavaScript. При чем здесь деревья?

Майк: В сущности, каждая веб-страница представляет собой дерево.

Дженни: Какой-то каламбур? Чем веб-страницы похожи на деревья?

Майк: Вы меня плохо слушаете. Я не сказал, что веб-страницы похожи на деревья... Я сказал, что веб-страницы представляют собой деревья. Каждый раз, когда вы передаете браузеру страницу с HTML-кодом, браузер воспринимает этот код как дерево.

Дженни: Как такое возможно? Разумеется, код HTML имеет некоторую структуру, но прежде я еще никогда не слышала о браузерах и деревьях.

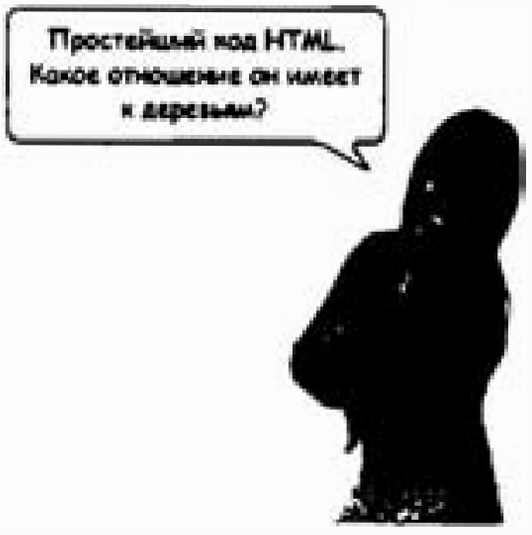
Майк: Да, большинство людей не в курсе. Но если вы собираетесь изменять и обновлять веб-страницы из кода JavaScript, придется познакомиться с концепцией дерева. Позвольте показать, что я имею в виду...

Код HTML, который вы передаете браузеру...

При создании веб-страницы вы пишете код HTML, представляющий различные части вашей страницы. Затем код HTML передается браузеру, а браузер определяет, как представить его на экране. Но если вы намереваетесь изменять свою веб-страницу из кода JavaScript, необходимо точно знать, как браузер «видит» ваш код HTML.

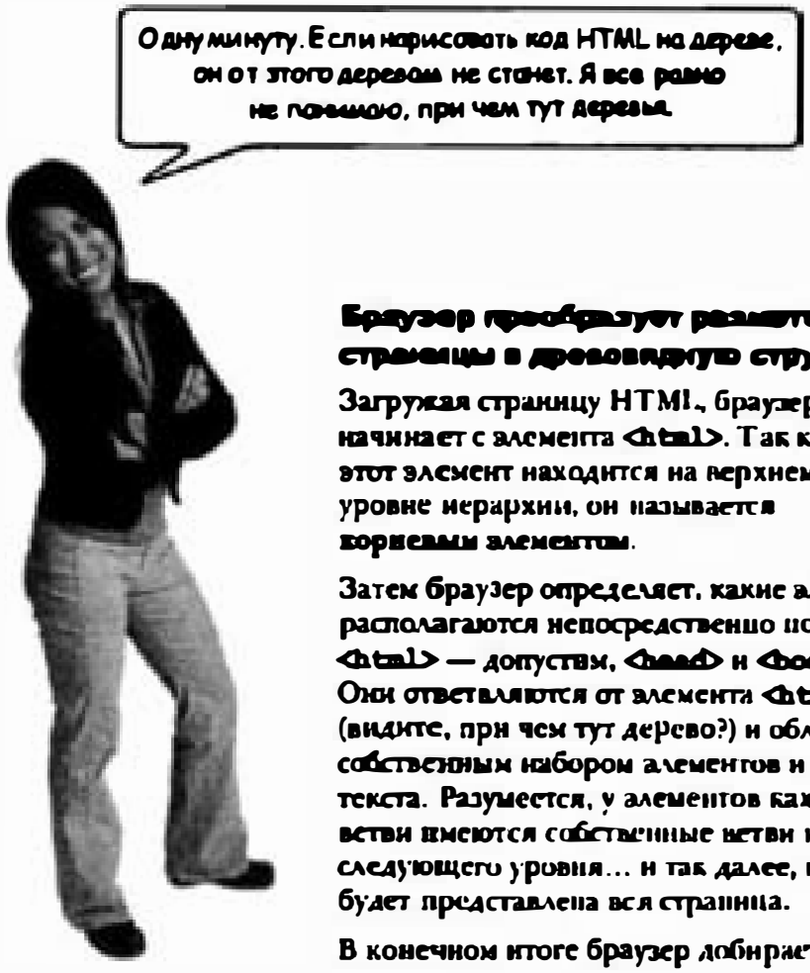
Для примера рассмотрим простой документ HTML:

```
<html>
 <head>
  <title>Webville Tree Farm</title>
 </head>
 <body>
  <h1>Webville Tree Farm</h1>
  <p>Welcome to the Webville Tree Farm. We're still learning
     about CSS, so pardon our plain site. We just bought
     <a href="http://www.headfirstlabs.com/books/hfhtml/">Head
     First HTML with CSS &amp;amp; XHTML</a>, though, so expect
     great things soon.</p>
  <p>You can visit us at the corner of Binary Blvd. and
     DOM Drive. Come check us out today!</p>
 </body>
```



...и как код HTML воспринимается браузером

Браузер должен как-то разобраться в этой разметке и организовать ее так, чтобы браузер (и наш код JavaScript) могли работать со страницей. Браузер преобразует разметку в следующий формат:



Браузер преобразует разметку страницы в древовидную структуру

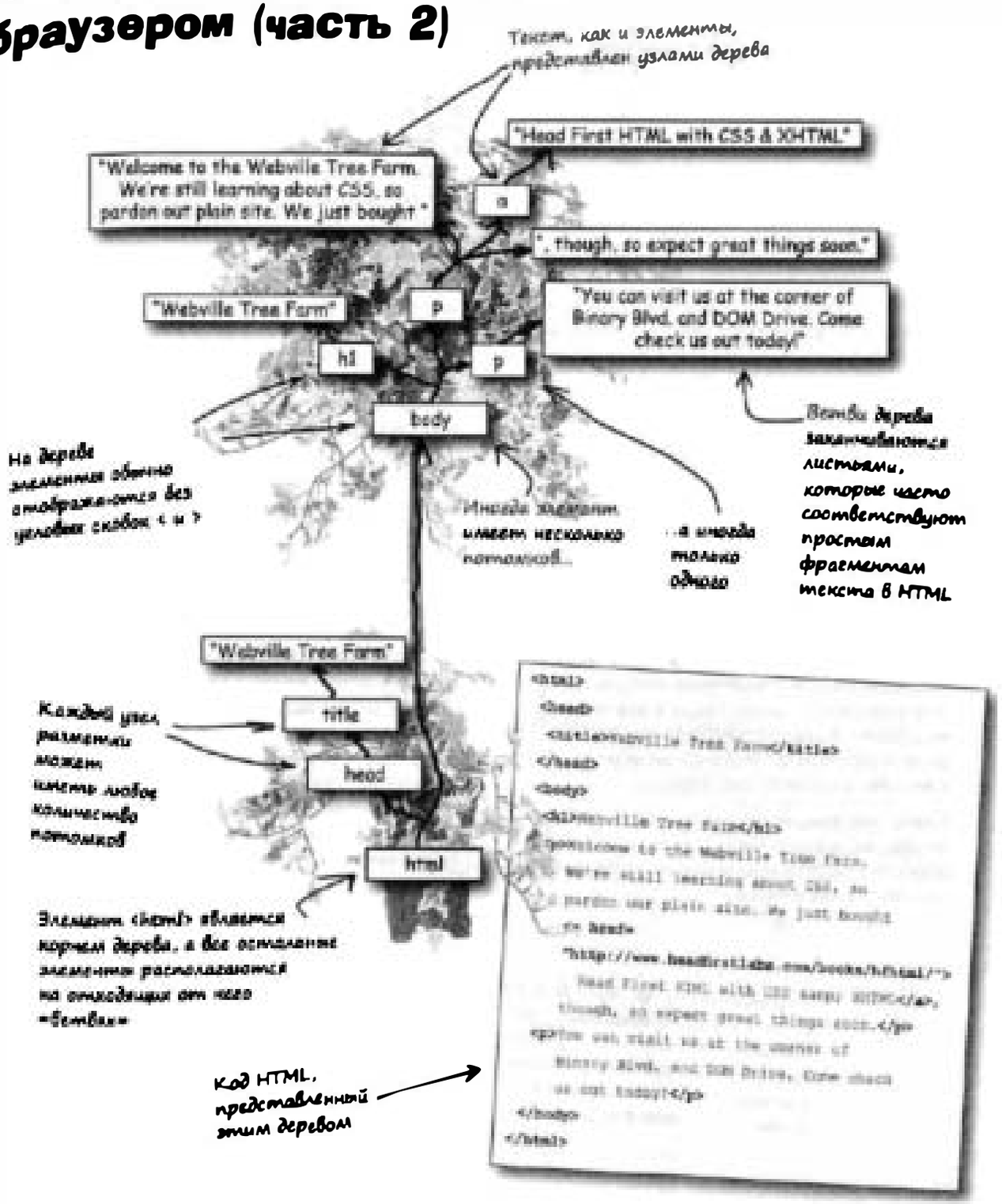
Загружая страницу HTML, браузер начинает с элемента <html>. Так как этот элемент находится на верхнем уровне иерархии, он называется **корневым элементом**.

Затем браузер определяет, какие элементы располагаются непосредственно под <html> — допустим, <head> и <body>. Они ответвляются от элемента <html> (видите, при чем тут дерево?) и обладают собственным набором элементов и текста. Разумеется, у элементов каждой ветви имеются собственные ветви и узлы следующего уровня... и так далее, пока не будет представлена вся страница.

В конечном итоге браузер добирается до составляющих разметки, которые не имеют узлов следующего уровня — скажем, текста в элементе <p> или элементов <img>. Такие составляющие разметки называются *листовыми узлами*, или просто *листьями*. Таким образом, вся страница с точки зрения браузера представляет собой одно большое дерево.

Взгляните еще раз на изображение дерева; возможно, с узлами, обозначающими связи между узлами разметки, картина станет более понятной.

Как код HTML воспринимается браузером (часть 2)



Часто задаваемые вопросы

В: Если в плану код JavaScript, использующий модель DOM, обязательно ли включать ссылку на файл ajax.js?

О: Нет. В сущности, Ajax и модель DOM никак не связаны, если не считать того факта, что большинство Ajax-приложений использует DOM. Впрочем, существует немало других приложений, к Ajax не относящихся, но использующих DOM в течение многих лет. Браузер с включенной поддержкой JavaScript — вот и все, что необходимо для написания веб-страниц средствами DOM.

В: А как насчет файла text-utils.js? Ведь он содержит несколько функций для работы DOM, верно? Он нам понадобится?

О: Мы использовали файл text-utils.js в главах 1 и 3. Файл содержит несколько вспомогательных функций, использующих DOM и значительно упрощающих модификацию веб-страниц. Однако в этой главе мы займемся более глубоким изучением DOM, и вы узнаете, как работать с этой моделью напрямую, без использования готового кода JavaScript.

К концу этой главы вы будете достаточно понимать, как работает код из файла text-utils.js, сможете разработать существующие и написать собственные вспомогательные функции.

В: Вы назвали некоторые узлы деревьев «потомками». Получается, у элемента могут быть элементы-потомки?

О: Да. Когда браузер структурирует код HTML в виде дерева, он начинает с корневого элемента. Каждый фрагмент содержимого под корневым узлом является ветвью, но он также может рассматриваться как «элемент-потомок». Аналогия их семейных отношений можно использовать и при перемещении от листьев к корню дерева; элемент <head> может рассматриваться как «родитель» элемента <title>. Запомните эту информацию, мы неоднократно вернемся к теме родителей и потомков в оставшейся части этой главы.

В: Вы используете много новых терминов — корни, ветви, потомки... Как их все запомнить?

О: Это не так сложно, как может показаться. Представьте себе дерево, и у вас не будет особых проблем. Вы уже много лет используете слова «корень», «ветвь» и «лист». Что касается родителей и потомков, то перемещаясь по направлению к корню дерева, вы переходите от потомков к родителям; перемещение в обратном направлении означает переход от родителей к потомкам. Совершенно новым может показаться только термин «узел», но и с ним мы сейчас разберемся.

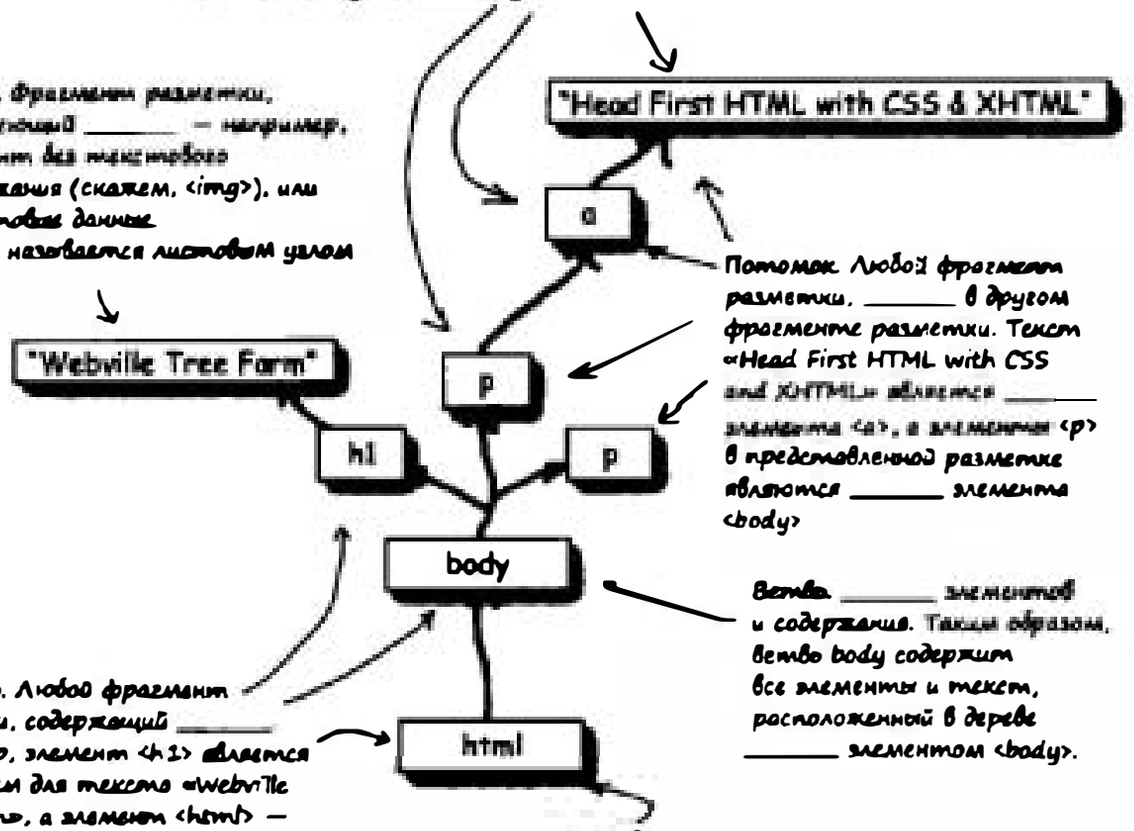
В приложении 2 приведен весь код из файла text-utils.js вместе с комментариями относительно того, как работает та или иная функция этого файла. После чтения этой главы мы рекомендуем обратиться к приложению 2 — ваши познания в области DOM позволят понять все содержимое файла text-utils.js.

СОЗДАЕМ ...

Словарь веб-терминов

Что за интерес рассматривать набор определений? Мы хотим, чтобы у читателя работал мозг, а не только глаза. Ниже приведено несколько определений из словаря веб-терминов, причем в каждом определении некоторые слова отсутствуют. Попробуйте закончить определения, вписав недостающие слова.

Узел. Любой _____ фрагмент разметки (например, элемент или текст). Элементу <a> соответствует узел _____, тогда как тексту «Head First HTML with CSS and XHTML» соответствует _____ узел.



Лист. Фрагмент разметки, не имеющий _____ — например, элемент без текстового содержания (скажем, <img>), или текстовые данные. Также называется листовым узлом.

Потомок. Любой фрагмент разметки, _____ в другом фрагменте разметки. Текст «Head First HTML with CSS and XHTML» является _____ элемента <a>, а элементы <p> в представленной разметке являются _____ элемента <body>.

Ветвь _____ элементов и содержания. Таким образом, ветвь body содержит все элементы и текст, расположенный в дереве _____ элементом <body>.

Родитель. Любой фрагмент разметки, содержащий _____. Например, элемент <h1> является родителем для текста «Webville Tree Farm», а элемент <html> — родителем для элемента <body>. Также встречаются термины «родительский элемент» и «родительский узел».

Корневой элемент. Элемент _____, содержащий все остальные элементы. В дереве HTML корневым элементом всегда является _____.

отдельный
содержащийся
потомков
текстовый

другую разметку
собеседничества
<html>
потомков

дерева
ней
элемента
потомками

Это слова, которыми вы должны заполнить пустые места в тексте

СОЗДАЕМ СОБСТВЕННЫЙ

Словарь веб-терминов

РЕШЕНИЯ

Далее приводится словарь веб-терминов с заполненными пропусками. Если вам кажется, что другие слова будут более уместными — ничего страшного... Главное — убедитесь в том, что вы понимаете смысл каждого определения и общий принцип объединения фрагментов для формирования дерева разметки веб-страницы.

Узел. Любой отдельный фрагмент разметки (например, элемент или текст). Элементу <a> соответствует узел элемента, тогда как тексту «Head First HTML with CSS and XHTML» соответствует текстовый узел.

Лист. Фрагмент разметки, не имеющий потомков, — например, элемент без текстового содержания (скажем, <img>), или текстовое данное. Также называются листовыми узлами.

Потомок. Любой фрагмент разметки, содержащийся в другом фрагменте разметки. Текст «Head First HTML with CSS and XHTML» является потомком элемента <a>, а элементы <p> в представленной разметке являются потомками элемента <body>.

Ветвь. Совокупность элементов и содержания. Таким образом, ветвь body содержит все элементы и текст, расположенный в дереве под элементом <body>.

Родитель. Любой фрагмент разметки, содержащий другую разметку. Например, элемент <h1> является родителем для текста «Webville Tree Farm», а элемент <html> — родителем для элемента <body>. Также встречаются термины «родительский элемент» и «родительский узел».

Корневой элемент. Элемент дерева, содержащий все остальные элементы. В дереве HTML корневым элементом всегда является <html>.

Разложите по полкам

Раз у нас появился удобный словарь веб-терминов, настало время воспользоваться им. Ниже изображено дерево HTML, которое мы уже рассматривали на одной из предыдущих страниц. Ваша задача — взять метки, сваленные внизу, и расставить их в нужных частях дерева. Будьте внимательны: для некоторых частей дерева подойдут разные метки, а использовать все метки не обязательно. Удачи!

Насчет представления HTML в виде дерева... Почему текст в элементе <p> был разбит на несколько узлов? Разве не проще работать с одним узлом, чем с двумя или тремя?

Чтобы понять, о чем говорит Джанет, вернитесь к с. 215

Для браузера важен порядок

Когда браузер получает код HTML и представляет его в виде дерева, он должен придерживаться заданного порядка следования текста и элементов. В противном случае абзацы могут следовать в неверном порядке, лишние слова окажутся выделенными жирным шрифтом или подчеркнуты, и т. д.

Давайте еще раз взглянем на разметку в тексте приветствия:

Узел содержит немалый объем текста.

Элемент <p>, который является родителем для текстового содержимого

```
<p>
  Welcome to the Webville Tree Farm. We're still
  learning about CSS, so pardon our plain site. We
  just bought <a href=
    "http://www.headfirstlabs.com/books/hfhtml/">
    Head First HTML with CSS &amp;amp; XHTML</a>,
  though, so expect great things soon.
</p>
```

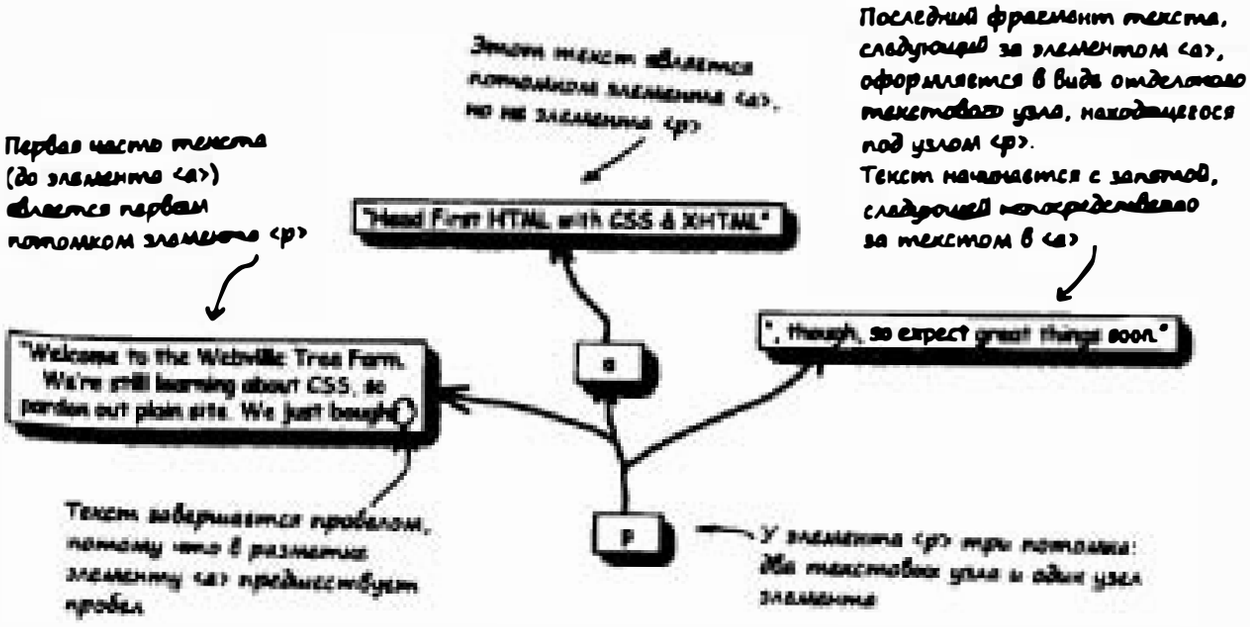
...так же как элемент <a>, создающий ссылку

Элемент <a> тоже содержит собственный текст

Дерево должно точно соответствовать коду HTML: в противном случае пользователи, работающие с веб-страницей, могут запутаться. Для примера возьмем это дерево... Элемент <p> должен точно знать, где располагается его элемент-потомок <a> по отношению к окружающему тексту.

Как браузеру проще всего сохранить правильный порядок следования текста и ссылки в коде HTML? Сначала первая часть текста помещается в один узел под элементом <P>, затем создаётся узел элемента <Р>, после чего добавляется еще один текстовый узел для всего текста, следующего *после* <Р>. Дерево страницы HTML обычно читается слева направо, вот так:

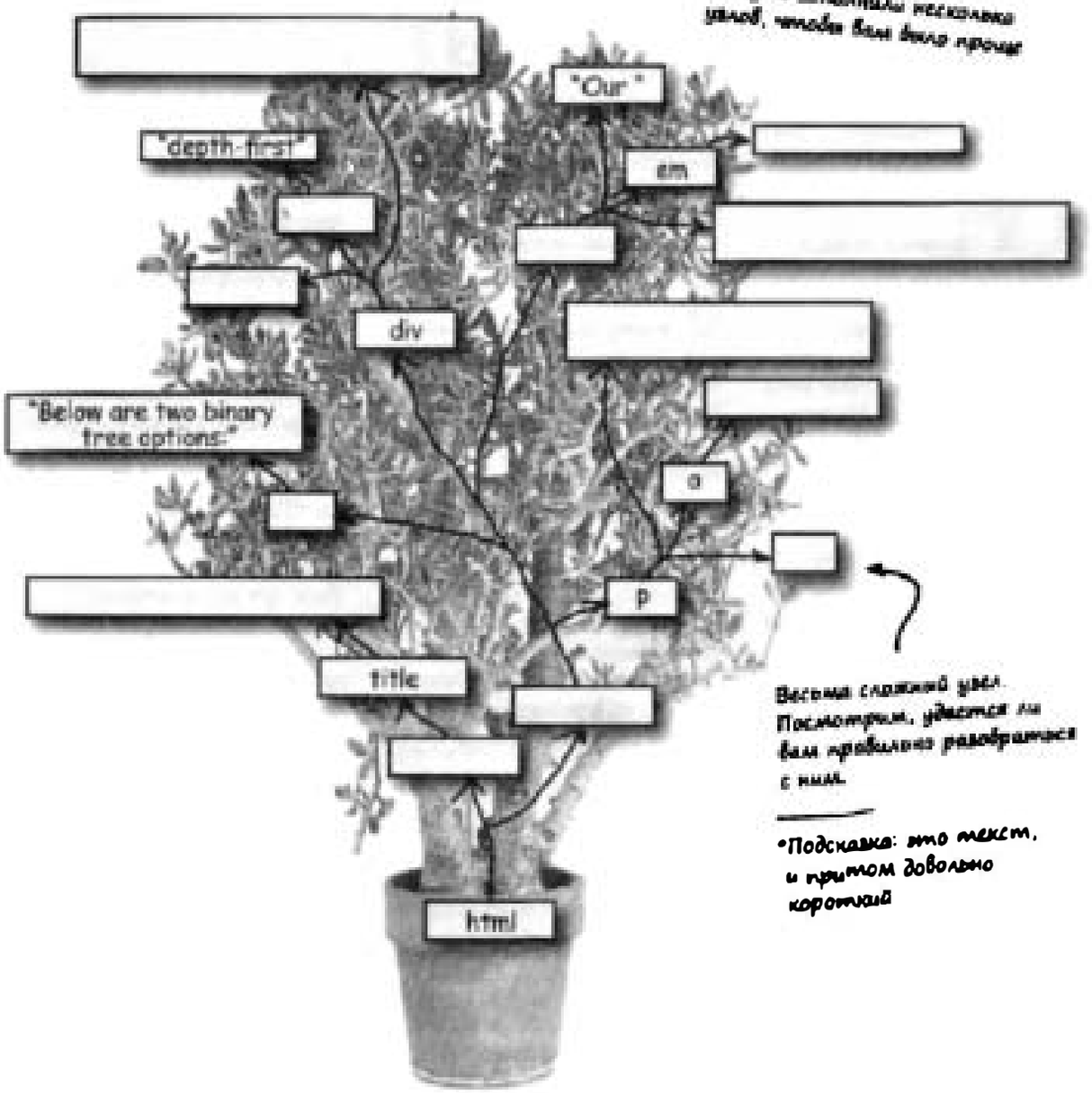
Да, с первого взгляда это понять нелегко. Прошлайте этот абзац, обратитесь к диаграмме и перечитайте еще раз. Не беспокойтесь вы все поймете!



Просто сделайте это

Ниже приведен простой документ HTML. Ваша задача — представить, как этот код HTML будет преобразован браузером в древовидный формат. Справа изображено **дерево с пустыми** вставями и листьями. Чтобы вам было удобнее, мы разместили **на нем пустые поля для всех фрагментов разметки;** заполните их элементами или текстом из разметки HTML

```
<html>
 <head>
  <title>Binary Tree Selection</title>
 </head>
 <body>
  <p>Below are two binary tree options:</p>
  <div>
   Our <em>depth-first</em> trees are great for folks that
   are far away.
  </div>
  <div>
   Our <em>breadth-first</em> trees are a favorite for
   nearby neighbors.
  </div>
  <p>You can view other products in the
     <a href="menu.html">Main Menu</a>.</p>
 </body>
</html>
```



Мы уже выполнили несколько узлов, чтобы было видно прочее

Весьма сложный узел. Посмотрим, удастся ли вам правильно разобраться с ним.

•Подсказка: это текст, и притом довольно короткий

Ответ будет через несколько страниц..

Часто задаваемые вопросы

В: Порядок следования элементов и текста в HTML важен, не так ли?

О: Да. Для элемента <table> первым потомком является элемент <head>, а вторым — элемент <body>. Если поменять их местами, дерево получится другим, и в результате страница будет иначе отображаться в браузере. Работая с древовидным представлением кода HTML, всегда следите за сохранением порядка элементов.

В: Вы постоянно говорите об элементах и тексте. Это разные вещи, или текст также является разновидностью элемента?

О: Элемент представляет собой имя, заключенное в угловые скобки (например, <title> или). Текст представляет собой символы, находящиеся в элементах ("Web'ville Tree Farm" или "directions"). Такие элементы обладают атрибутами — такими, как id="binary" или class="greenbar".

Все эти сущности считаются узлами, но элементы, атрибуты и текст относятся к разным категориям узлов. Следовательно, дерево может содержать текстовый узел, но не может содержать текстовых элементов.

В: А как же атрибуты? Вы говорите, что дерево содержит узлы атрибутов, но такие узлы не являются «потомками» тех элементов, которым они принадлежат?

О: Атрибуты — особый случай. Если разметка содержит конструкцию вида , было бы неправильно говорить, что атрибут id является потомком элемента .

Браузер сохраняет атрибуты каждого элемента в специальном списке. Таким образом, узлы атрибутов существуют, но не имеют простого и удобного представления на дереве. Через несколько страниц мы займемся моделью DOM, и вы увидите, что для каждого элемента ведется специальный список атрибутов.

В: Во многих старых страницах HTML не используются закрывающие теги ( ). Как браузер поступает со страницами, содержащими подобную разметку?

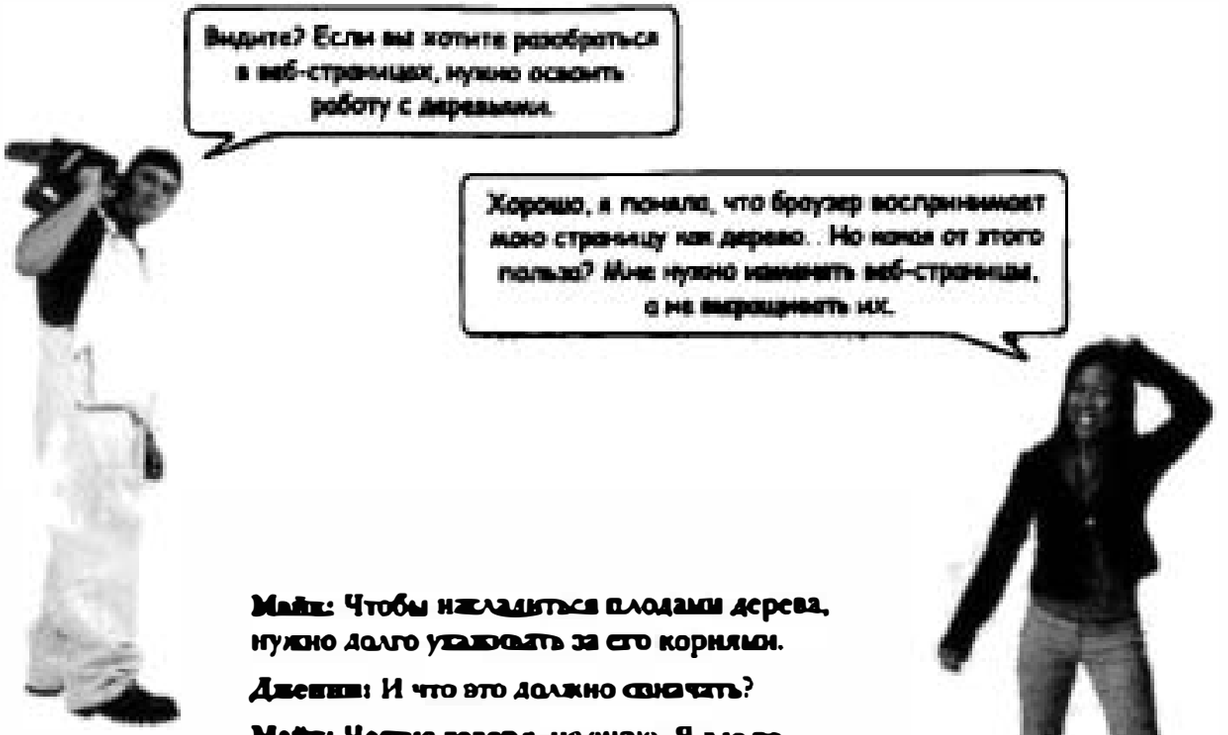
О: Да вы настоящий знаток HTML! Все верно, многие старые страницы HTML написаны небрежно и содержат элементы с некорректным закрытием или вложением. В таких случаях браузер делает все, что может, для правильного создания структуры дерева.

Обычно браузер строит дерево правильно, но если вам когда-либо попадется страница, которая выглядит не так, как предполагалось — возможно, браузер неверно построил дерево для кода HTML этой страницы. Это главный повод в пользу аккуратности при написании кода HTML.

В: Будет ли страница одинаково отображаться в разных браузерах?

О: Если вы пишете стандартный код HTML (а еще лучше, проверяете написанный код на корректность), разные браузеры почти всегда строят для него одинаковые деревья. Спецификации HTML 4.01 и XHTML 1.0 и 1.1, среди прочего, обеспечивают стандартное представление HTML, на которое могут рассчитывать браузеры.

Впрочем, если вы по небрежности забудете закрыть элемент или используете устаревший код HTML, браузер постарается сделать все возможное для правильного представления вашей разметки. В этом случае деревья, построенные разными браузерами, могут не совпасть. Но пока вы пишете стандартные, синтаксически корректные страницы HTML, проблем быть не должно.



Майк: Чтобы наслаждаться плодами дерева, нужно долго ухаживать за его корнями.

Джейни: И что это должно означать?

Майк: Честно говоря, не знаю. Я где-то вычитал эту фразу и подумал, что она красиво звучит. Как бы то ни было ... поскольку браузер рассматривает страницу как дерево, вы должны написать код для работы с деревом. И в этом вам поможет модель DOM.

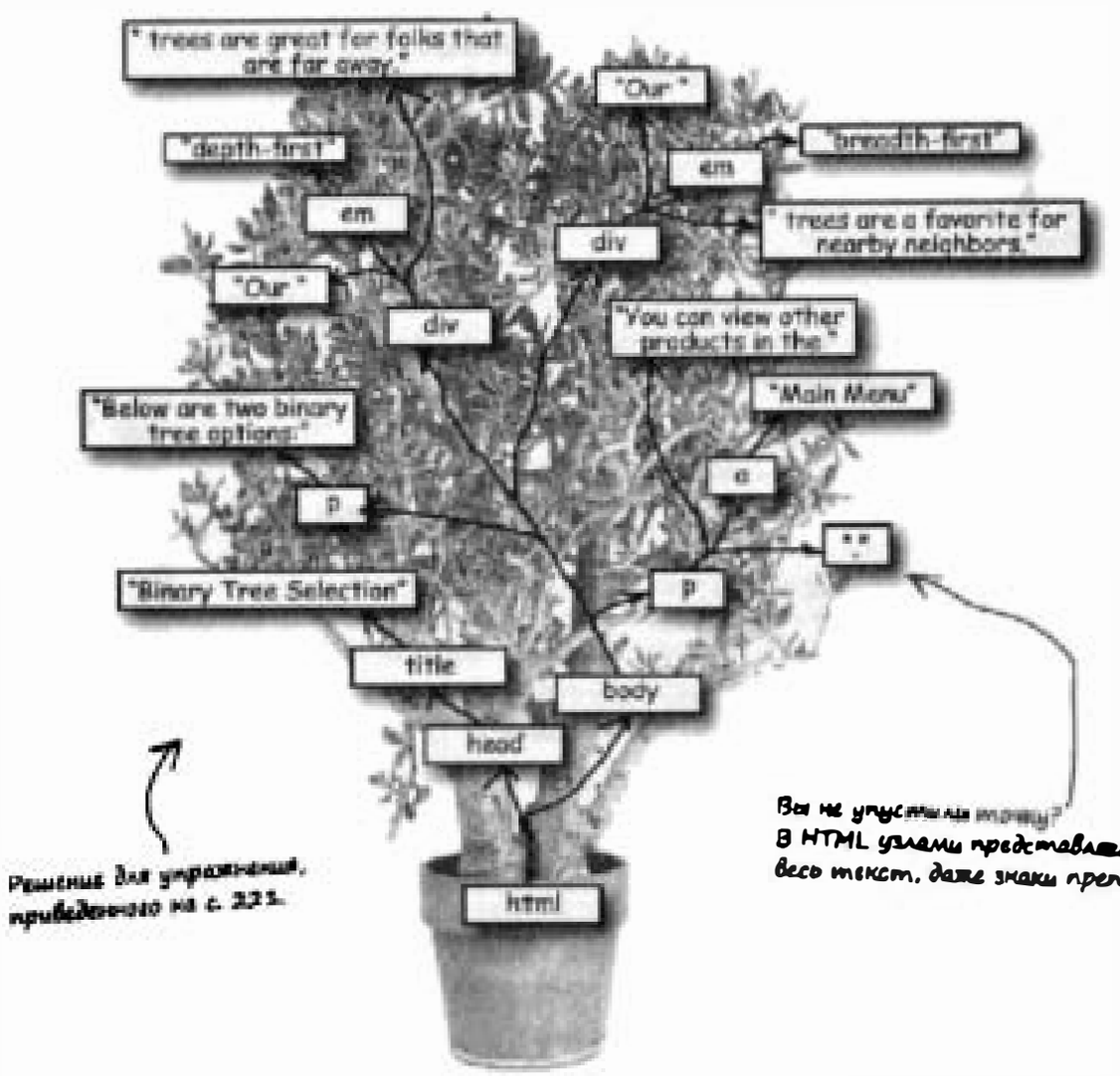
Джейни: Другой разговор. За этим я сюда и пришла... Чтобы узнать о модели DOM

Майк: Модель Document Object Model (сокращенно DOM) предназначена для работы с деревом, построенным браузером на основе веб-страницы. Вы работаете с DOM из кода JavaScript и обновляете дерево браузера. После обновления дерева страница автоматически изменяется браузером без перезагрузки или обновления страниц.

Спецификации и стандарты DOM публикуются группой, которая называется World Wide Web Consortium (часто сокращенно называемым W3C). Сообщения специально для любителей стандартов.

Обратно в лес

Запомните: браузер воспринимает веб-страницу как дерево из элементов, текста, атрибутов и других составляющих разметки:



Решения для упражнений, приведенного на с. 223.

Вы не упустили мысль? В HTML узлами представляется весь текст, даже знаки препинания

Браузер видит деревья вверх ногами

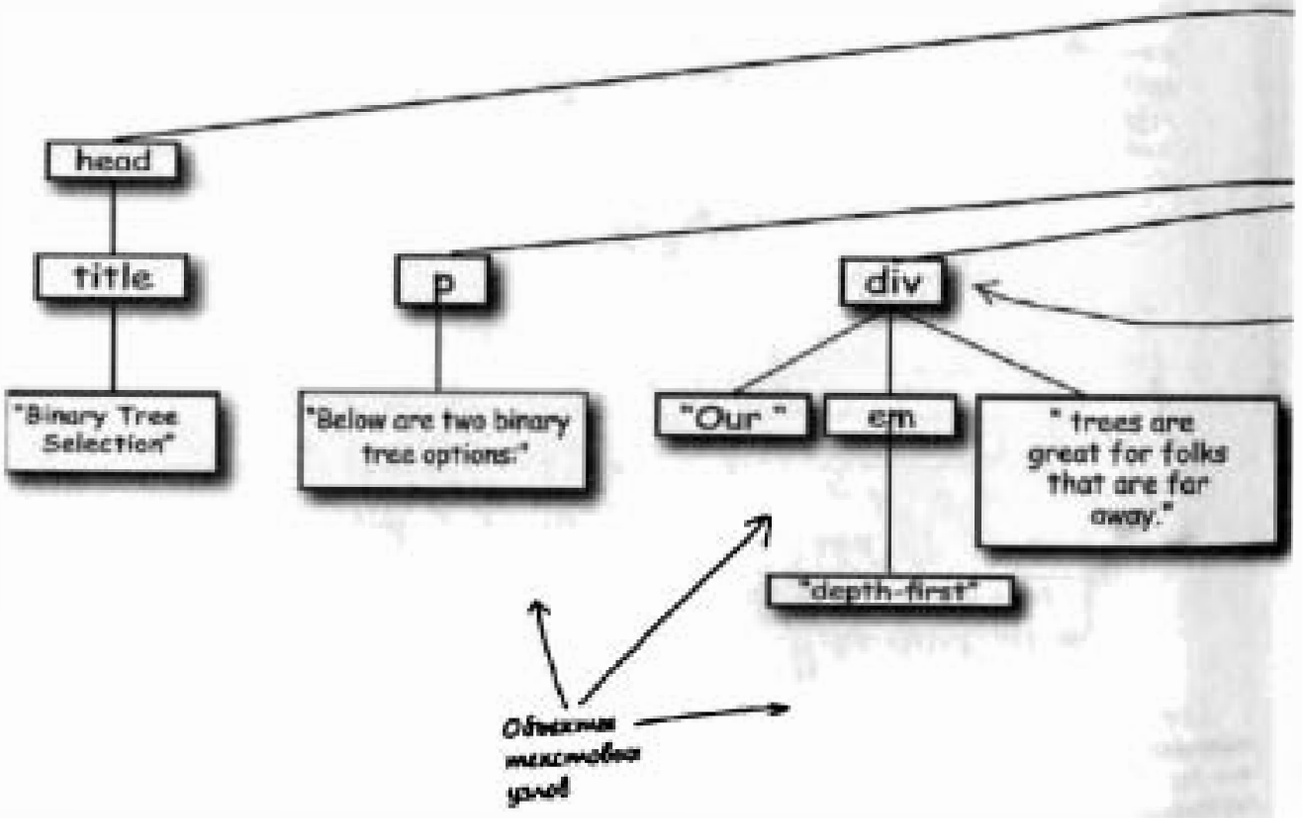
Конечно, браузер не работает с изображением дерева, на которое наносится ваша разметка. Чтобы действительно хорошо понять, что браузер хранит в памяти, необходимо познакомиться с моделью DOM. Давайте посмотрим, как браузер берет только что описанное дерево и представляет его в виде набора объектов.

Сначала браузер переворачивает все дерево корнем вверх, чтобы элемент <html> находился не внизу, а наверху:

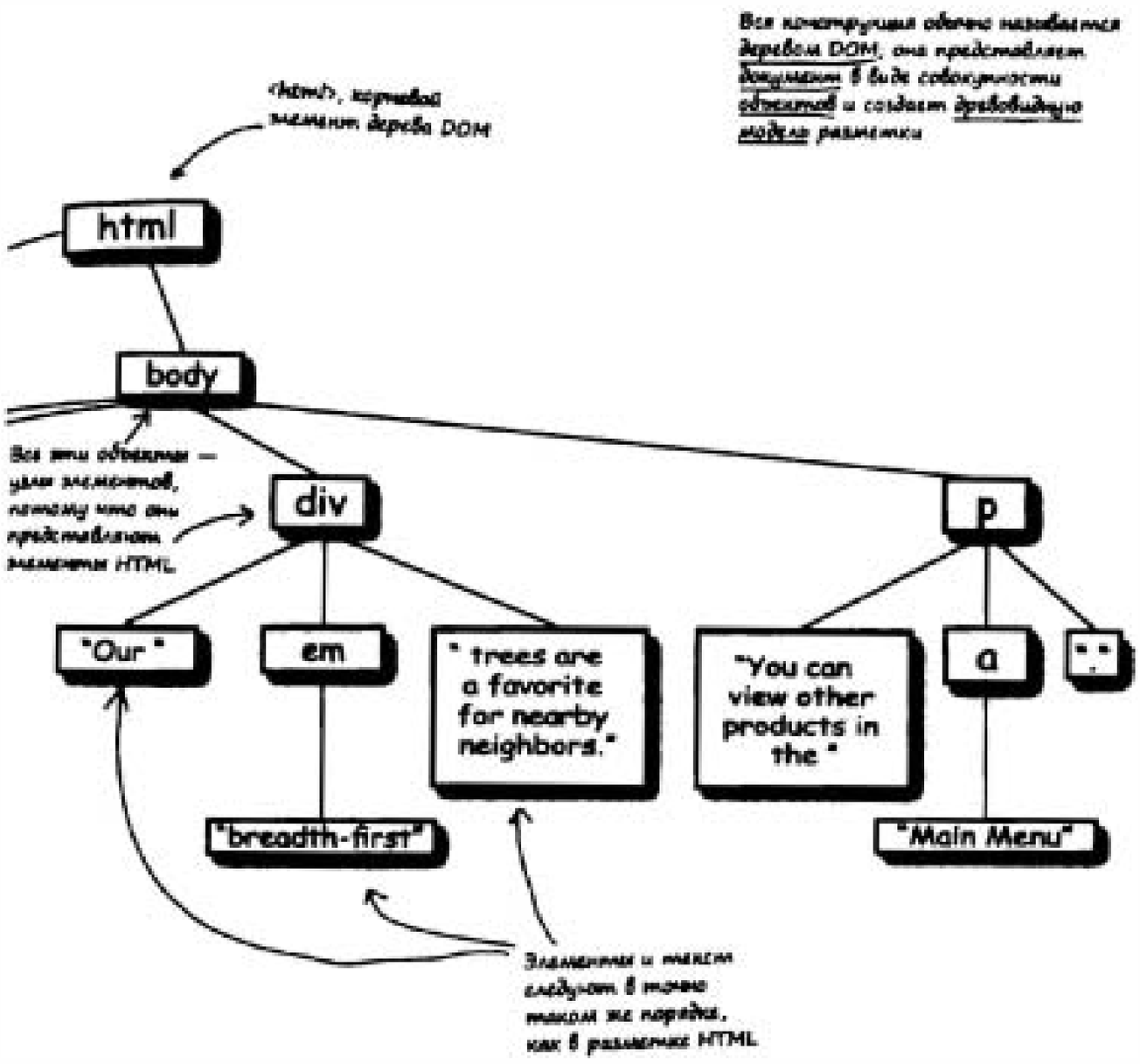
Новая разновидность: деревья DOM

Получив разметку с корневым элементом `<html>`, браузер создаст новый объект для каждого узла дерева. Результат представляет собой совокупность объектов, «связанных» в единое целое, как показано на следующем рисунке.

Дерево не рисуем... Вероятно, вы уже и так поняли



Картинка немного похожа на дерево и все же на ней хорошо различаются корень, ветви и листья, как на с. 225


Все конструкции обычно называются деревом DOM, они представляют документ в виде совокупности объектов и создают древовидную модель разметки
<html>, корневой элемент дерева DOM
html
body
Все эти объекты — узлы элементов, потому что они представляют элементы HTML
div
p
"Our "
em
" trees are a favorite for nearby neighbors."
"You can view other products in the "
a
"."
"breadth-first"
"Main Menu"
Элементы и текст следуют в точно таком же порядке, как в разметке HTML

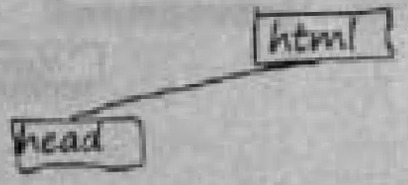
Сделайте это...

Возьмите фломастер и представьте себя в роли браузера. Ваша задача — нарисовать дерево DOM для приведенного кода HTML.

```
<html>
 <head><title>Gonna Get Me Some Blues</title></head>
 <body>
  <p>Do you have the blues? If not, check out Stefan
     Grossman's <a href="http://www.guitarvideos.com">
     Guitar Workshop</a> for some great DVDs and
     instructional videos.</p>
 </body>
</html>
```

Нарисуйте дерево DOM таким, каким оно должно быть по вашему представлению:

Начните с блоков с именами элементов и текстом. Затем соедините блоки линиями, как здесь*



* Один из рецензентов указал, что рисование блоков перед вписыванием имен элементов необязательно. После многочасовых раздумий мы согласились.

Разговор в студии

Редакция: У нас в гостях модель Document Object Model. Мы побеседуем о том, как браузер в реальности воспринимает HTML, и что должен знать программист JavaScript для оперативного обновления страниц HTML. Приятно познакомиться, мисс Document... эээ, мисс Object... Как вас лучше называть?

DOM: Большинство знакомых называет меня просто DOM. «Document Object Model» слишком длинно и неудобно.

Редакция: О, так гораздо лучше. Итак, я задам вопрос напрямую: когда браузер смотрит на страницу HTML, он видит вас?

DOM: Вообще-то браузер начинает с файлов HTML, CSS и JavaScript. Но в действительности браузеры не любят работать с текстом, а в файлах хранится именно текст. Стили CSS и обработчики событий JavaScript трудно применять к набору строк текста.

Редакция: Что же, это логично. В наши дни никто не хранит CSS в файлах HTML... CSS обычно хранится в отдельном файле.

DOM: Верно, и в большинстве случаев код JavaScript тоже не хранится в одном файле с HTML. Поэтому браузер с моей помощью объединяет HTML, CSS и JavaScript в одну структуру данных. Для каждого фрагмента HTML браузер создает объект. Я слежу за тем, чтобы эти объекты были правильно структурированы и связаны друг с другом.

Редакция: Действительно, браузеру гораздо удобнее работать с элементами HTML в таком виде. Но пока неясно, какое место в этой картине занимают CSS и JavaScript.

DOM: Мои объекты, представляющие HTML, обладают полезными методами, которые можно вызывать, и свойствами, которые вы можете задавать. Скажем, для кнопки можно вызвать метод addEventListener(); после этого при каждом ее нажатии будет вызываться заданная вами функция JavaScript. Подобные задачи решаются очень легко, если браузеры используют меня — и мои объекты — для моделирования веб-страниц.

Редакция: Понятно. Значит, вы также упрощаете изменение текущего содержания веб-страниц?

DOM: Да, это так. Например, в элемент можно включить новый текстовый узел, чтобы на странице появился новый текст, или удалить элемент из родительского узла, чтобы исчезла целая секция страницы.

Редакция: Замечательно. И все это не требует перезагрузки страницы?

DOM: Вот именно! Мои структуры данных постоянно находятся в памяти браузера, поэтому ему не придется обращаться к серверу, и даже быть подключенным к сети.

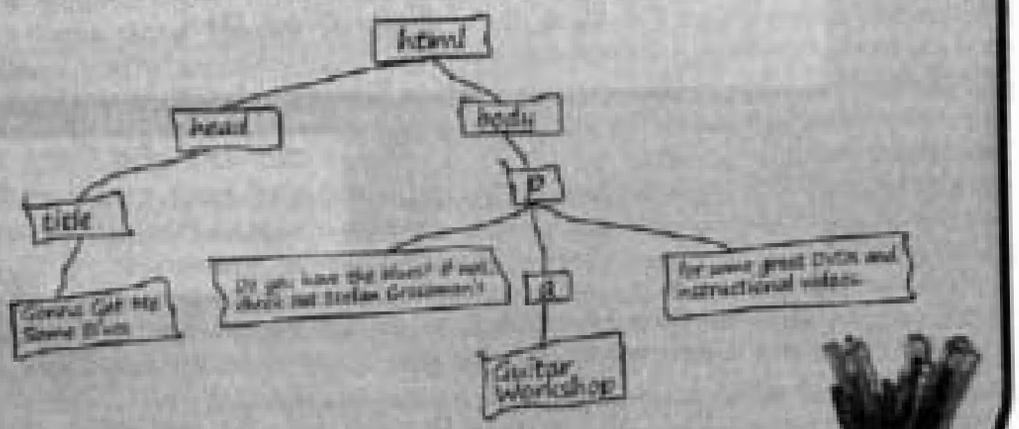
Редакция: Надеемся, в будущем нам удастся познакомиться поближе...

Сделайте это... — Ответ

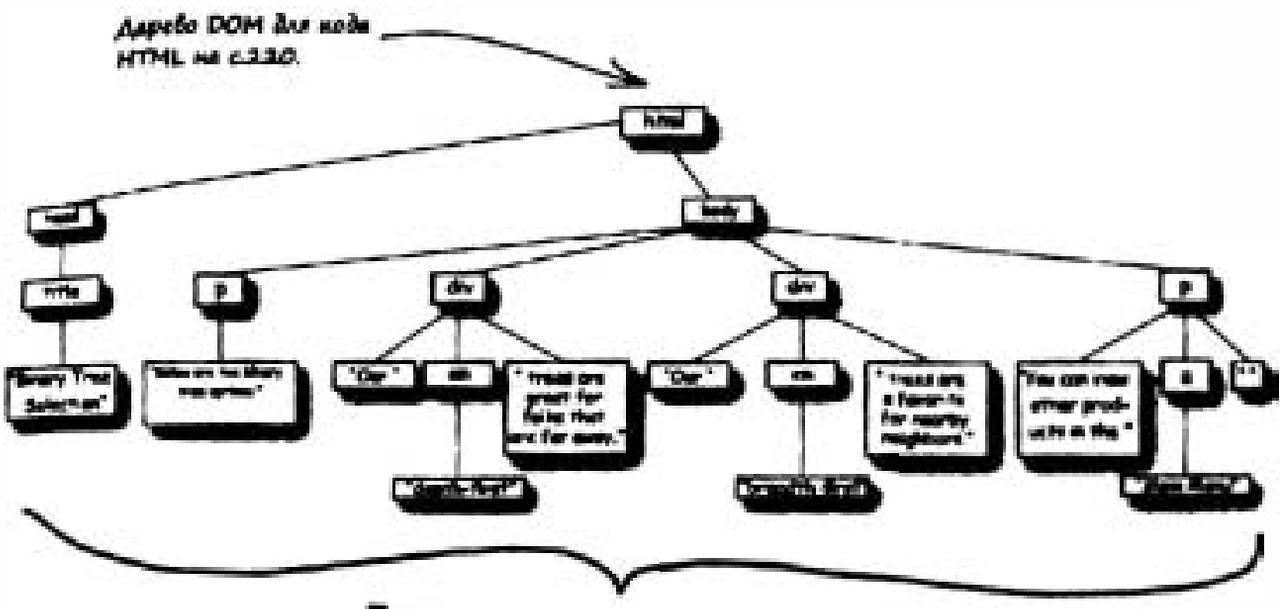
Возьмите флипчарт и представьте себя в роли браузера. Ваша задача — нарисовать дерево DOM для приведенного кода HTML.

```
<html>
 <head> <title>Gonna Get Me Some Blues</title> </head>
 <body>
  <p>Do you have the blues? If not, check out Stefan
     Grossman's <a href = "http://www.guitarvideos.com">
     Guitar Workshop</a> for some great DVDs and
     instructional videos. </p>
 </body>
</html>
```

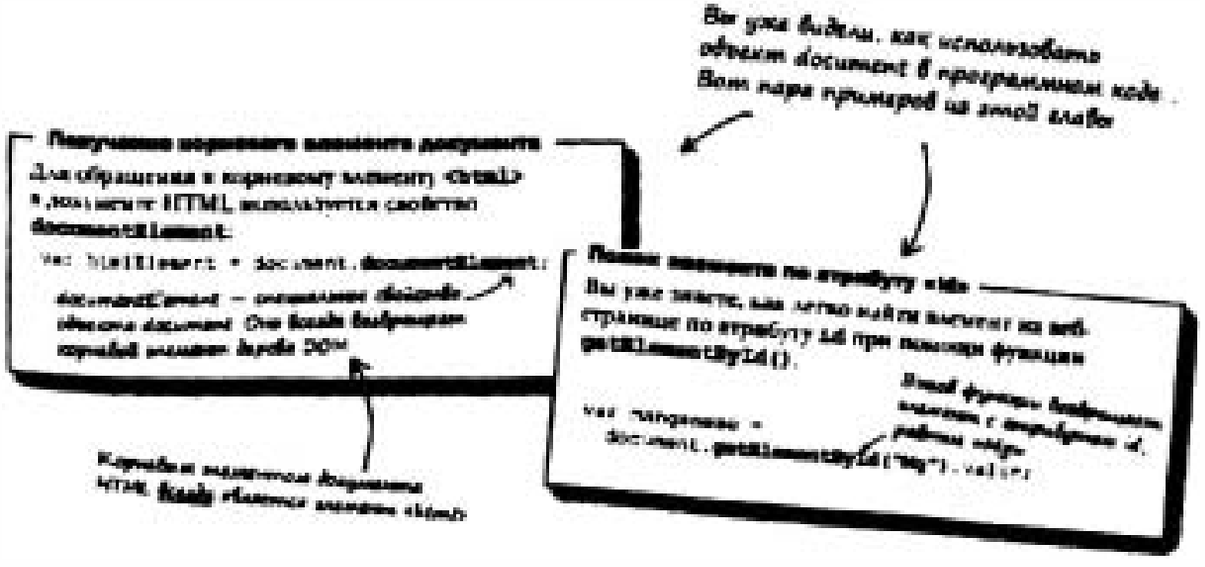
Вот дерево получилось таким:



Помните то громадное дерево DOM, которое мы рассматривали?



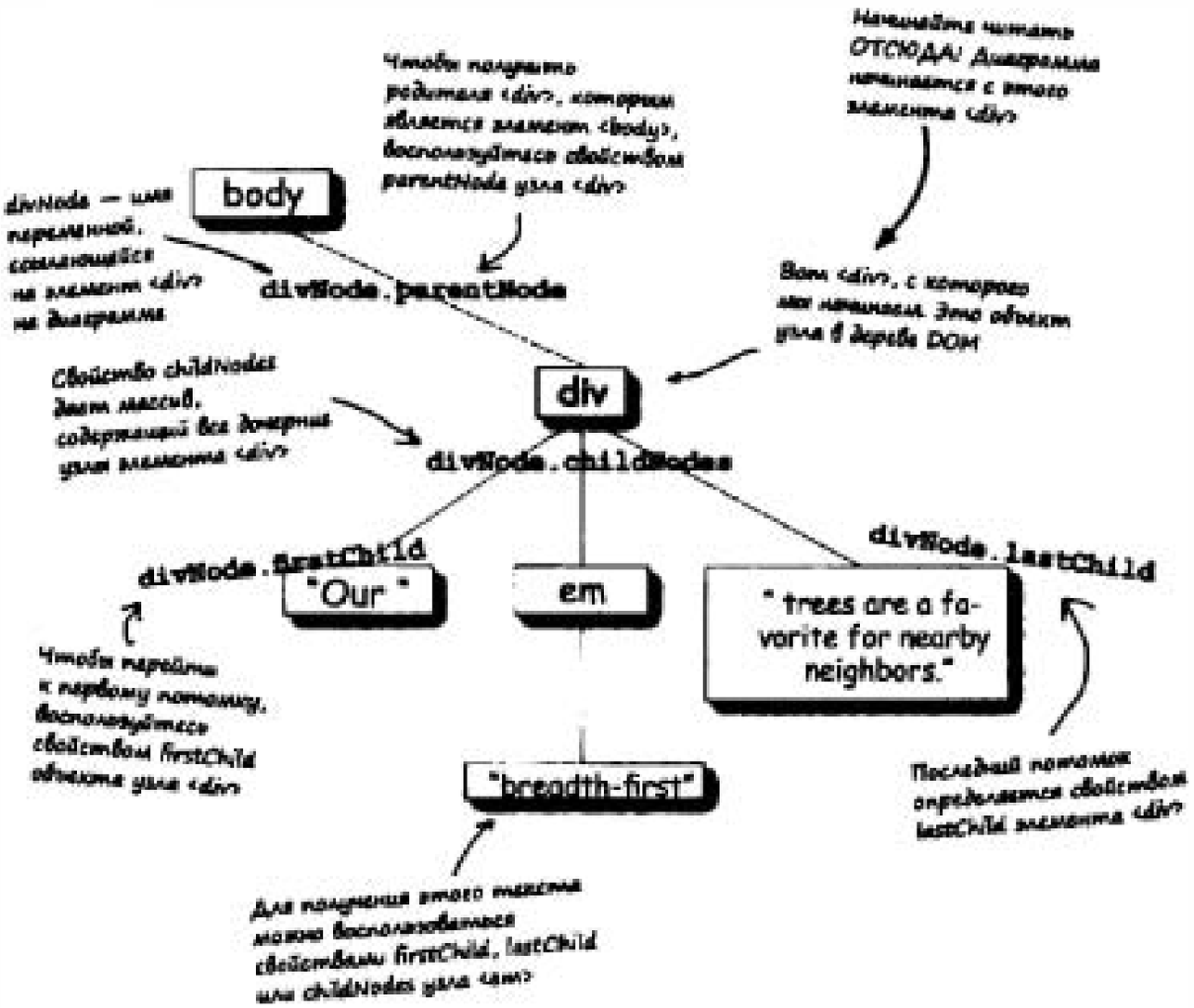
Переменная с именем "document" позволяет работать со всем деревом в JavaScript

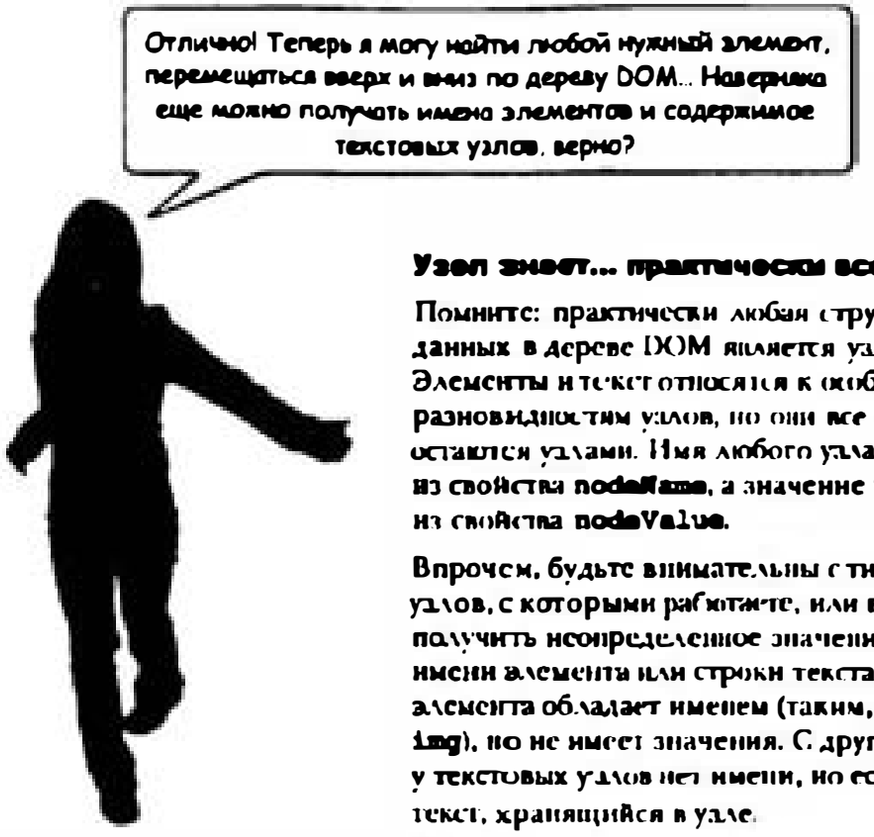


Перемещение по дереву DOM

Вы уже знаете, что объект **document** позволяет найти элемент с конкретным значением атрибута **id**, однако на практике существуют и другие способы перехода к нужному элементу дерева DOM. Каждый узел обладает родителем, а у многих узлов элементов имеются потомки; эти отношения могут использоваться для перемещения вверх и вниз по дереву DOM.

Далее приводится часть дерева DOM со с. 226; давайте посмотрим, как может происходить перемещение по дереву, начинающееся с одного из элементов `<div>`.



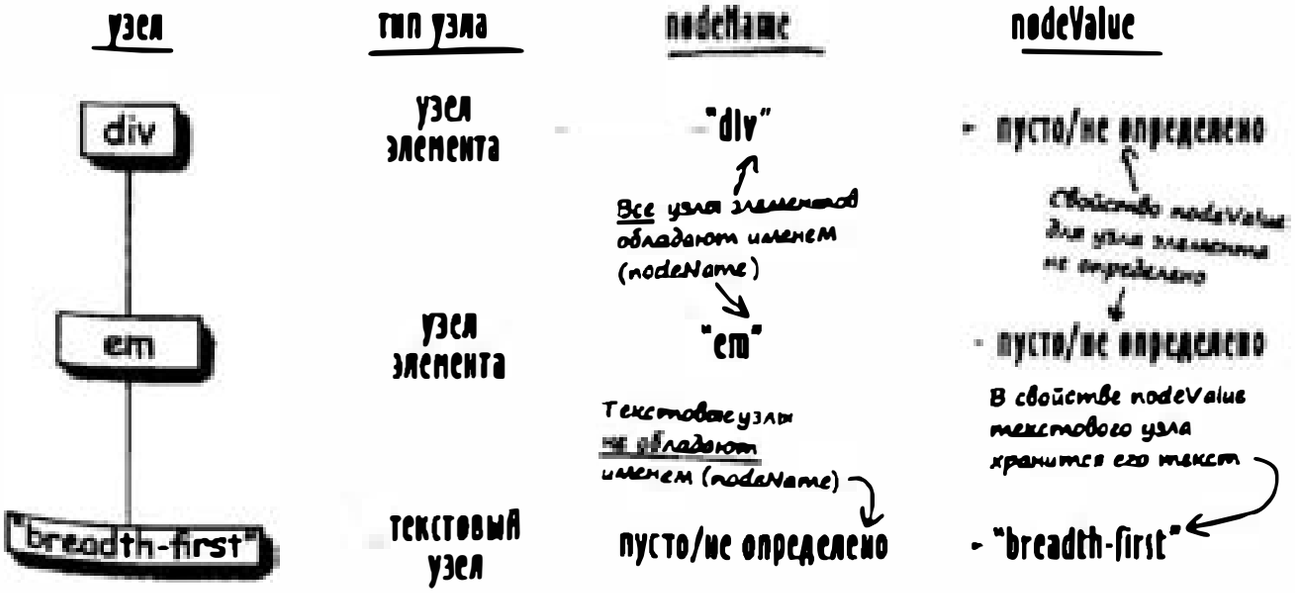


Узел знает... практически все

Помните: практически любая структура данных в дереве DOM является узлом. Элементы и текст относятся к особым разновидностям узлов, но они все равно остаются узлами. Имя любого узла читается из свойства **nodeName**, а значение узла — из свойства **nodeValue**.

Впрочем, будьте внимательны с типами узлов, с которыми работаете, или вы рискуете получить неопределенное значение вместо имени элемента или строки текста. Узел элемента обладает именем (таким, как **div** или **img**), но не имеет значения. С другой стороны, у текстовых узлов нет имени, но есть значение: текст, хранящийся в узле.

Давайте посмотрим, как это делается.



Всегда учитывайте тип узла, с которым вы работаете, **и следите за тем, на какой узел дерева DOM ссылаются переменные. Для тренировки предлагаем вам немного кода JavaScript и HTML. Ваша задача — разобраться, что будет** выведено при вызове **функции alert(). Сначала попробуйте предположить результат... но если** возникнут **трудности — введите код и протестируйте его.**

```
function guess () {
 var whatAmI;
 var element =
   document.documentElement.lastChild;
 alert ("I am a " + element.nodeName);
 var anotherElement =
   document.getElementsByTagName ("h1")[0];
 alert("I am a " + anotherElement.nodeValue);
 var child = anotherElement.firstChild;
 alert("I am a " + child.nodeValue);
 element =
   document.getElementById("tiger").lastChild;
 alert("I am a " + element.nodeValue);
 alert("I am a " +
   element.parentNode
          .getAttributeNode ("id").nodeValue);
}
```

Напишите, что выведет функция alert().

Код HTML, используемый в упражнении

```
<html>
 <head>
  <title>Who Am I exercise</title>
 </head>
 <body>
  <h1>I am a cow</h1>
  <div id="ranch">
   I am a <em>horse</em>, but I wish I
   was a <span id="tiger">tiger</span>.
  </div>
  <form>
   <input type="button" value="What Am I?"
          onClick="guess();" />
  </form>
 </body>
</html>
```

Ответ на с. 240.

Часто задаваемые вопросы

В: Я вспомнил, что такое текст, элементы и затем... но еще не понял, что за такое «узел»?

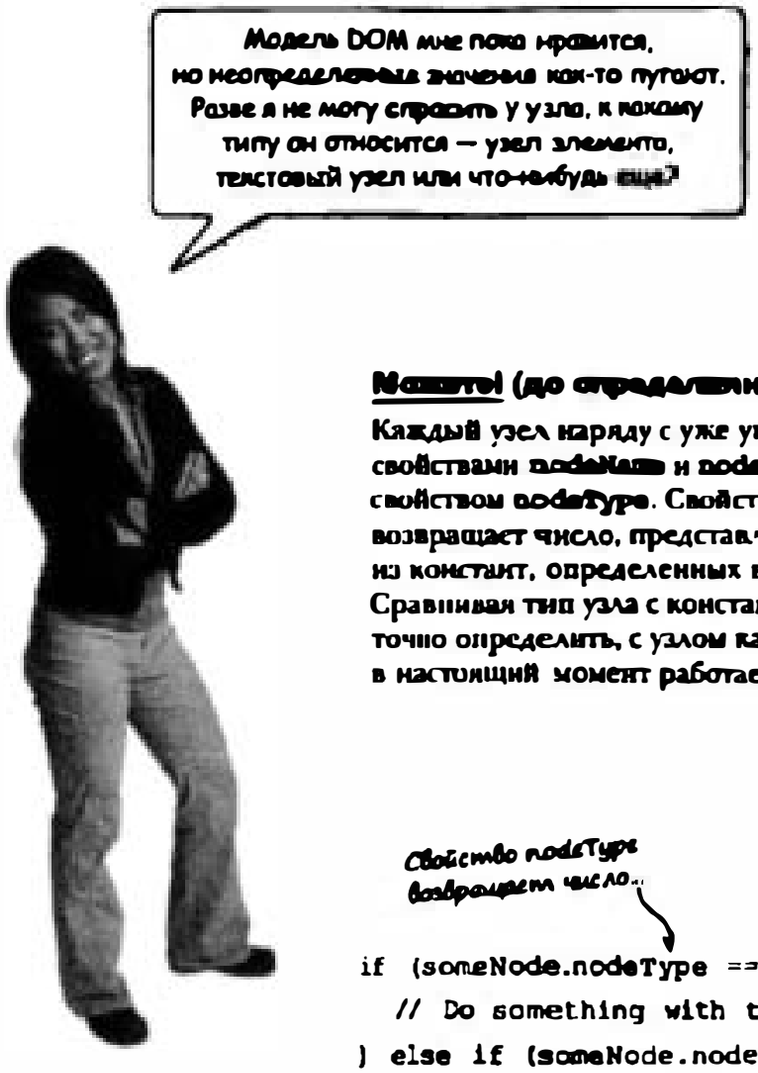
О: Узлом в DOM называется все: элементы, текст, атрибуты и даже комментарии. Каждый фрагмент разметки обладает стандартными свойствами (потомки, родитель), а в большинстве случаев DOM группирует эти общие свойства в объект Node. Затем для каждого типа узла набор общих свойств типа Node дополняется уникальными свойствами.

Например, узлы элементов поддерживают методы `getAttribute()` и `setAttribute()`, потому что атрибутами могут обладать только элементы. С другой стороны, свойства parent и childNodes передаются узлам элементов от объекта Node, так как данная функциональность является общей для разных типов элементов.

В: Но некоторые узлы не имеют имени. Что произойдет, если использовать свойство nodeName или что-нибудь в этом роде для текстового узла?

О: При попытке обратиться к свойству узла, который этим свойством не обладает, вы получите «неопределенное» значение. Так модель DOM сообщает вам, что она не понимает, что вы имели в виду — или по крайней мере о том, что ваши намерения не соответствуют дереву DOM, построенному браузером на основе вашей разметки.

Следовательно, при попытке обратиться к свойству nodeName для текстового узла вы получите неопределенное значение — ведь текстовые узлы, в отличие от узлов элементов, не обладают [illegible]. С другой стороны, при обращении к свойству nodeValue элемента вы также получите неопределенное значение, так как элементы не обладают значениями. Элемент может обладать атрибутами, но любой текст в элементе будет размещаться в одном из узлов-потомков элемента, и обратиться к нему через свойство nodeValue элемента не удастся.



Машите! (до определенной степени)

Каждый узел наряду с уже упоминавшимися свойствами **nodeName** и **nodeValue** обладает свойством **nodeType**. Свойство **nodeType** возвращает число, представленное одной из констант, определенных в классе **Node**. Сравнивая тип узла с константами, вы можете точно определить, с узлом какого типа вы в настоящий момент работаете:

Свойство nodeType возвращает число...

...которое сравнивается с константами, определенными в классе Node

```
if (someNode.nodeType == Node.ELEMENT_NODE) {
  // Do something with the element node
} else if (someNode.nodeType == Node.TEXT_NODE) {
  // Do something with the text node
}
```

Константа определена для всех типов узлов, включая узлы комментариев, текстовые узлы и узлы атрибутов

Зная тип узла, вы не будете обращаться к свойствам, возвращающим неопределенные значения

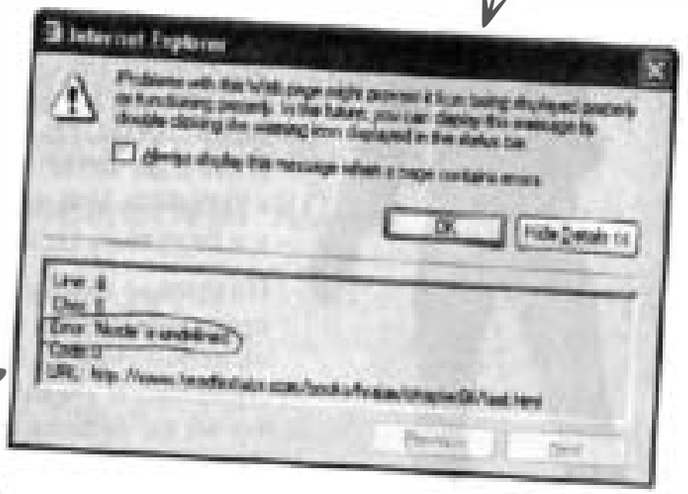
Некоторые браузеры не поддерживают Node

К сожалению, некоторые браузеры не поддерживают класс Node в коде JavaScript. Код не работает, и вам снова приходится иметь дело с ошибками и неопределенными значениями.

Все браузеры поддерживают свойство nodeType.

...но некоторые браузеры выдают сообщение об ошибке в этом месте

```
if (someNode.nodeType == Node.ELEMENT_NODE) {
  // Do something with the element node
} else if (someNode.nodeType == Node.TEXT_NODE) {
  // Do something with the text node
}
```



IE сообщает, что он не распознает ссылку на объект Node в вашем коде

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

В: Значит, если мои посетители не работают с Internet Explorer, я смогу использовать объект Node?

О: Вообще говоря, объект Node лучше вообще не использовать... по крайней мере до тех пор, пока он не будет поддерживаться всеми основными браузерами. Даже если вы считаете, что ваши пользователи не работают с IE, этот браузер остается самым популярным в мире (притом с большим отрывом). В следующей главе вы увидите, как добиться того же результата без использования Node, объем работы несколько увеличивается, но зато полученный код будет работать во всех браузерах.

Здорово! У DOM есть свои странности, но я помаленьку начинаю осваиваться. А что с тем Великим испытанием, о котором вы упоминали?

Вы готовы к испытанию...

DOM — весьма обширная тема, и нам понадобилось почти 40 страниц для объяснения принципов, по которым работает модель DOM. Но теперь вы почти стали асом DOM, почти готовы к созданию приложений DOM... и к Великому Испытанию.

Но прежде чем браться за него, проверьте ответы к упражнениям на нескольких ближайших страницах и убедитесь в том, что вы все поняли правильно. Затем обратитесь к главе 4.5, и мы приступим к работе над приложением DOM.

Да, это не опечатка. В книге есть глава 4.5. И вы готовы к тому, чтобы прочитать ее. Закройте глава, несколько раз произнесите мантру «D-O-M», и переходите к программированию.

Великое испытание для главы 4

Написать веб-приложение, использующее модель DOM для создания динамического интерфейса пользователя, без применения кода Ajax.

* Ладно признаемся: глава получилась настолько большой, что мы решили разбить ее на две. Но тогда Великое испытание для главы 4 превращалось в Великое испытание для главы 5, а это никому не нравилось. Поэтому мы присвоили следующей главе номер 4.5, и теперь можем с полным правом сказать...

...(барабанная дробь)

Великое испытание для главы 4. Кто сказал, что в книгах по программированию нет места сильным эмоциям?

Просто сделайте это — решение

Всегда учитывайте тип узла, с которым вы работаете, и следите за тем, на какой узел дерева DOM **ссылаются переменные. Ваш ответ совпал с нашим?** Давайте проверим.

```
function guess() {
  var whatAmI;
  var element =
    document.documentElement.lastChild;
  alert("I am a " + element.nodeName);
  var anotherElement =
    document.getElementsByTagName("h1")[0];
  alert("I am a " + anotherElement.nodeValue);
  var child = anotherElement.firstChild;
  alert("I am a " + child.nodeValue);
  element =
    document.getElementById("tiger").lastChild;
  alert("I am a " + element.nodeValue);
  alert("I am a " +
    element.parentNode
           .getAttributeNode("id").nodeValue);
}
```

Основным элементом документа является элемент <html>. Его первый потомок — элемент <head>, а последний — элемент <body>

Документ содержит только один элемент <h1>...

...а элементы не обладают свойством nodeValue

Первым (и единственным) потомком <h1> является текстовый узел с текстом «I am a cow»

Вы уже привыкли к такому коду

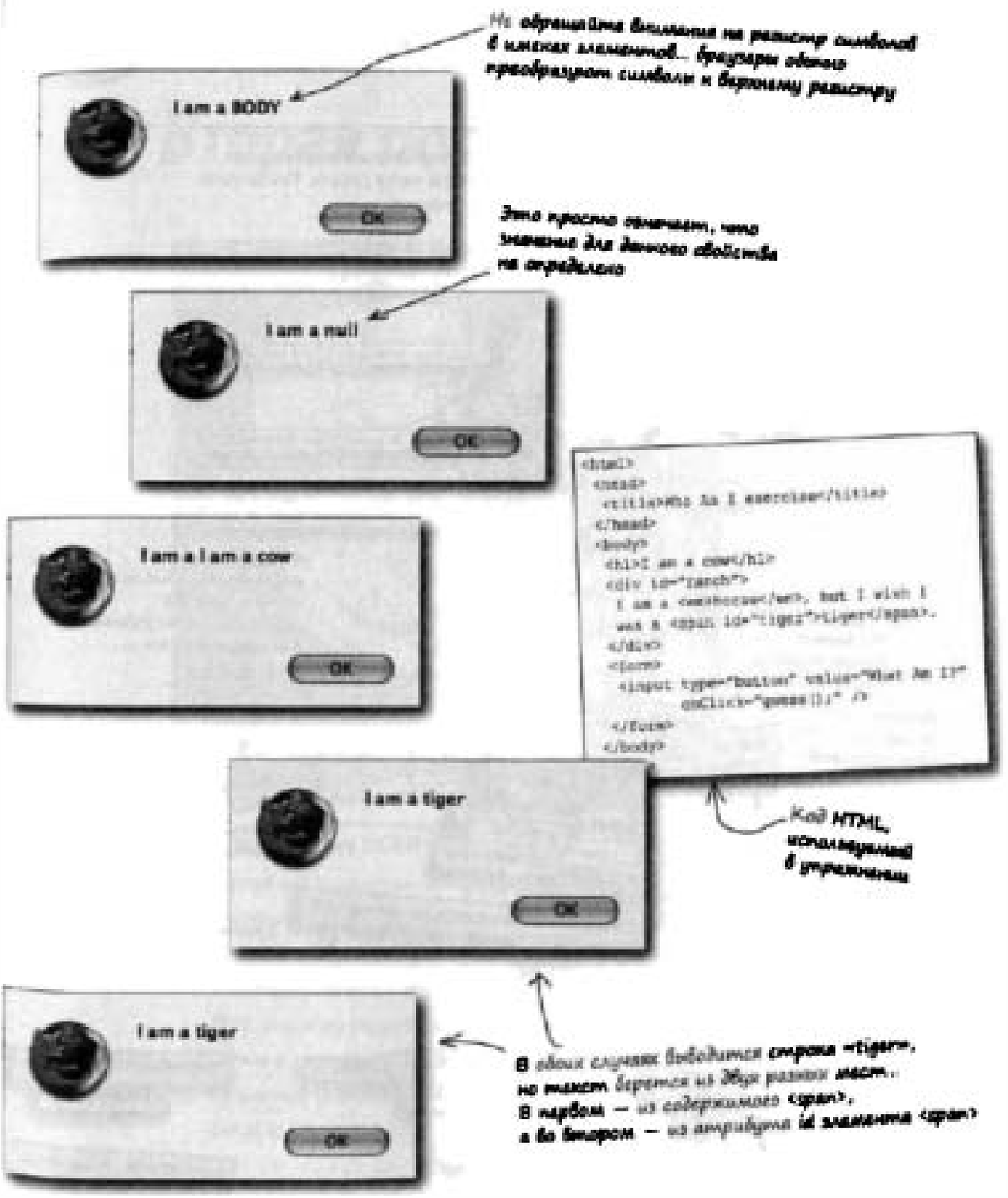
Последним потомком элемента является его текст, то есть строка «tiger»

Родителем текстового узла является элемент

Ранее мы еще не встречали эту функцию, но как нетрудно догадаться, она возвращает атрибут «id»...

то есть значение «tiger»

Вы правильно ответили на последний вопрос? Так легко забыть, что переменная «element» в действительности элементом не является! Это текстовый узел, а его родителем является элемент .
Если вы думали, что в последней строке мы получаем элемент с атрибутом id, равным «ranch», ничего страшного...
Но прежде чем читать дальше, обязательно убедитесь в том, что вы поняли, почему функция alert выводит строку «tiger».


Не обращайте внимания на регистр символов в именах элементов... браузеры обычно преобразуют символы к верхнему регистру
I am a BODY
OK
Это просто означает, что значение для данного свойства не определено
I am a null
OK
I am a I am a cow
OK
<html>
<head>
<title>Who Am I exercise</title>
</head>
<body>
<h1>I am a cow</h1>

I am a cow, but I wish I
was a tiger.

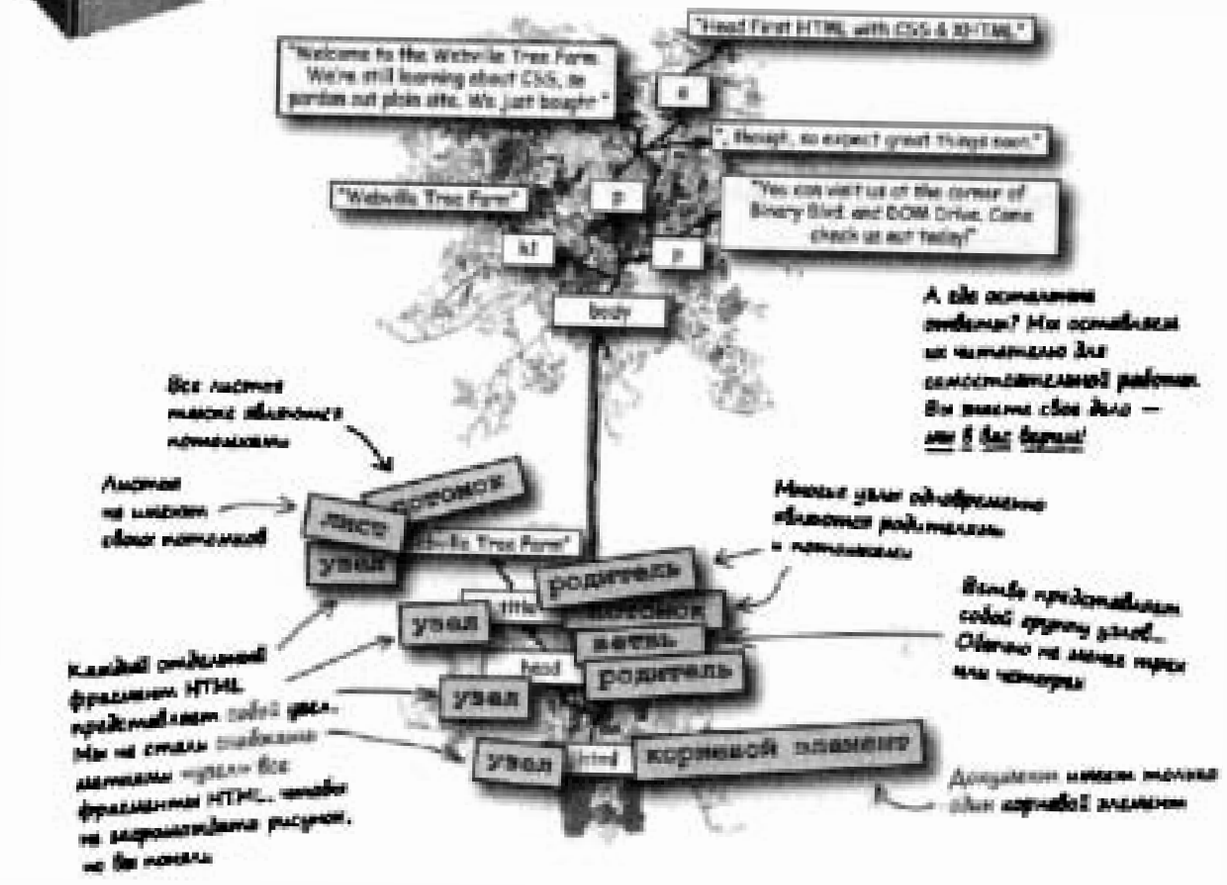
<form>
<input type="button" value="What Am I?"
onClick="guess()" />
</form>
</body>
Код HTML, используемый в упражнении
I am a tiger
OK
I am a tiger
OK
В обоих случаях выводится строка «tiger», но текст берется из двух разных мест... В первом — из содержимого , а во втором — из атрибута id элемента

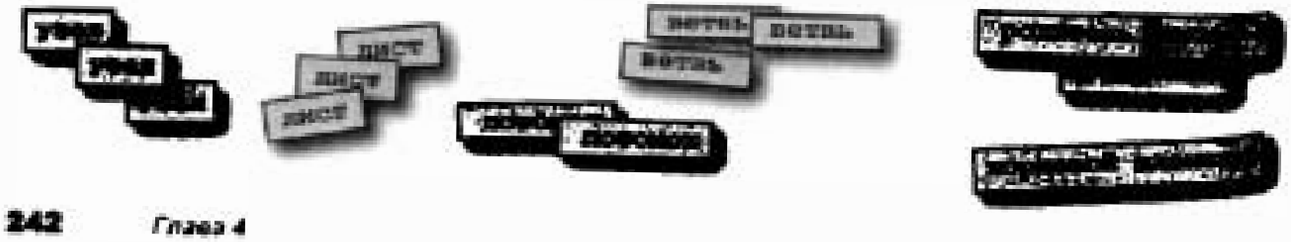
Решение упражнений

Разложите по полкам

Решения

Вот простая страница HTML в представлении браузера. Мы разместили метки у каждой части дерева. Посмотрите, совпадут ли ваши ответы с нашими.





4.5 Разработка DOM-приложений

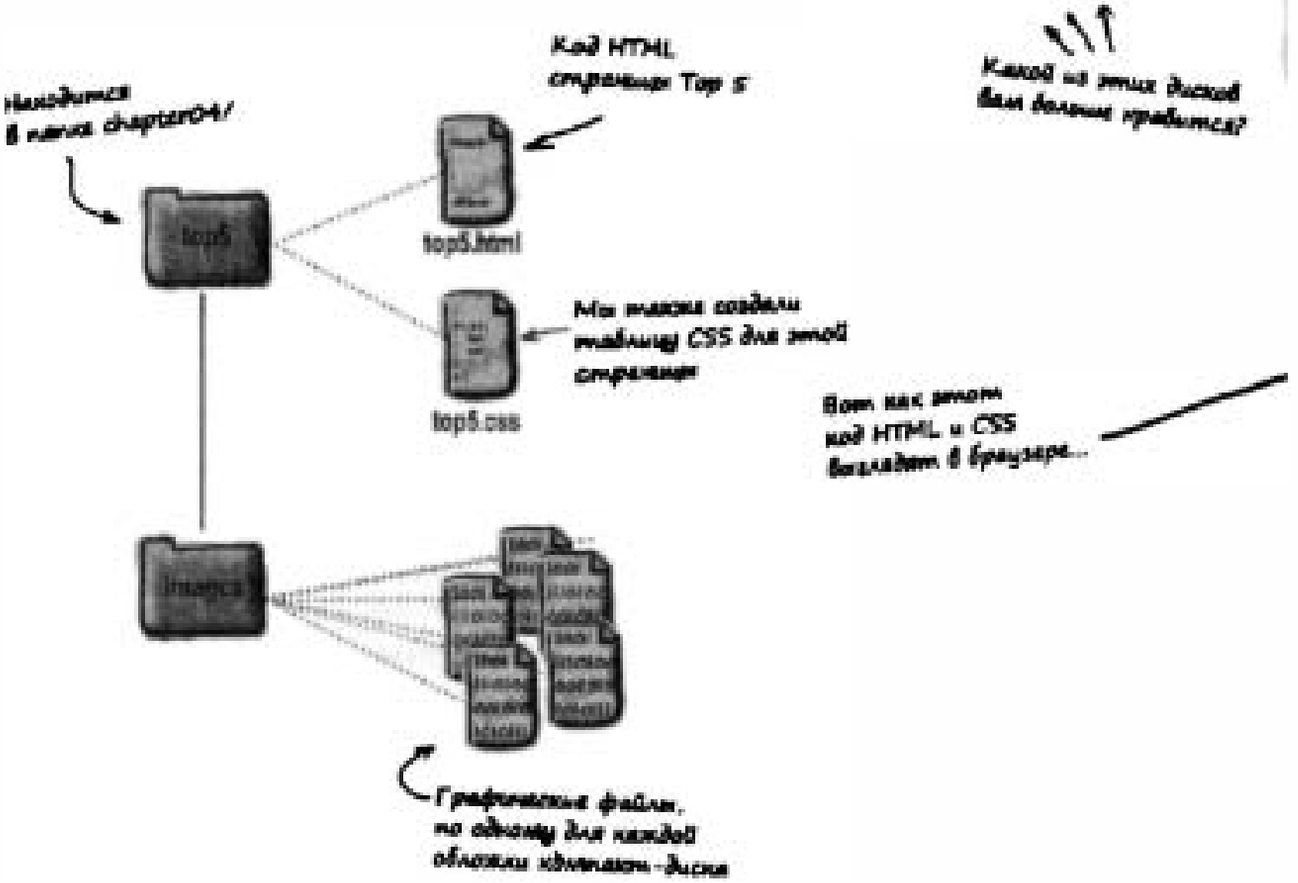
Вторая порция

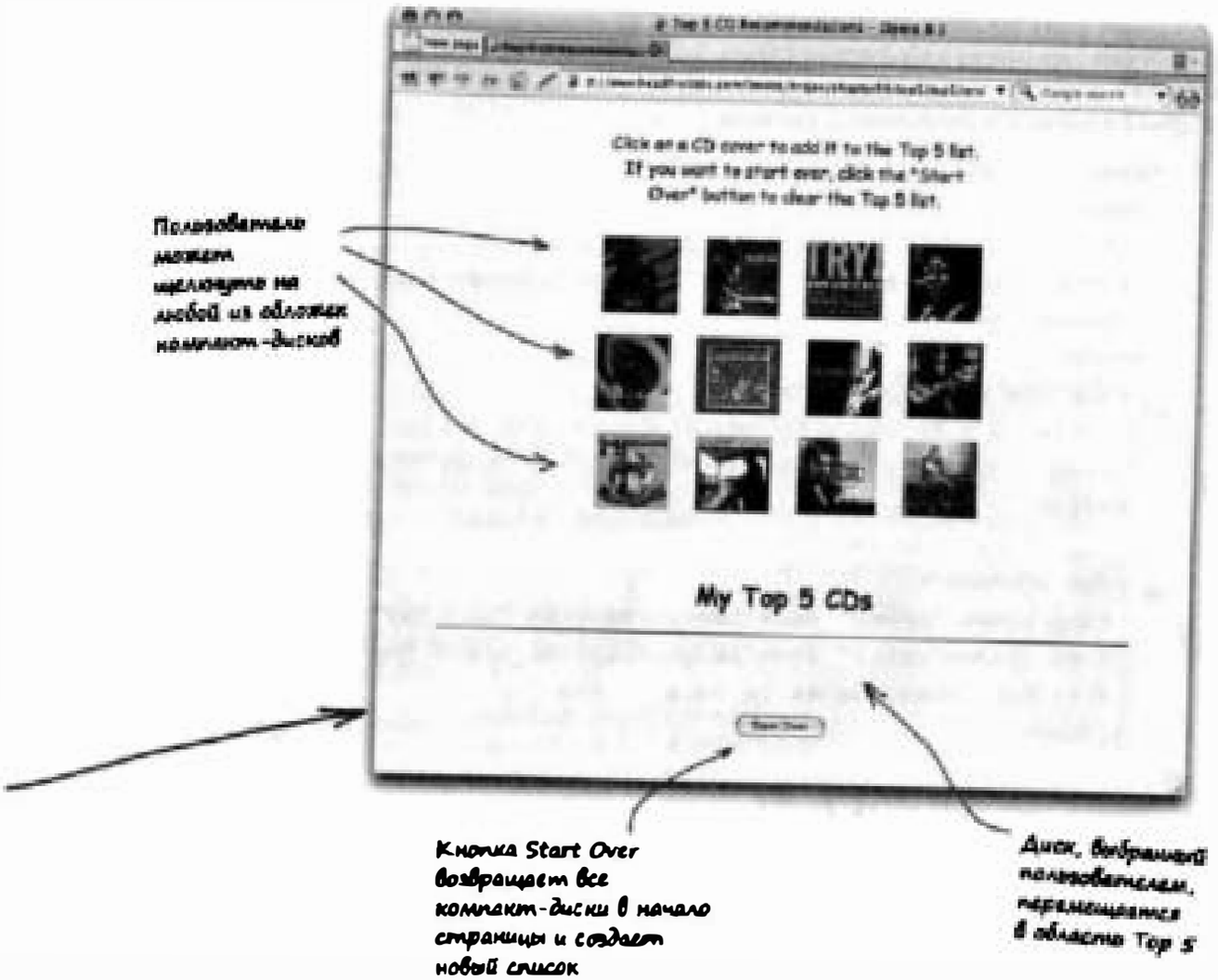
Кому добавки DOM? В предыдущей главе был приведен краткий курс по самой интересной технологии обновления веб страниц **DOM** (Document Object Model). Но скорее всего, ваш аппетит еще не удовлетворен, поэтому в этой главе мы используем уже имеющуюся информацию для написания приложения на базе DOM. Заодно мы рассмотрим несколько [illegible], научимся изменять стиля узлов и создадим **двунаправленное, динамическое приложение**. В этой главе ваши навыки работы с DOM поднимутся на совершенно новый уровень.

Каждый в душе критик

Вероятно, каждый любитель музыки хорошо представляет, что стоит послушать... а что не стоит. Давайте воспользуемся тем, что мы узнали о модели DOM, и построим веб-страницу с информацией о 5 лучших блюзовых компакт-дисках всех времен.

Чтобы читателю не пришлось тратить время на написание кода HTML и CSS, мы заранее подготовили несложную страницу и добавили стили. Откройте папку chapter04 в архиве примеров книги. Вы найдете в ней другую папку с именем top5, содержащую несколько файлов и вложенную папку:





ШЕВЕЛИН НОЗГАНИ

Можно ли написать это приложение без применения DOM? Чем оно будет отличаться?

Знакомимся с кодом top5.html

Откройте файл top5.html и просмотрите его код. Большая часть файла уже заполнена элементами `<img>` для отображения обложек компакт-дисков. Также в файл входят инструкции, форма с кнопкой и несколько элементов `<div>`:

Как обычно, в этом приложении используется внешняя таблица стилей CSS

```
<html>
 <head>
  <title>Top 5 CD Recommendations</title>
  <link rel="stylesheet" type="text/css" href="top5.css" />
 </head>
 <body>
  <div id="instructions">
   Click on a CD cover to add it to the Top 5 list. If you want to start
   over, click the "Start Over" button to clear the Top 5 list.
  </div>

  <div id="cds">
   <img class="cover" src="images/vaughan_flood.jpg" />
   <img class="cover" src="images/clapton_cream.jpg" />
   <!-- Lots more images in here... -->
  </div>

  <div id="top5-listings">
   <h2>My Top 5 CDs</h2>
   <div id="top5"></div>
  </div>

  <form>
   <input type="button" value="Start Over" />
  </form>
 </body>
</html>
```

Ничего сложного... Набор элементов `<img>` для отображения обложек дисков

Сюда помещаются компакт-диски, выбранные пользователем

Вероятно, при нажатии этой кнопки должен выполняться код JavaScript...

Все эти атрибуты пригодятся при поиске элементов в нашем коде

Что предстоит сделать?

Код HTML и CSS уже написан; остается только написать код JavaScript. Давайте разберемся, что необходимо сделать, и напишем часть кода.

Самая простая часть. Назовем новый файл top5.js

1. **Создать новый файл для хранения кода JavaScript страницы.**
2. **Написать функцию, которая при щелчке на обложке включает диск в список Top 5.**

Для этого с каждой обложкой будет связан обработчик события onClick(). Затем мы напишем новую функцию с именем addToTop5()

3. **Снабдить каждый диск рейтингом, чтобы пользователь видел, в каком порядке следуют диски в «верхней пятерке».**

DOM сильно упрощает решение подобных задач. Соответствующий код также включается в функцию addToTop5()

4. **Написать функцию для отмены действий пользователя и повторного заполнения списка.**

Обработчик события onClick() кнопки Start Over запустит другую функцию JavaScript, которую мы назовем startOver()

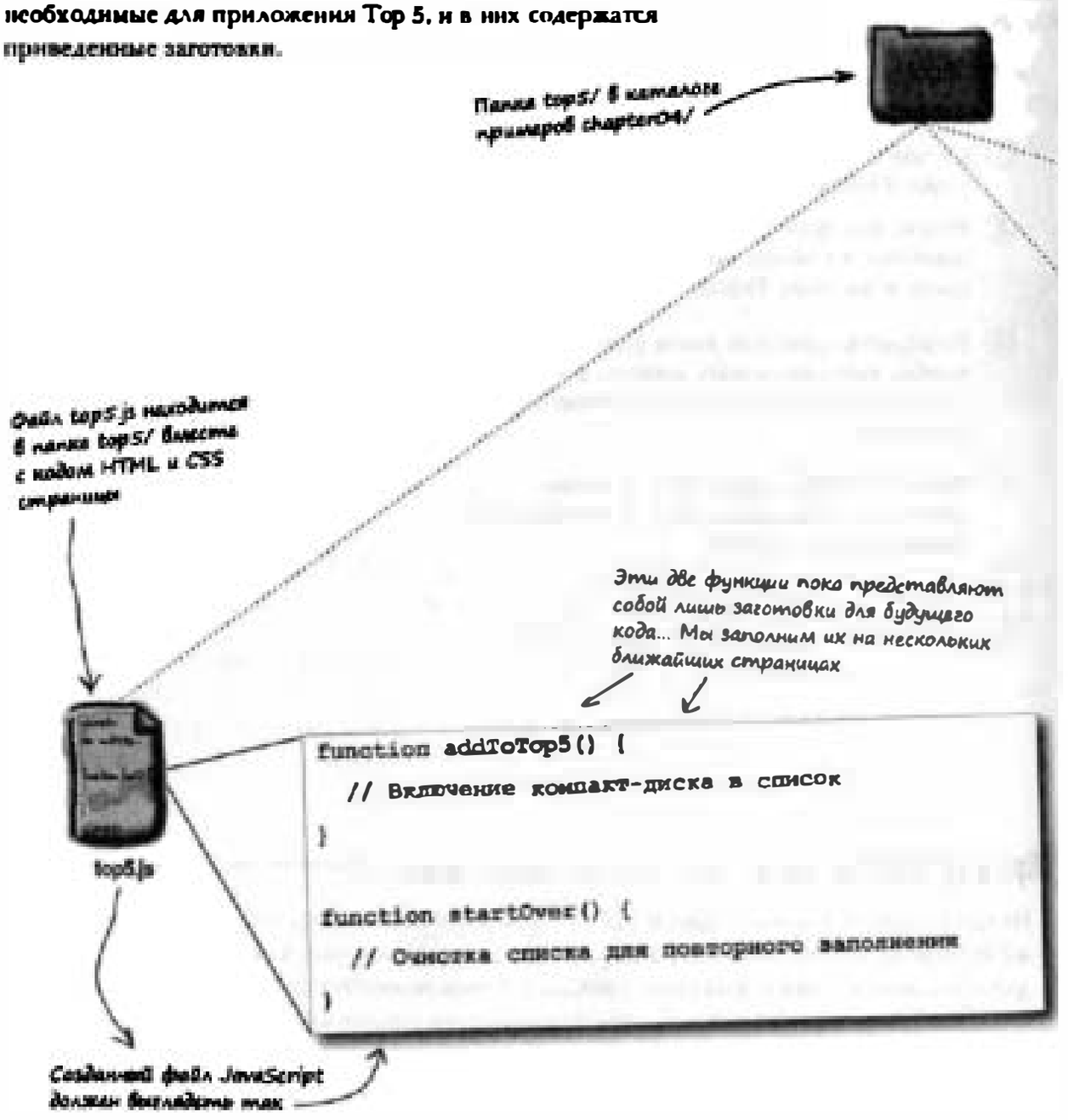
Просто сделайте это

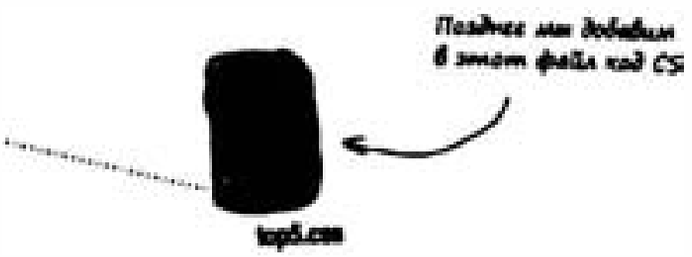
Начнем с шага 1. Создайте новый файл с именем top5.js. Включите в него функцию **addToTop5()** для выполнения шага 2, затем еще одну функцию **startOver()** для выполнения шага 4. Код шага 3 войдет в функцию **addToTop5()**, отдельная функция для него не потребуется. Все функции пока остаются пустыми; мы заполним их позднее в этой главе.

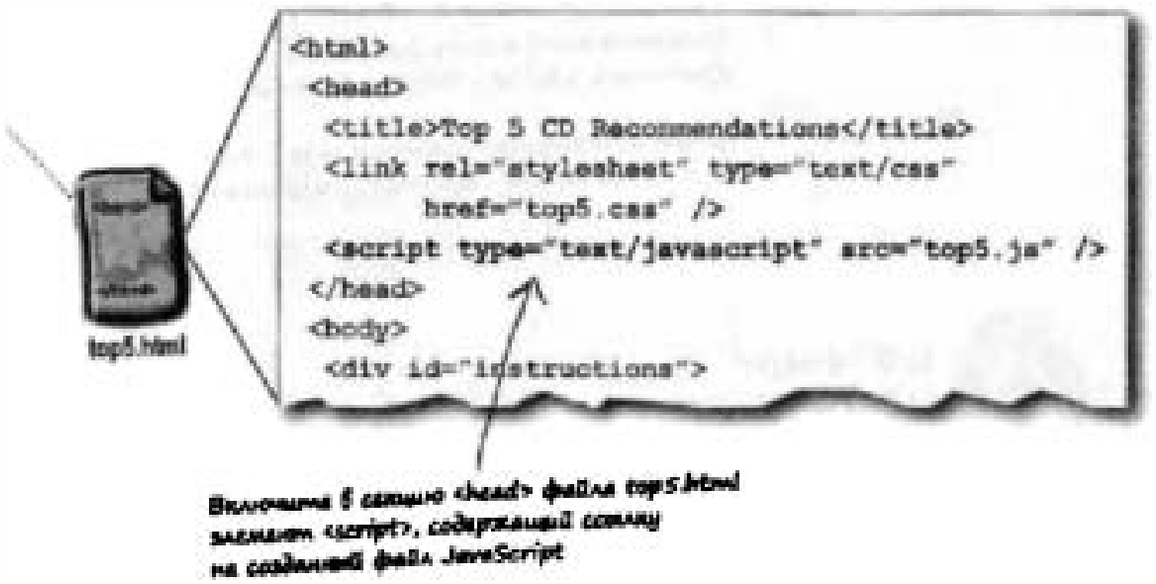
Внесите все перечисленные изменения и сохраните файл JavaScript под именем top5.js в каталоге top5, наряду с файлами top5.html и top5.css.

Общая картина

Вы уловили суть происходящего? Обратитесь к следующей диаграмме; убедитесь в том, что у вас имеются все файлы, необходимые для приложения Top 5, и в них содержатся приведенные заготовки.




Позднее мы добавим
в этот файл код CSS
top5.com


<html>
<head>
<title>Top 5 CD Recommendations</title>
<link rel="stylesheet" type="text/css"
href="top5.css" />
<script type="text/javascript" src="top5.js" />
</head>
<body>

top5.html
Включение в секцию <head> файла top5.html
элемента <script>, содержащего ссылку
на созданный файл JavaScript

Подготовка обложек

Каждый раз, когда пользователь щелкает на изображении обложки компакт-диска, страница Top 5 должна запускать функцию `addToTop5()`. Эта функция (после того, как мы напишем ее код) будет включать выбранный диск в список Top 5.

Задача может быть решена двумя способами:

Способ 1: Определение обработчика `onClick` для каждого элемента `<img>` в top5.html.

Плюсы:

- ✓ Просто реализуется. Просто включите строку `onClick="addToTop5();"` в каждый элемент `<img>` кода HTML.
- ✓ Не требует написания кода JavaScript.

Минусы:

- ✗ Вы должны помнить, что при изменении или добавлении изображений в страницу HTML необходимо добавлять обработчик события `onClick`.

Способ 2: Программное определение обработчиков событий для элементов `<img>` из кода JavaScript.

Плюсы:

- ✓ Гарантирует вызов `addToTop5()` для всех изображений (в том числе и для новых, добавленных позднее).
- ✓ Приложение может вызвать функцию для добавления обработчиков событий в любой удобный момент.

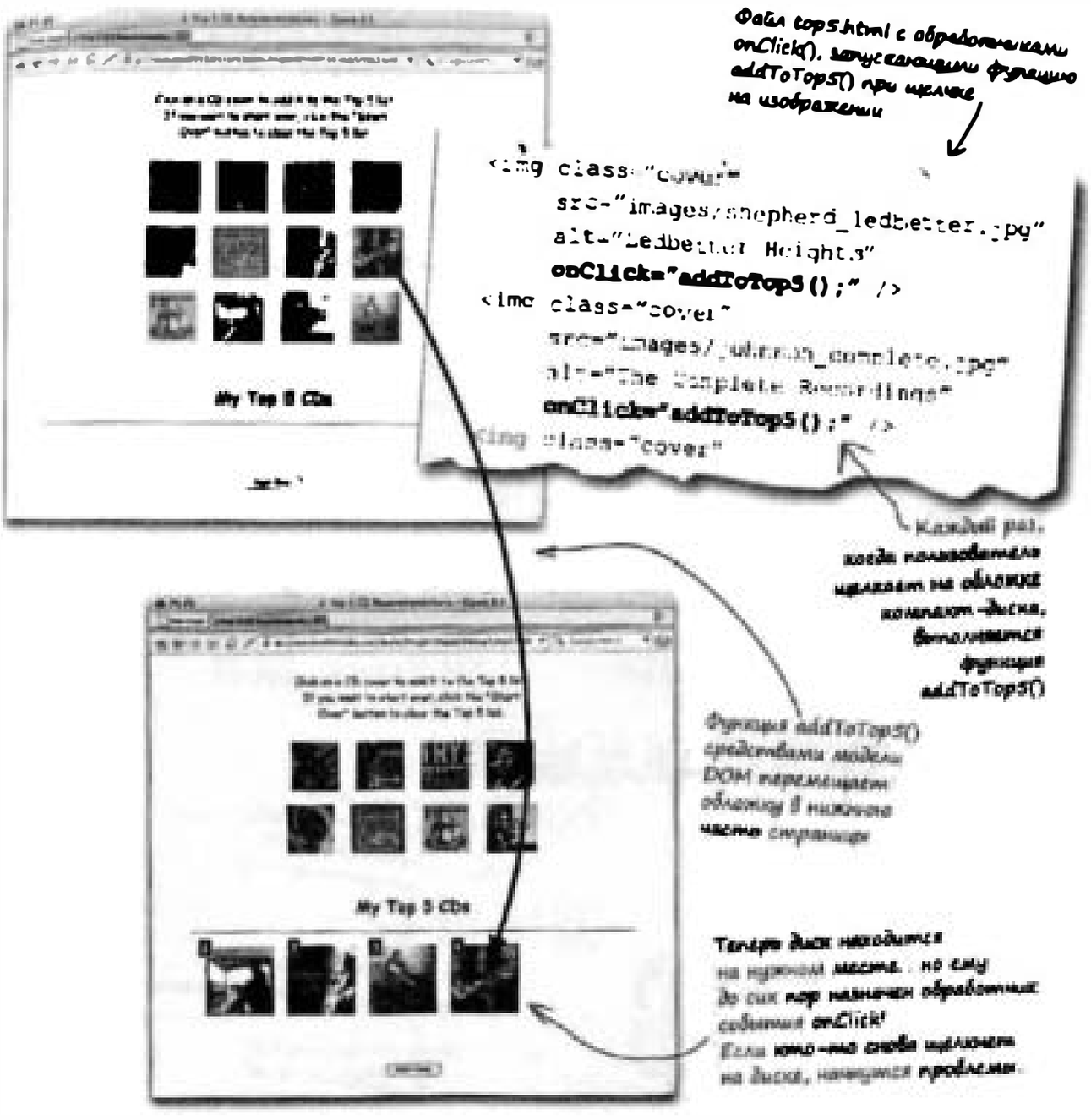
Минусы:

- ✗ Требует написания программного кода (в отличие от включения обработчиков `onClick` прямо в HTML).

ШЕВЕЛИМ МОЗГАМИ

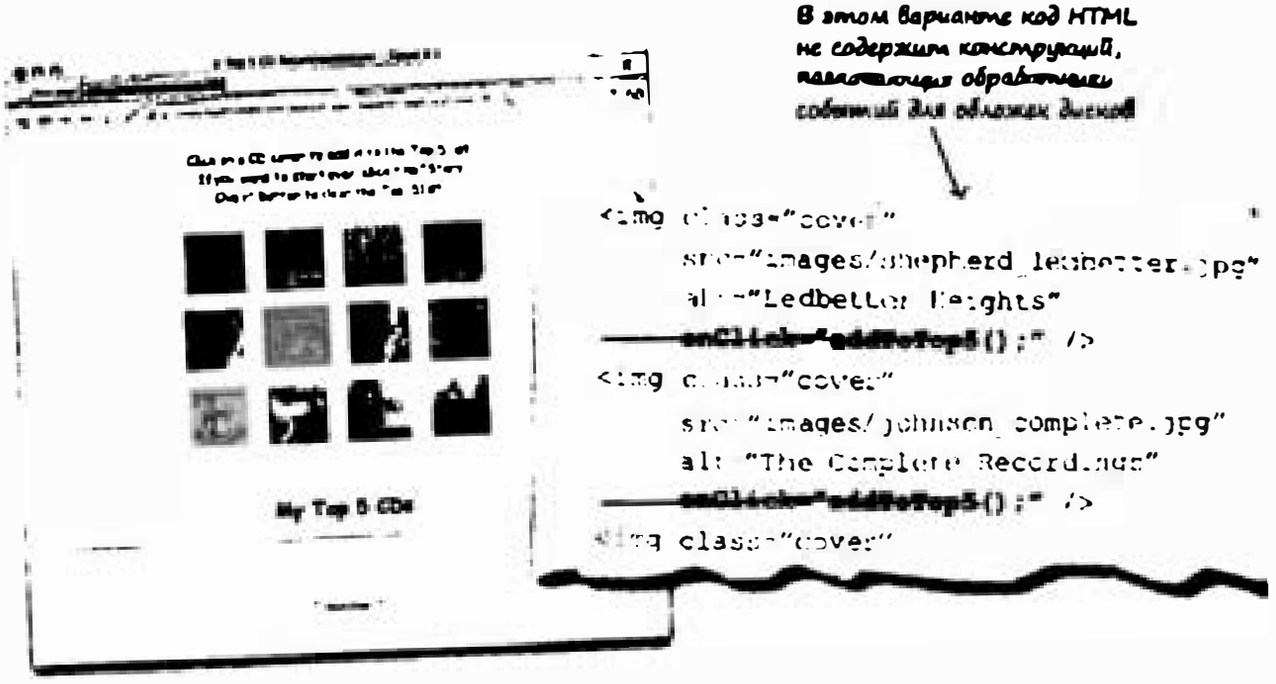
Как вы думаете, какой вариант лучше? Помните: после включения диска в список Top 5 при повторном щелчке на обложке ничего происходить не должно. В противном случае в список будет включен диск, который уже был включен в него ранее. Влияет ли это обстоятельство на выбор способа определения обработчиков `onClick` для элементов `<img>`?

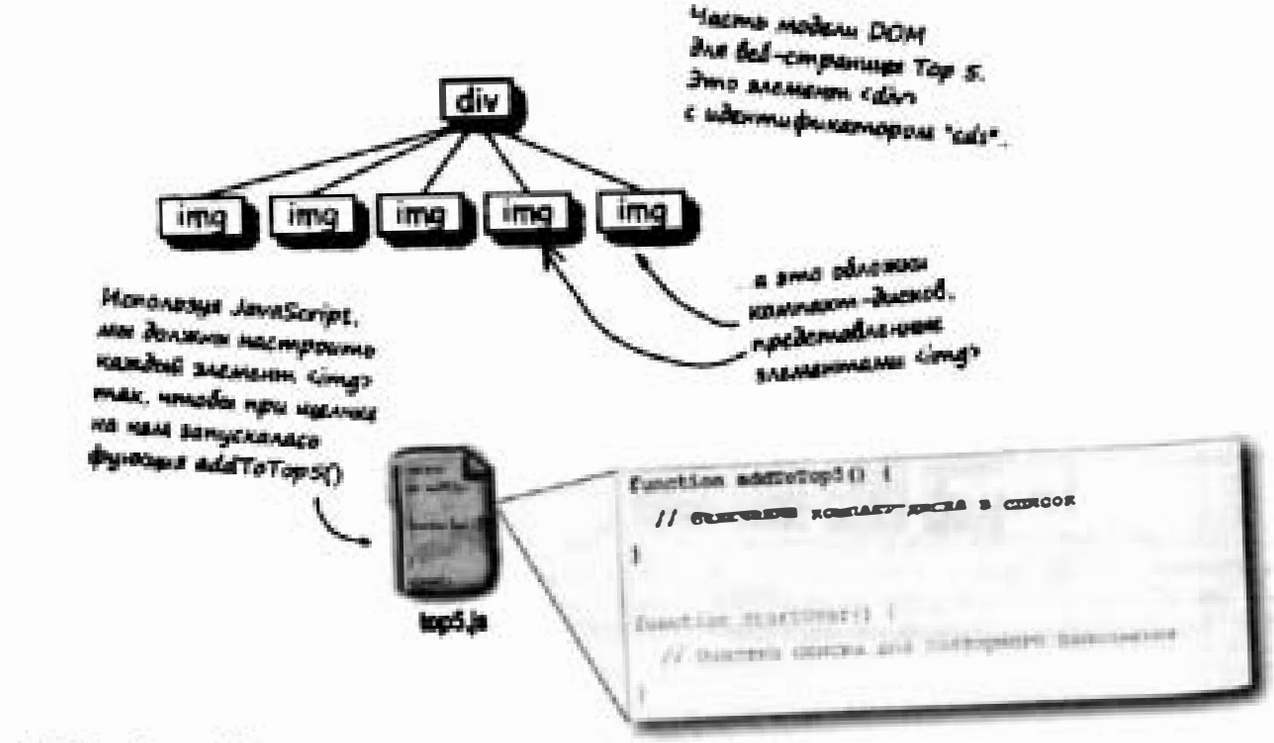
Способ 1: Включение обработчиков onClick во все элементы <img> в файле top5.html.



Лучше обратимся к способу 2...

Способ 2: Программное определение обработчиков событий для элементов <img> из кода JavaScript





Пожалуй, лучше воспользоваться JavaScript. При включении диска в «верхнюю пятерку» мы должны отключить обработчик события для этого элемента <img>... но когда пользователь щелкает на кнопке Start Over, изображение необходимо снова переместить в секцию "cds" и вернуть ему обработчик. Похоже, обработчик onClick будет проще назначать и удалять из JavaScript.

Правильно мыслите! Мы можем написать функцию, которая добавляет обработчик события onClick ко всем элементам <img> в секции "cds". При включении компакт-диска в «верхнюю пятерку» обработчик удаляется, а элемент <img> перемещается в секцию "top5". Когда пользователь щелкает на кнопке Start Over, мы должны вернуть все изображения в верхнюю секцию (с именем "cds"). Затем можно снова запустить функцию и назначать всем изображениям обработчики событий.

Добавление обработчиков событий

Вы уже использовали обработчики onClick и onChange в коде HTML для связывания функций JavaScript с событием. То же самое можно сделать на уровне кода JavaScript; заодно мы попрактикуемся в работе с моделью DOM. Давайте создадим новую функцию с именем addOnClickHandlers(), назначающую обработчики событий для всех элементов <img> в секции "cds".

В начале главы на с. 247 этого пункта не было, но он все равно входит в процедуру подготовки к включению диска в «фирменную плейлисту»

Включите эту функцию в top5.js

Все обложки дисков, которым мы хотим назначить обработчики, вложены в секцию "cds"

Команда возвращает массив элементов с именем "img" в секции "cds"

```
function addOnClickHandlers() {

  var cdsDiv = document.getElementById("cds");

  var cdImages = cdsDiv.getElementsByTagName("img");

  for (var i=0; i<cdImages.length; i++) {

    cdImages[i].onclick = addToTop5;

  }

}
```

Перебираем все элементы <img> и назначаем каждому из них обработчик события

"onClick" — событие JavaScript, соответствующее обработчику события HTML onClick.

addToTop5 — функция, которая должна выполняться для данного события

E: getElementsByTagName? Это еще что такое?

O: Объект JavaScript document, как и все узлы элементов, поддерживает метод с именем `getElementsByTagName()`. Метод возвращает все вложенные элементы с заданным именем. В данном случае мы хотим получить все элементы <img> из секции "cds", поэтому вызов принимает вид `getElementsByTagName("img")`. Помните: метод возвращает только элементы, вложенные в узел, для которого вызывается этот метод... в данном случае узел `cdsDiv`.

E: Чем отличается использование свойства onclick в JavaScript и атрибута onClick страницы HTML?

O: Никаких различий не существует. Тем не менее, как мы обсуждали несколько страниц назад, написание метода для настройки обработчиков событий обеспечивает чуть большую гибкость... причем этот метод можно вызвать несколько раз, что нам вскоре пригодится.

Не забывайте: браузер читает разметку из файла HTML, после чего текстовый файл более не используется. Разметка преобразуется в дерево DOM, и в дальнейшем вы работаете с деревом.

Выполнение addOnClickHandlers()

Обработчики событий должны быть назначены сразу же после загрузки страницы. К счастью, у элемента **<body>** имеется обработчик события **onLoad()**. Он позволяет выполнять функцию JavaScript при каждой загрузке страницы — именно то, что нам требуется.

Внесите следующее изменение в файл top5.html:

Также не забудьте внести изменения, описанные на последней странице, в файл top5.js.

```
<html>
 <head>
  <title>Top 5 CD Recommendations</title>
  <link rel="stylesheet" type="text/css"
        href="top5.css" />
  <script type="text/javascript" src="top5.js" />
 </head>
 <body onload="addOnClickHandlers();">
  <div id="instructions">
```

Функция, заданная в обработчике события onLoad(), будет запускаться при каждой загрузке страницы HTML в браузере

Функция addOnClickHandlers() запускается при загрузке страницы. Когда пользователь щелкает на изображении обложки, будет вызываться функция addToTop5()

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

В: onClick, onChange теперь onLoad... Откуда берутся все эти обработчики? Вы достали их, сказано фокусник из шляпы!

О: Если вы не знакомы с этими обработчиками событий, не волнуйтесь... Информацию о них можно найти в любом нормальном справочнике по JavaScript. По мере углубления знакомства с JavaScript мы начнем знакомиться с разными обработчиками событий и тем, что они делают. Пока просто доверьтесь нам.

В: Разве не лучше будет воспользоваться методом addEventHandler(), чем работать со свойством onclick напрямую?

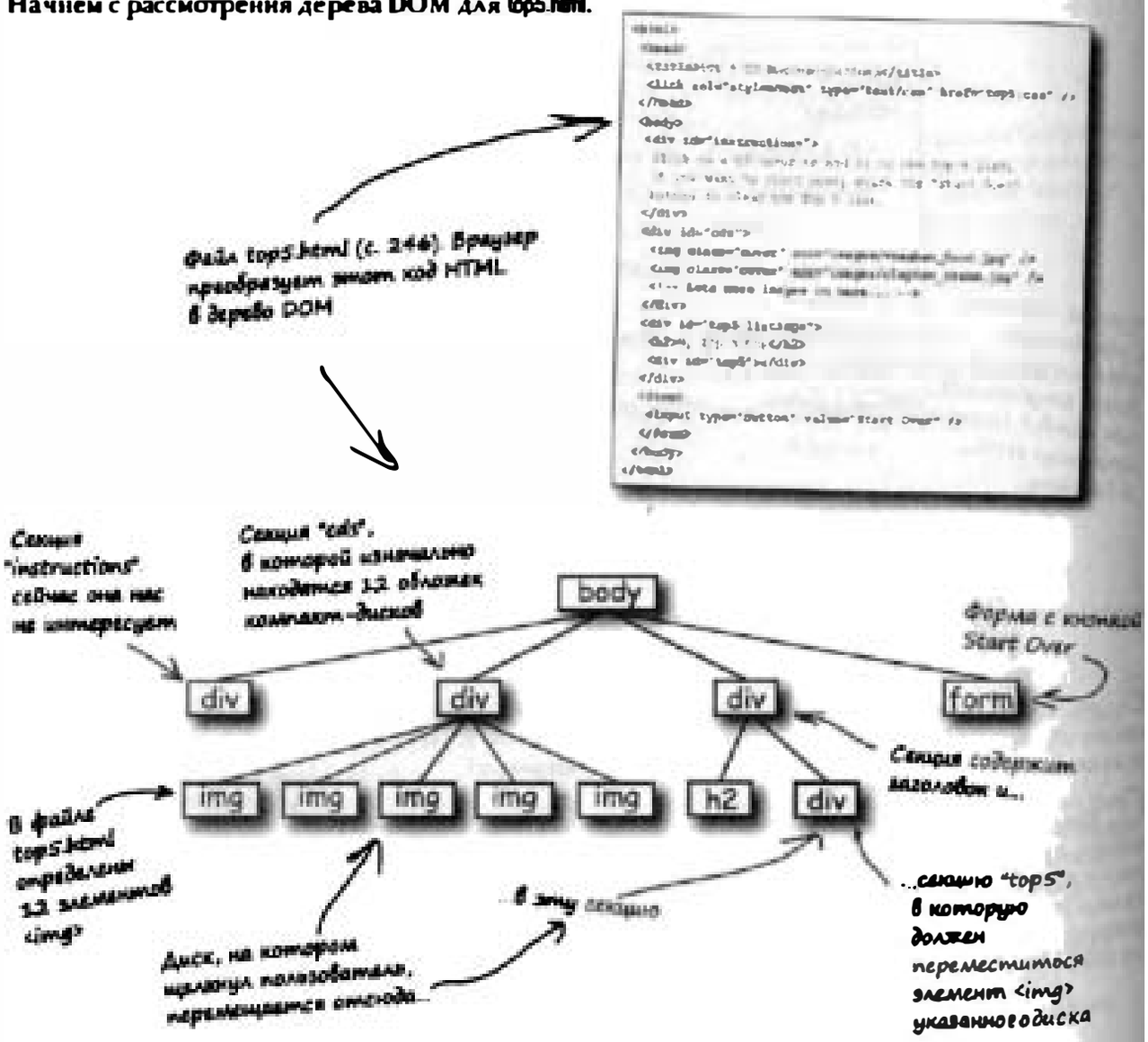
О: Метод `addEventHandler()` используется для добавления обработчиков событий элементов, и работает примерно так же, как свойство `onclick`. Тем не менее `addEventHandler()` не работает в Internet Explorer, тогда как свойство onclick работает во всех современных браузерах. Нужно ли говорить больше?

Кстати говоря, свойство называется «onclick», а не «onClick». При использовании любых символов, кроме строчных, вы получите сообщение об ошибке.

Включение диска в список

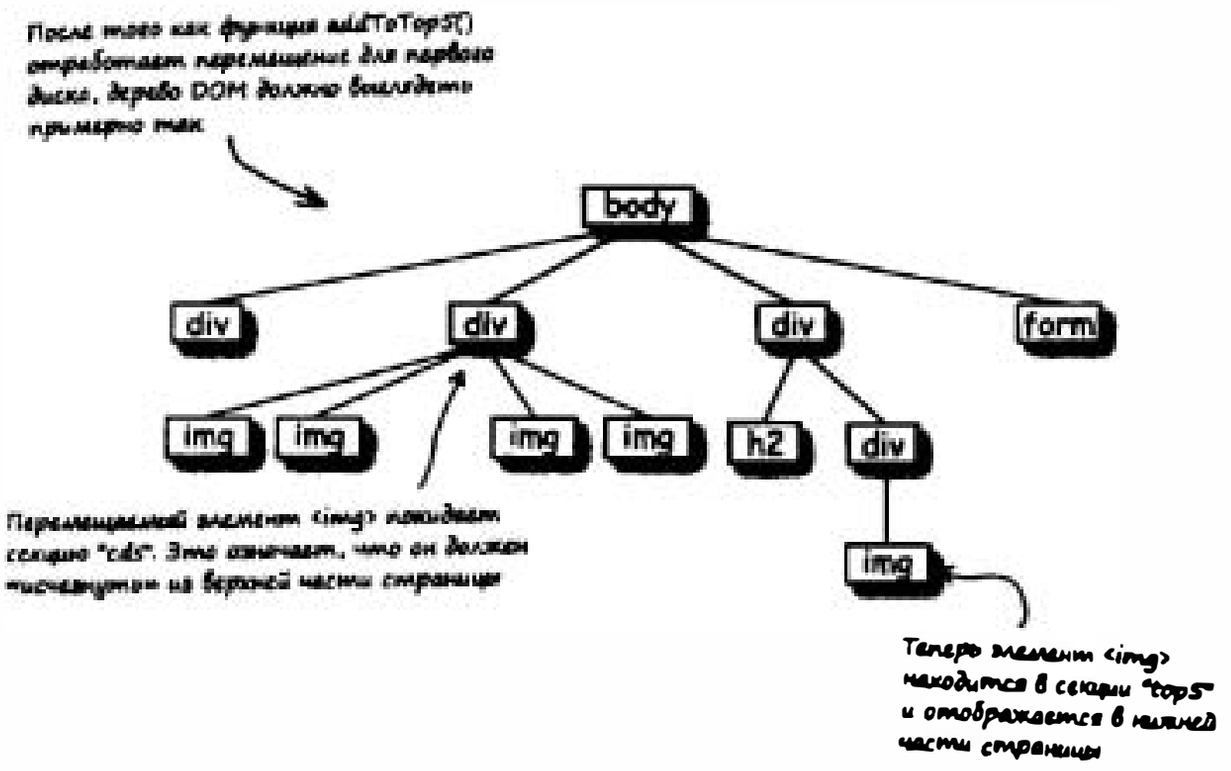
Итак, щелчок на обложке диска приводит к запуску функции **addToTop5()**. Пора заставить эту функцию работать, не правда ли? Диск, на котором щелкнул пользователь, должен перемещаться из секции ** ** "cds" в верхней части экрана в список «верхней пятерки», находящийся внизу. Благодаря DOM такое перемещение выполняется проще простого.

Начнем с рассмотрения дерева DOM для top5.html.



После выполнения функции addToTop5()...

Щелчок на обложке компакт-диска запускает функцию **addToTop5()**
Мы хотим, чтобы эта функция JavaScript переместила элемент **<img>**,
на котором щелкнул пользователь, из секции "cds" в секцию "top5".
Вот как должен выглядеть конечный результат:



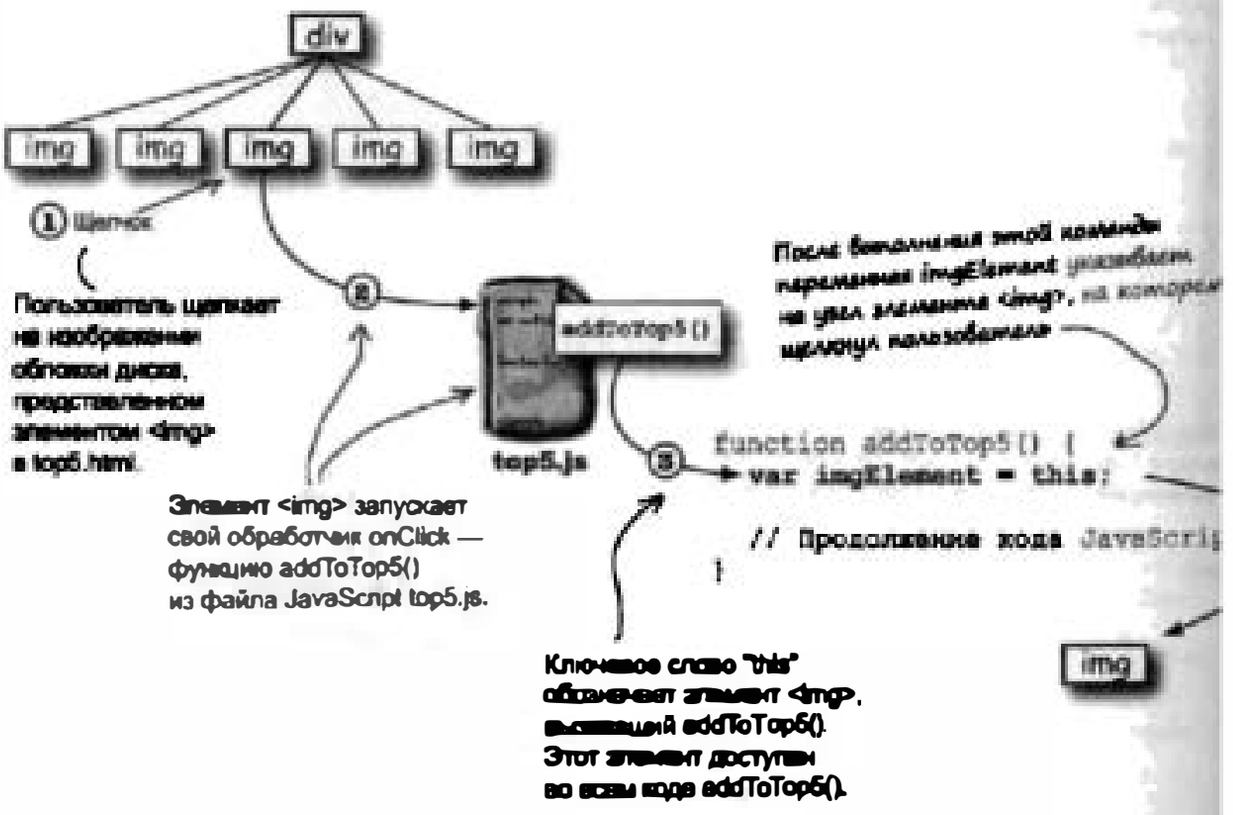
ШЕВЕЛИМ МОЗГАМИ

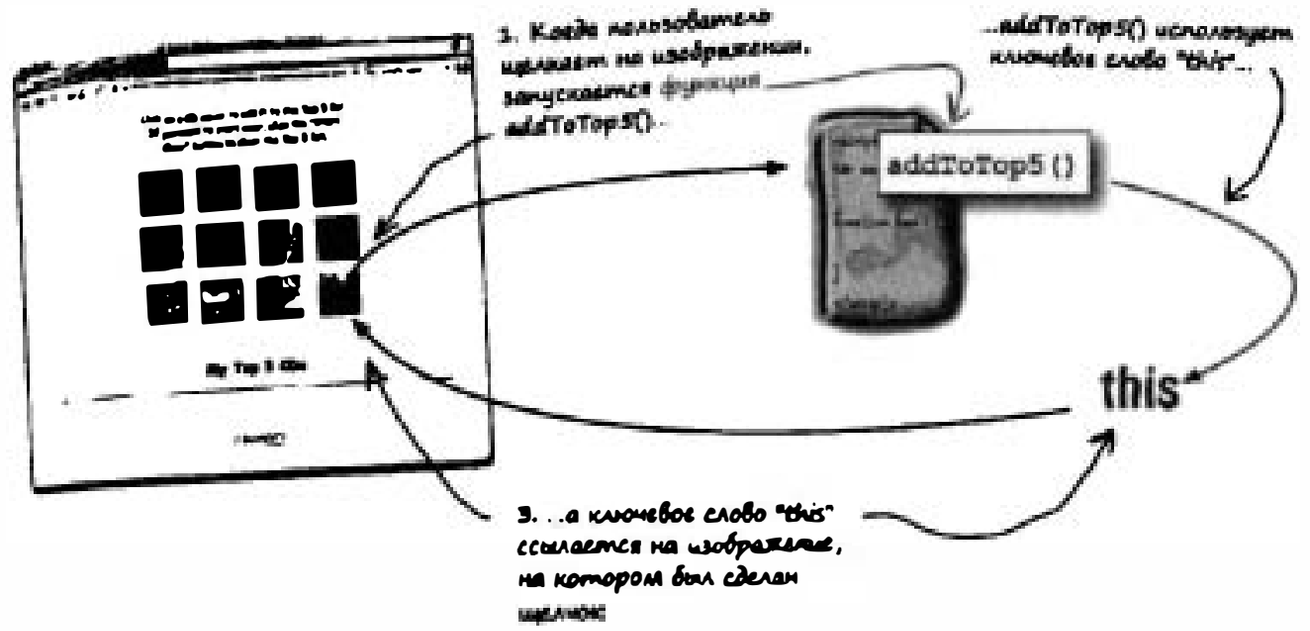
Как вы думаете, что произойдет с веб-страницей при перемещении элемента <img> из секции "cds" в секцию "top5"?

Обратите внимание на «this»

Прежде всего необходимо определить, на каком именно элементе <img> был сделан щелчок; эта информация позволит добавить нужную обложку в секцию "top5". Для определения части дерева DOM, вызвавшей функцию, в JavaScript существует специальное ключевое слово this.

Посмотрим, как работает this.





Поиск секции "top5"

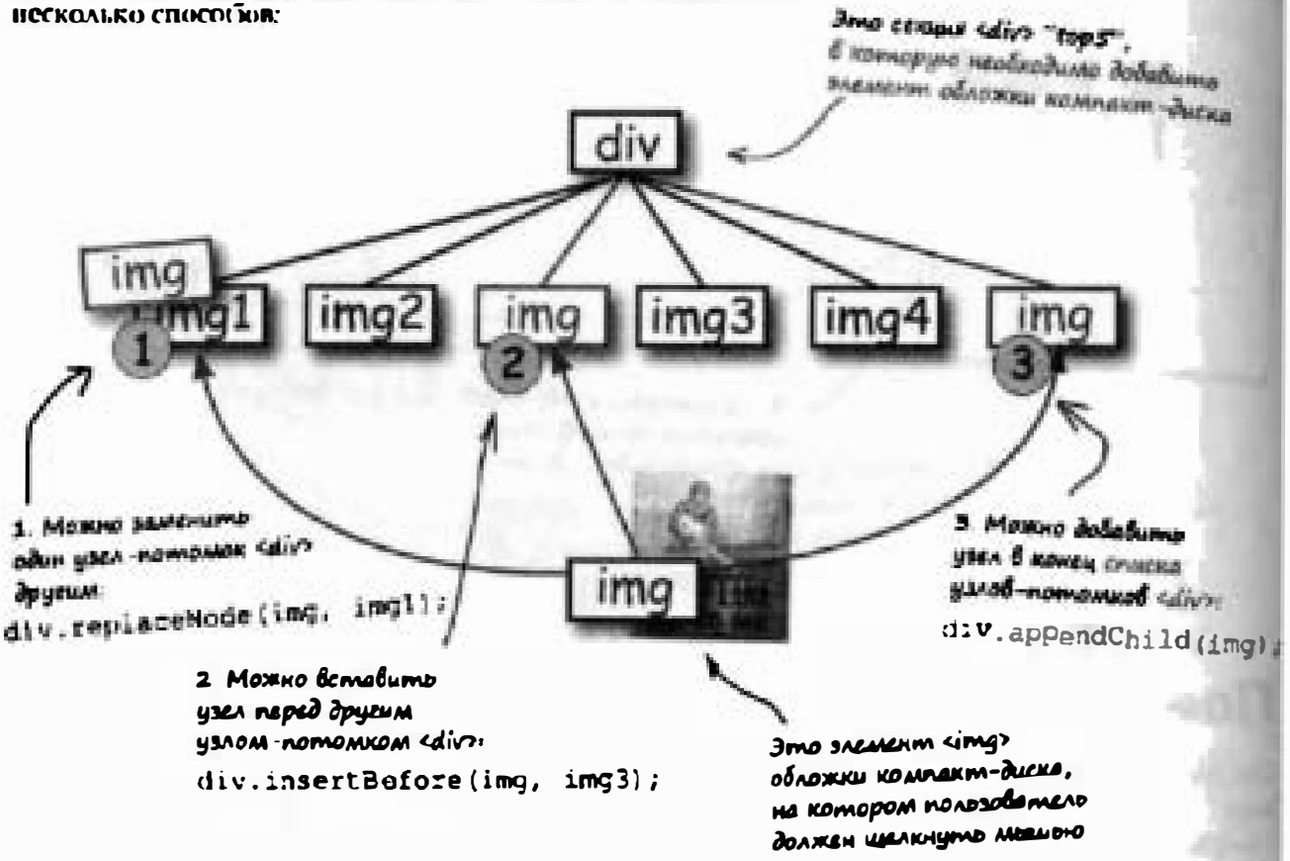
Следующим шагом должно стать получение элемента "top5" из дерева DOM браузера. Мы хотим найти элемент <img>, на котором был сделан щелчок, в секции "top5", и удалить обработчик onClick элемента <img>, чтобы функция **addToTop5()** не запускалась при повторных щелчках.

Полученные элементы с изображением обложки и элементом секции "top5", в который включается обложка

```
function addToTop5() {
  var imgElement = this;
  var top5Element = document.getElementById("top5");
}
```

Добавление потомков к элементу

После того как вы получили элемент<img> обложки компакт-диска и секцию "top5", вам необходимо добавить этот элемент в эту секцию. Для этого существует несколько способов:

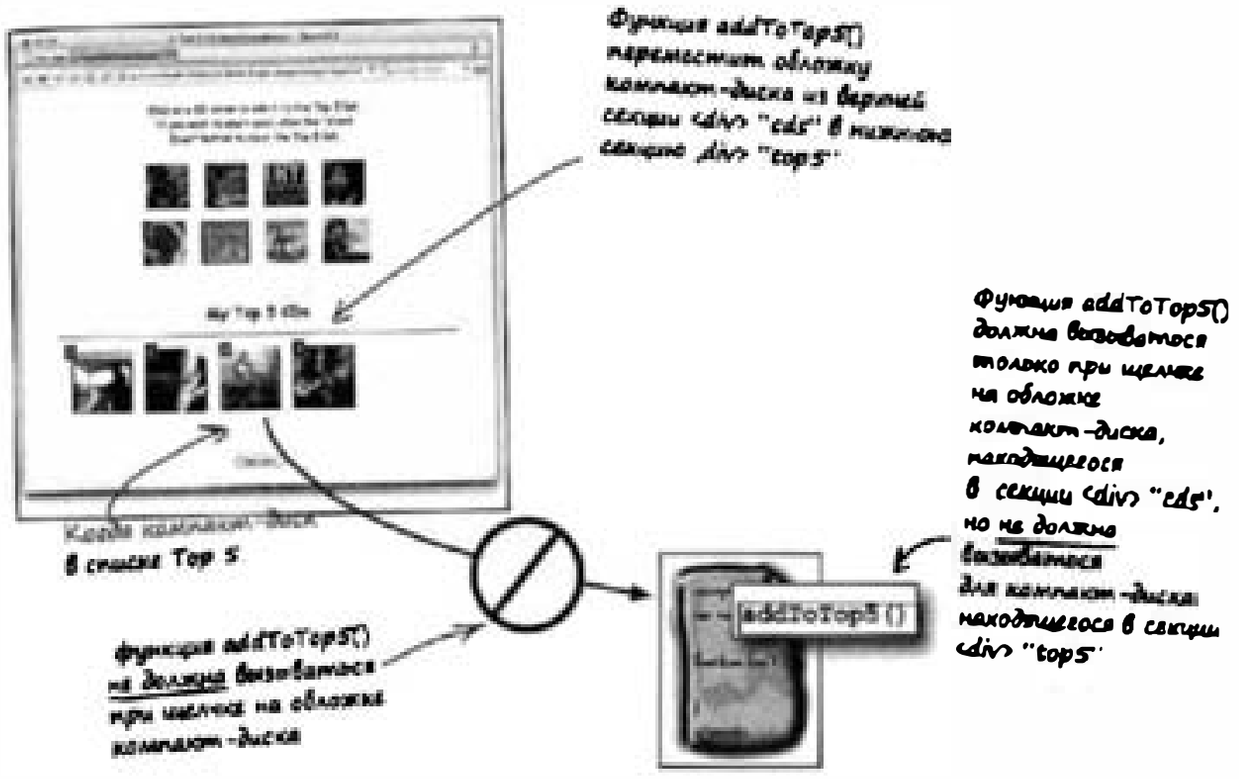


Как вы думаете, какой подход следует использовать для добавления элемента обложки компакт-диска в секцию "top5"?

После добавления компакт-диска в список Top 5 вы можете захотеть добавить новые компакт-диски в конец этого списка

Вернемся к обработчикам событий

Итак, вы готовы писать код **addToTop5()**... но есть еще один момент, на который следует обратить внимание. После добавления оболочки компакт-диска в секцию ** ** "top5" необходимо удалить из обработчика событий вызов функции **addToTop5()** при щелчке пользователя на обложке компакт-диска.



Помните, как мы добавляли обработчик событий?

```
cdImages[i].onclick = addToTop5;
```

Элемент <img> для обложки диска...

имя обработчика события

Выполняемая функция. Чтобы отменить назначенный обработчик события, достаточно задать свойству равенство null

Разложите по полкам

На последних нескольких страницах вы узнали много нового. Пора пустить в ход новые навыки DOM и JavaScript. Завершите функцию addToTop5(), подставив фрагменты кода на пустые места. Будьте внимательны — некоторые фрагменты кода остаются неиспользованными!

```
function addToTop5() {
  var imgElement = _____;
  var top5Element = document.getElementById(_____);

  top5Element._____;
  imgElement._____ = _____;
}
```

После того как обложка диска переместится в «верхнюю пятерку», обработчик события должен быть удален

Здесь в список "top5" добавляется элемент <img>; для этого используется один из методов, представленных на предыдущей странице

appendChild(imgElement)

that

onclick

this

insertBefore(imgElement)

addToTop5

"top5"

imgElement

top5Element

replaceNode(imgElement)

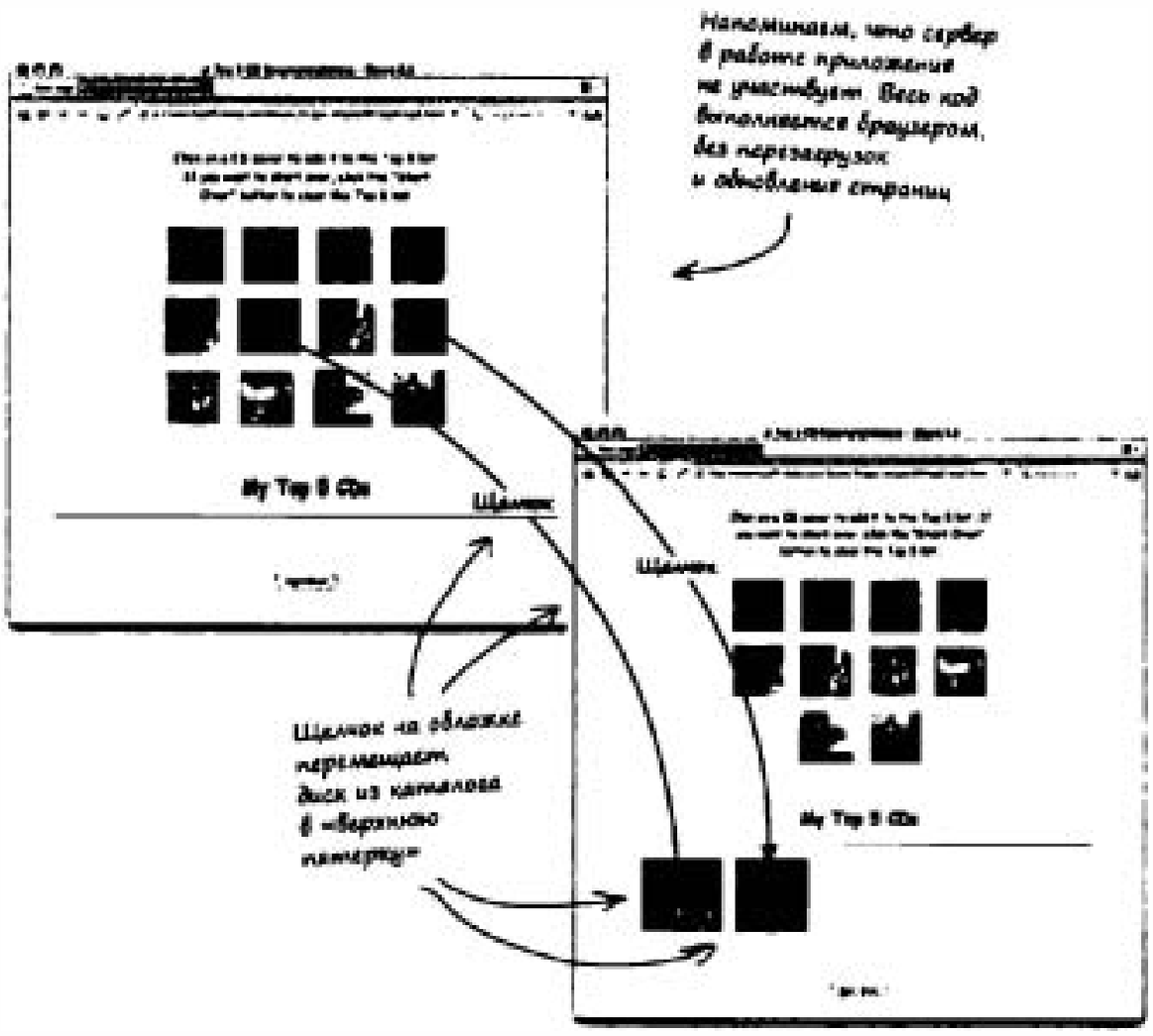
top5-listings

null

Помните: null означает «значение не определено» или «удалить текущее значение»

Тестируем addToTop5()

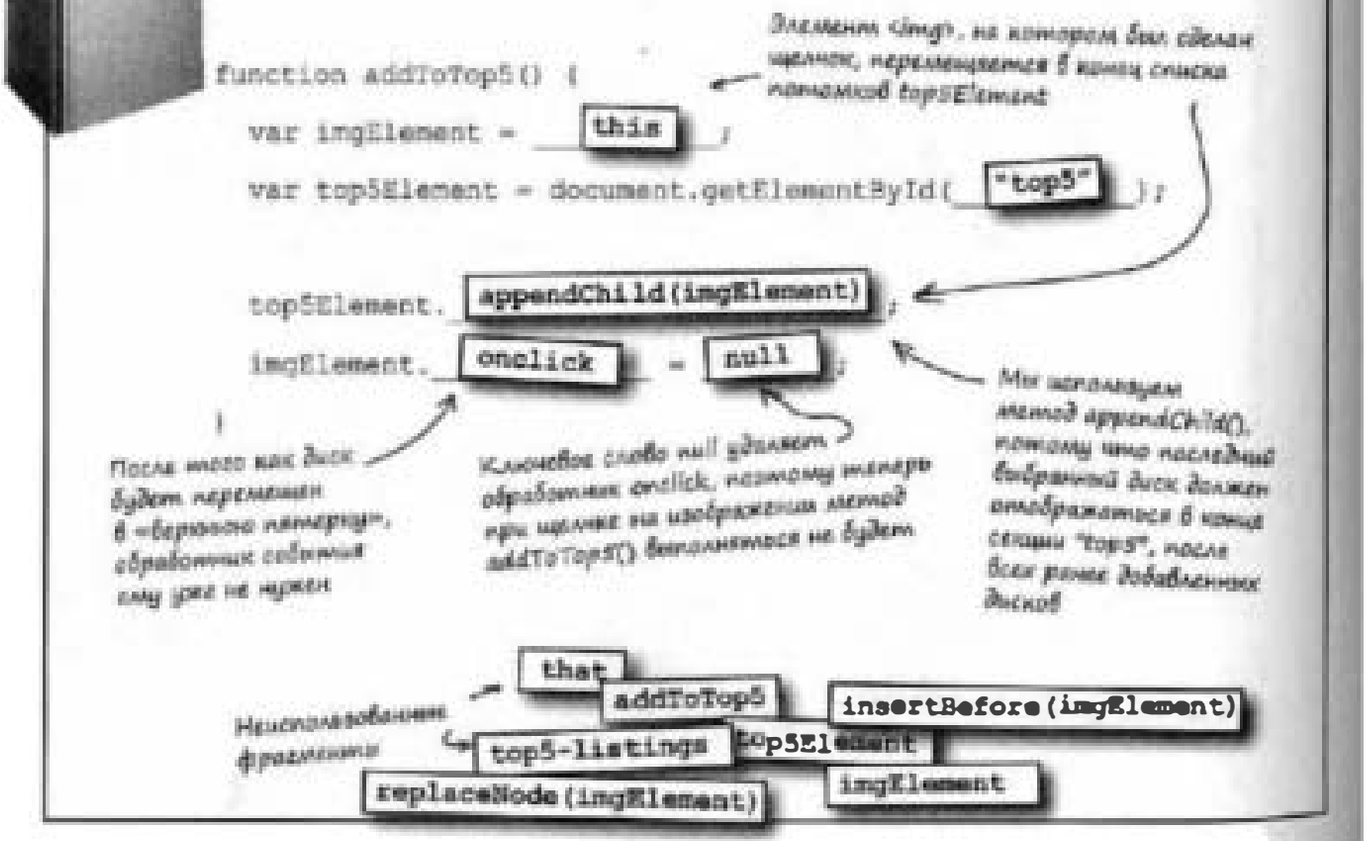
Выполните упражнение «Разложите по полкам» и сверьте свои ответы с нашими на с. 264. Внесите изменения в файл top5.js и загрузите top5.html в браузере. Что происходит, если щелкнуть на обложке диска?



Разложите по полкам

У вас возникли трудности с выполнением этого задания? Убедитесь в том, что фрагменты были расставлены в нужных местах, и обновите код JavaScript.

Решения

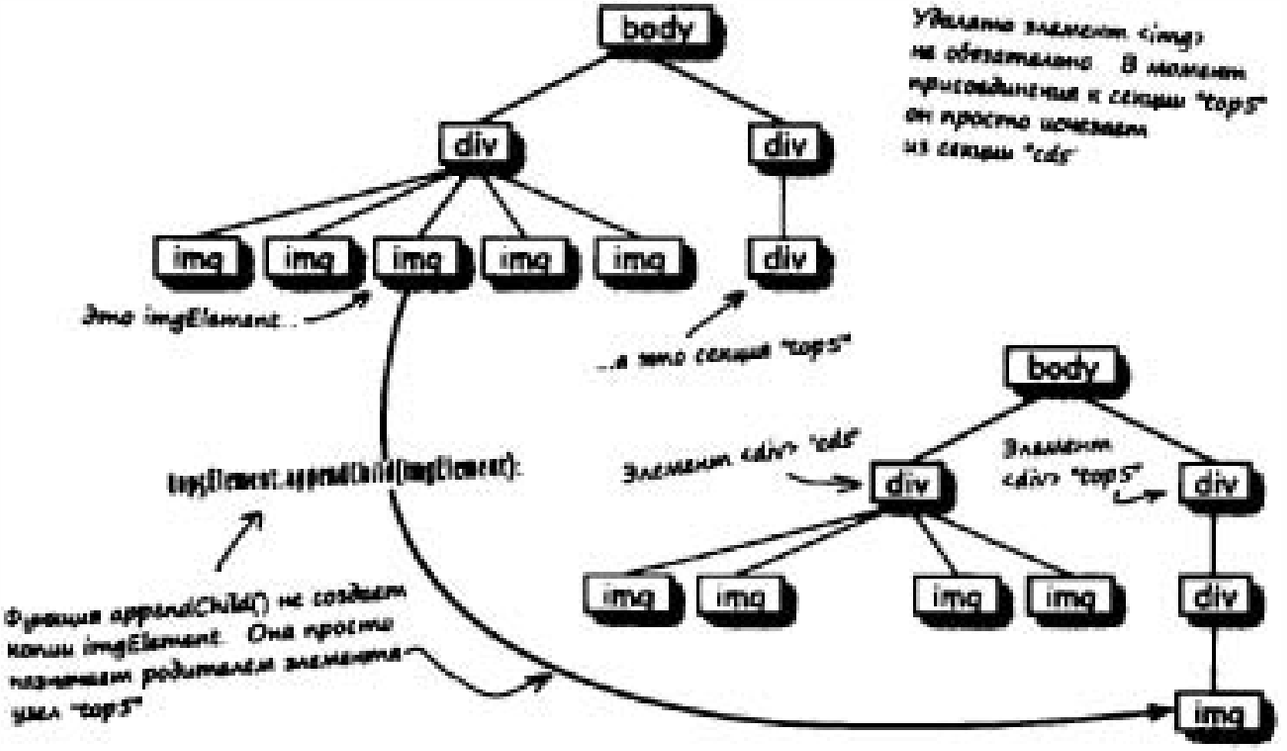


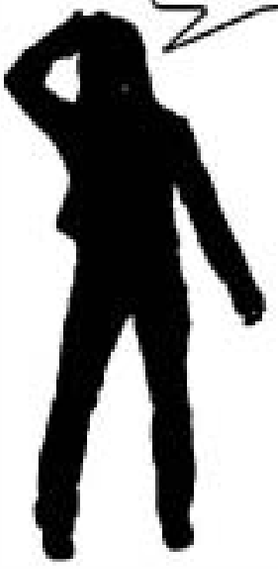
Эй, вы кое-что забыли... Где код удаления диска из верхней секции при перемещении его в «верховою пятерку»?

Элемент может иметь только одного родителя

Присмотримся повнимательнее к коду **addToTop5()**. Сначала мы получаем элемент **<img>** для обложки компакт-диска, а затем получаем ссылку на секцию "top5". Пока никаких сюрпризов, не так ли?

Затем мы берем элемент **<img>** и добавляем его в качестве потомка в секцию ** ** "top5". В этот момент родителем **<img>** уже не является элемент ** ** "cds"... им становится элемент ** ** "top5". Но элемент может иметь только одного родителя, поэтому он перемещается в дереве DOM:





Я думала, как организовать добавление рейтингов... но ведь мы не следим за тем, сколько дисков было включено в «пятерку». Пожалуй, начать нужно именно с этого. Достаточно подсчитать количество дочерних элементов с именем "img" в секции "top5", верно?

Если вас смущает цикл и проверка имени элемента, обратитесь к функции addOnClickHandlers() на с. 254

Точно!

А теперь, когда вы знаете, как в цикле перебрать потомков элемента и умеете определять имена элементов, это должно быть совсем легко.

На этом шаге мы включим новую переменную в функцию **addToTop5()** и присвоим ей количество дисков в «верхней пятерке».

```
function addToTop5() {
  var imgElement = this;
  var top5Element = document.getElementById("top5");
  var numCDs = 0;

  for (var i=0; i<top5Element.childNodes.length; i++) {
    if (top5Element.childNodes[i].nodeName.toLowerCase() == "img") {
      numCDs = numCDs + 1;
    }
  }

  /* Код перемещения диска в "верхнюю пятерку" */
}
```

Новая переменная. Начальное значение равно 0

Мы должны проверить каждого потомка в секции "top5"

В цикле перебираем потомков узла "top5" и подсчитываем только элементы <img>. Поскольку здесь также могут присутствовать текстовые узлы, содержащие один пробел, мы проверяем каждого потомка и убеждаемся в том, что это элемент <img>

Увеличиваем счетчик для каждой обложки

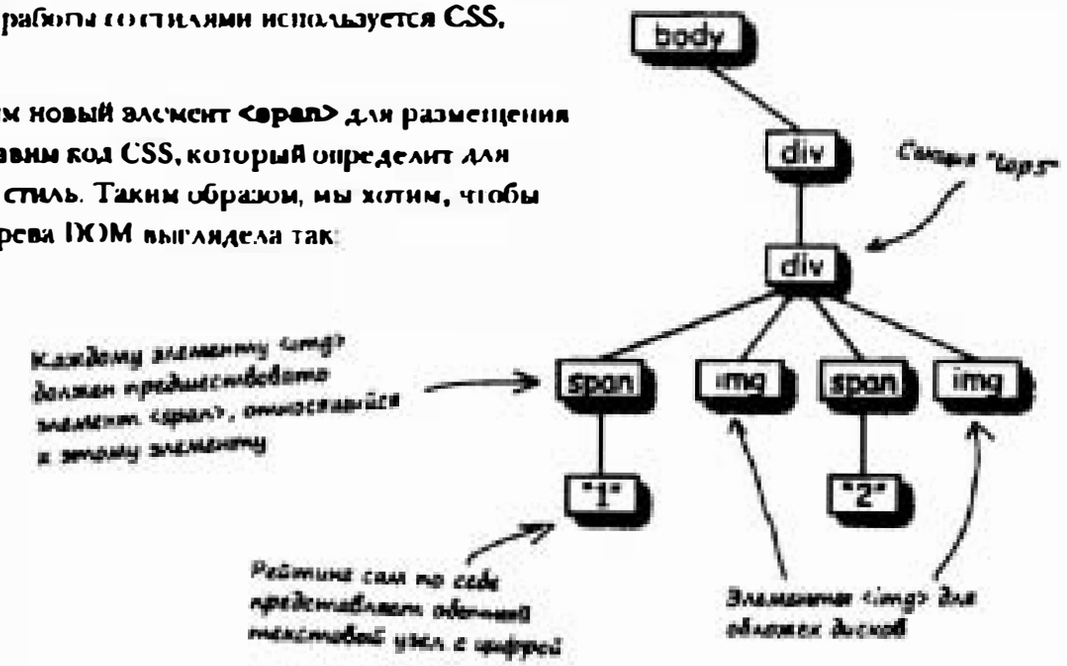
Добавление рейтинга

Зная, сколько дисков входит в «верхнюю пятерку», можно переходить к добавлению рейтингов к каждому добавляемому диску. Вот как должен выглядеть рейтинг:

Новые дополнения к дереву DOM

Для отображения рейтинга нам потребуется текстовый элемент. Что касается его значения, достаточно увеличить **currCds** на 1. Но форматирование не относится к задачам JavaScript... для работы со стилями используется CSS, не так ли?

Давайте создадим новый элемент **** для размещения рейтинга, и добавим код CSS, который определит для **** нужный стиль. Таким образом, мы хотим, чтобы секция "top5" дерева DOM выглядела так:



Что остается сделать?

Функция **addToTop5()** почти готова. Давайте разберемся, что именно необходимо сделать, и перейдем к программированию.

① Добавить новый класс CSS для рейтингов дисков

Далее приводится код CSS, который накладывает рейтинг на обложку диска и обеспечивает нужную цветовую гамму (белая цифра на черном фоне). Включите следующий код CSS в файл top5.css:

```
.rank {
  position:         absolute;
  text-align:       center;
  top:              20px;
  font-size:        small;
  background-color: black;
  color:            white;
  border:           thin solid
white;
  width:            20px;
  z-index:            99;
}
```

Прежде чем следовать дальше, включите этот код CSS в файл top5.css

② Создать новый элемент и задать для него класс CSS

На этом шаге задействован файл top5.html

Присвоим классу имя "rank" и настроим его так, чтобы рейтинги выводились поверх обложек, белыми цифрами на черном фоне

③ Создать новый текстовый узел и занести в него рейтинг диска

Все остальное делается с функцией JavaScript addToTop5()

Количество дисков уже известно, осталось увеличить его на 1 и задать полученное число как рейтинг текущего диска

④ Включить текстовый узел в , а — в

Простое подключение новых узлов к дереву DOM (обеспечивает вывод рейтингов дисков на веб-странице)

⑤ Проверить, что «верхняя пятерка» содержит не более 5 дисков

Простая проверка гарантирует, что в любой момент времени количество дисков в нижней секции не превышает 5

Просто сделайте это

Запустите свой текстовый редактор и завершите функцию addToTop5(). Ваша задача — добавить в addToTop5() код шагов 2–5. Как вы думаете, получится?

Посмотрим, удастся ли вам заполнить все пропуски. Мы добавили несколько замечаний, которые помогут вам в этом. Обязательно выполните упражнение перед тем, как переворачивать страницу.

```
function addToTop5() {
  var imgElement = this;
  var top5Element = document.getElementById("top5");
  var numCDs = 0;

  for (var i=0; i<top5Element.childNodes.length; i++) {
    if (top5Element.childNodes[i].nodeName.toLowerCase() == "img") {
      numCDs = numCDs + 1;
    }
  }
  if (_______ >= ____) {
    alert("You already have 5 CDs. Click \"Start Over\" to try again.");
    return;
  }

  top5Element.appendChild(imgElement);
  imgElement.onclick = null;

  var newSpanElement = _____.createElement(_____);
  _____.className = "_____";
  var newTextElement = document._____(numCDs + 1);
  newSpanElement._____(newTextElement);
  _____.insertBefore(_____, imgElement);
}
```

Убедитесь в том, что в «верхнюю пятерку» включается не более 5 дисков

Если с этим пунктом возникнут затруднения, вернитесь к началу главы 4

"className" используется для связывания класса CSS с элементом

Текст добавляется «в последнюю позицию» списка потомков (других потомков все равно нет)

Кто должен быть родителем вложенного ?

В первом аргументе insertBefore() передается вставляемый узел

Завершенная функция addToTop5()

Вот как выглядит готовая функция addToTop5(). Убедитесь в том, что ваши ответы совпадают с нашими, и включите весь новый код в top5.js.

```
function addToTop5() {
  var imgElement = this;
  var top5Element = document.getElementById("top5");
  var numCDs = 0;

  for (var i=0; i<top5Element.childNodes.length; i++) {
    if (top5Element.childNodes[i].nodeName.toLowerCase() == "img") {
      numCDs = numCDs + 1;
    }
  }

  if (numCDs >= 5) {
    alert("You already have 5 CDs. Click \"Start Over\" to try again.");
    return;
  }

  top5Element.appendChild(imgElement);
  imgElement.onclick = null;

  var newSpanElement = document.createElement("span");

  newSpanElement.className = "rank";

  var newTextElement = document.createTextNode(numCDs + 1);

  newSpanElement.appendChild(newTextElement);

  top5Element.insertBefore(newSpanElement, imgElement);
}
```

Сравните numCDs с 5, чтобы предотвратить включение лишних дисков

Задаем атрибут "className" элемента

Все методы create вызываются через объект документа

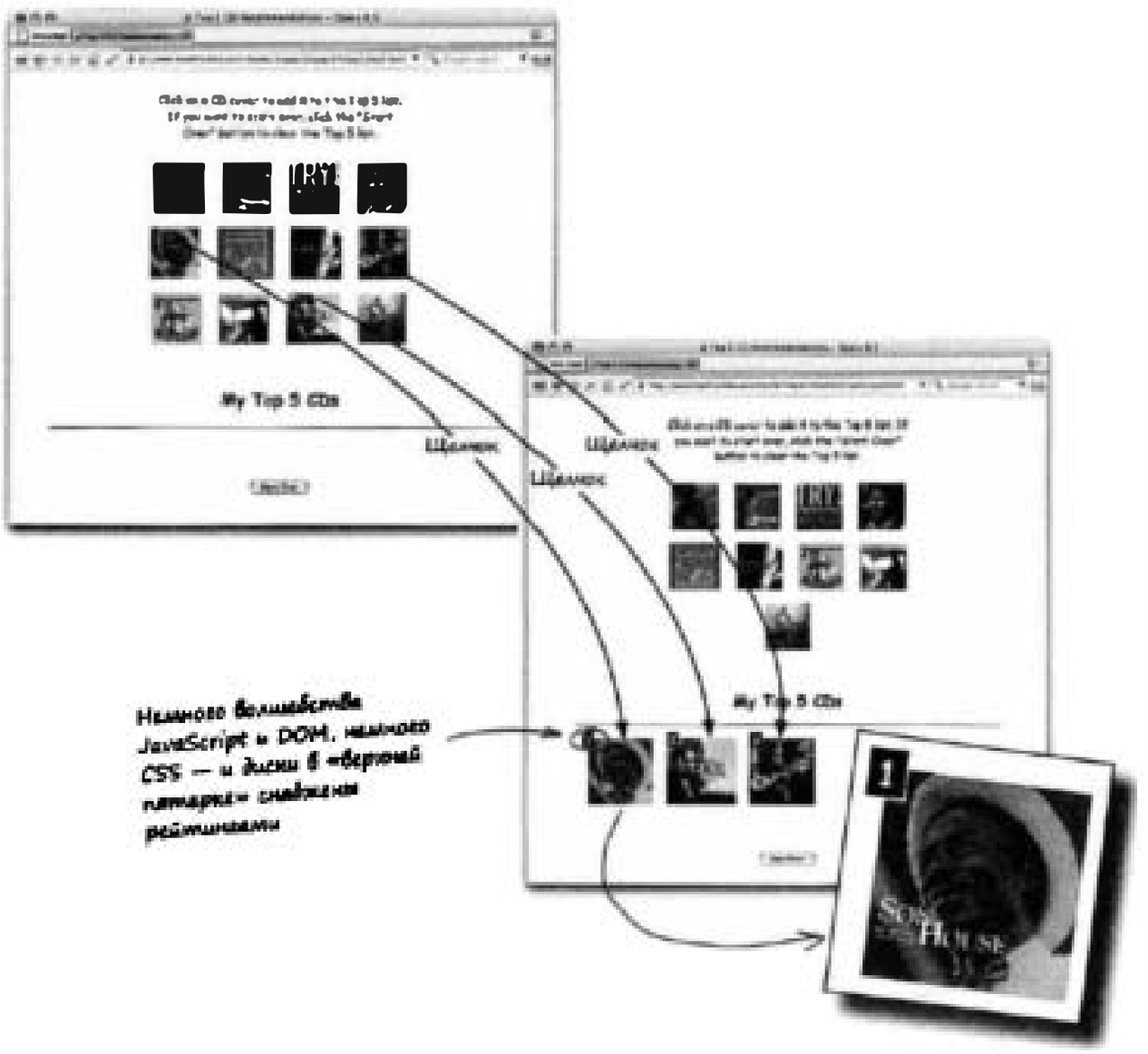
Текстовый узел должен содержать следующий рейтинг

Просто вставляем текстовый узел в конец списка потомков

Элемент размещается перед элементом <img> диска, в секции "top5"

Тестирование рейтингов (снова)

Завершив код функции **addToTop5()** в файле JavaScript, снова запустите браузер. Загрузите страницу Top 5 и щелкните на нескольких обложках дисков. Вы увидите, как в левом верхнем углу обложек появляются рейтинги, что происходит благодаря абсолютному позиционированию текстового элемента на уровне CSS. Рейтинги улучшают внешний вид страницы



Просто сделайте это

Пора в очередной раз подвергнуть испытанию ваше мастерство DOM. Функция **startOver()** должна решать несколько задач.

1. Перебор всех потомков узла "top5".
2. Возврат всех найденных обложек в верхнюю часть страницы.
3. Удаление со страницы всех остальных элементов (например,).

Посмотрим, удастся ли вам заполнить пропуски и дописать код JavaScript в файле top5.js.

```
function startOver() {
  var top5Element = document._____("top5");
  var cdsElement = document.getElementById("_____");
  while (_____.hasChildNodes()) {
    var firstChild = top5Element._____;
    if (firstChild._____.toLowerCase() == "img") {
      _____.appendChild(firstChild);
    } else {
      top5Element.removeChild(_____);
    }
  }
  _____;
}
```

Это маленькая хитрость. Помните, когда все обложки помещают в дисковой набор, вы после щелчка должны вызвать addToTop5(). Возможно, есть функция, которая делает это?

Осталось обновить файл top5.html и построить кнопку Start Over для вызова завершенной функции startOver():

```
<form><p>
 <input type="button" value="Start Over" [redacted] />
</p></form>
```

Не забывайте связывать кнопки HTML и элементы с кодом JavaScript

Последняя проверка

Введите код JavaScript функции **startOver()**, и не забудьте добавить обработчик **onClick** для кнопки **<input>** в файле HTML. Затем проведите последнюю проверку страницы Top 5. При щелчке на обложке компакт-диск перемещается в «верхнюю пятерку» (при этом на нем появляется симпатичная цифра рейтинга), а кнопка Start Over возвращает все диски на исходное место и позволяет начать отбор заново.

Блюзы? Электроритмы? Выбирайте любимые мелодии

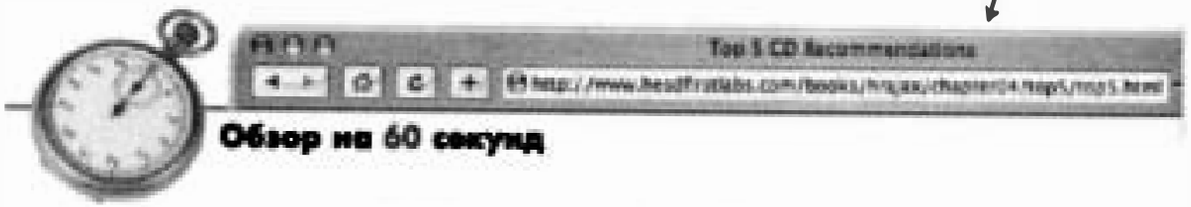
Великое испытание для главы 4

Написать веб-приложение, использующее модель DOM для создания динамического интерфейса пользователя, без применения кода Ajax.

Еще не забыли? Мы построили отличное веб-приложение без единой строки асинхронного кода JavaScript. Отличная работа!

Не думайте, что мы забросили асинхронное программирование... Мы вернемся к нему в главе 5.

Проверьте сами!



Обзор на 60 секунд

- Для представления кода HTML, CSS и JavaScript, образующего веб-страницу, браузер использует модель дерева объектов Document Object Model (сокращенно DOM)
- Для просмотра и модификации модели DOM используется код JavaScript. Изменения, вносимые в DOM, автоматически отражаются на веб-странице, отображаемой браузером
- Объект JavaScript с именем document используется для обращения к дереву DOM текущей веб-страницы.
- Работа с моделью DOM может осуществляться в любых веб-приложениях, не только в асинхронных.
- Дерево DOM состоит из узлов разных типов: узлов элементов, узлов атрибутов и текстовых узлов
- Узел элемента может иметь только одного родителя. Смена родителя или назначение элемента потомком другого узла приводит к перемещению элемента в дереве DOM
- С узлами DOM можно связывать стили CSS и обработчики событий из кода JavaScript.

Просто сделайте это — решение

Пора в очередной раз подвергнуть испытанию ваше мастерство DOM. Функция `startOver()` должна решать несколько задач:

1. Перебор всех потомков узла `<div>` "top5".
2. Возврат всех найденных обложек в верхнюю часть страницы.
3. Удаление со страницы всех остальных элементов (например, `<span>`)

```
function startOver() {

  var top5Element = document.getElementById("top5");

  var cdsElement = document.getElementById("cds");

  while (top5Element.hasChildNodes()) {

    var firstChild = top5Element.firstChild;

    if (firstChild.nodeName.toLowerCase() == "img") {

      cdsElement.appendChild(firstChild);

    } else {

      top5Element.removeChild(firstChild);

    }

  }

  addOnClickHandlers();

}
```

Начинаем с получения двух элементов `<div>`, с которыми мы будем работать

Работаем с потомками узла `<div>`, пока остаются хотя бы один потомок

Если узел является элементом `<img>`, вернуть его в верхнюю часть страницы

...а если это элемент `<span>`, пропуск или что-нибудь еще — просто удалить со страницы

This calls the `addOnClickHandlers()` function you wrote earlier, and adds event handlers back to all the CD cover `<img>` elements

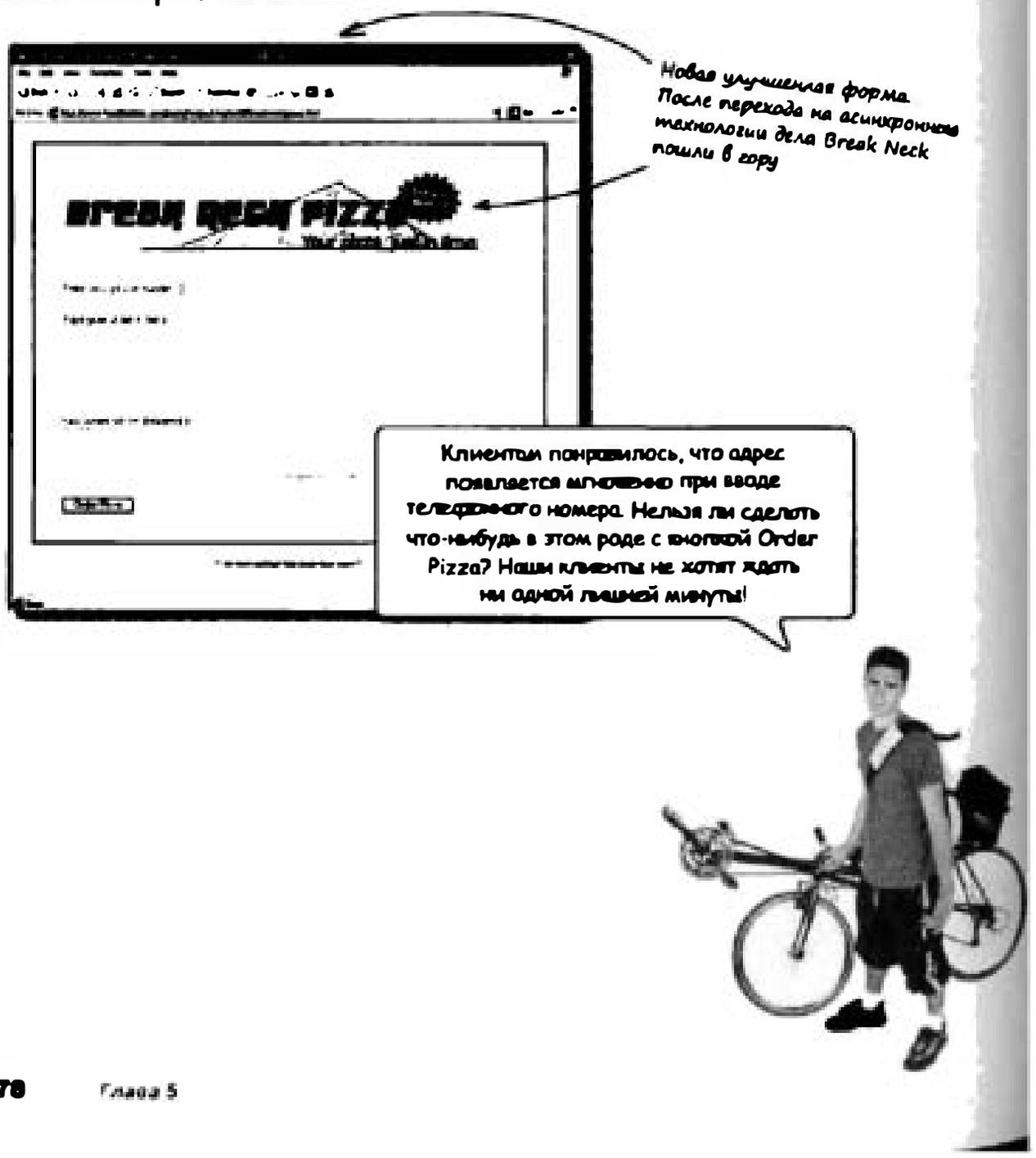
5 Запросы POST

Передача информации в POST

То, чего мы так долго ждали. Наконец-то мы забудем о веб(пай) и научимся передавать расширенные данные серверу. Для этого придется немного потрудиться, но к концу этой главы ваши асинхронные запросы к серверу уже не будут ограничиваться категорией «без данных». Так что пристегните ремни — нам предстоит открыть страну POST, страну типов контента и заголовков запросов.

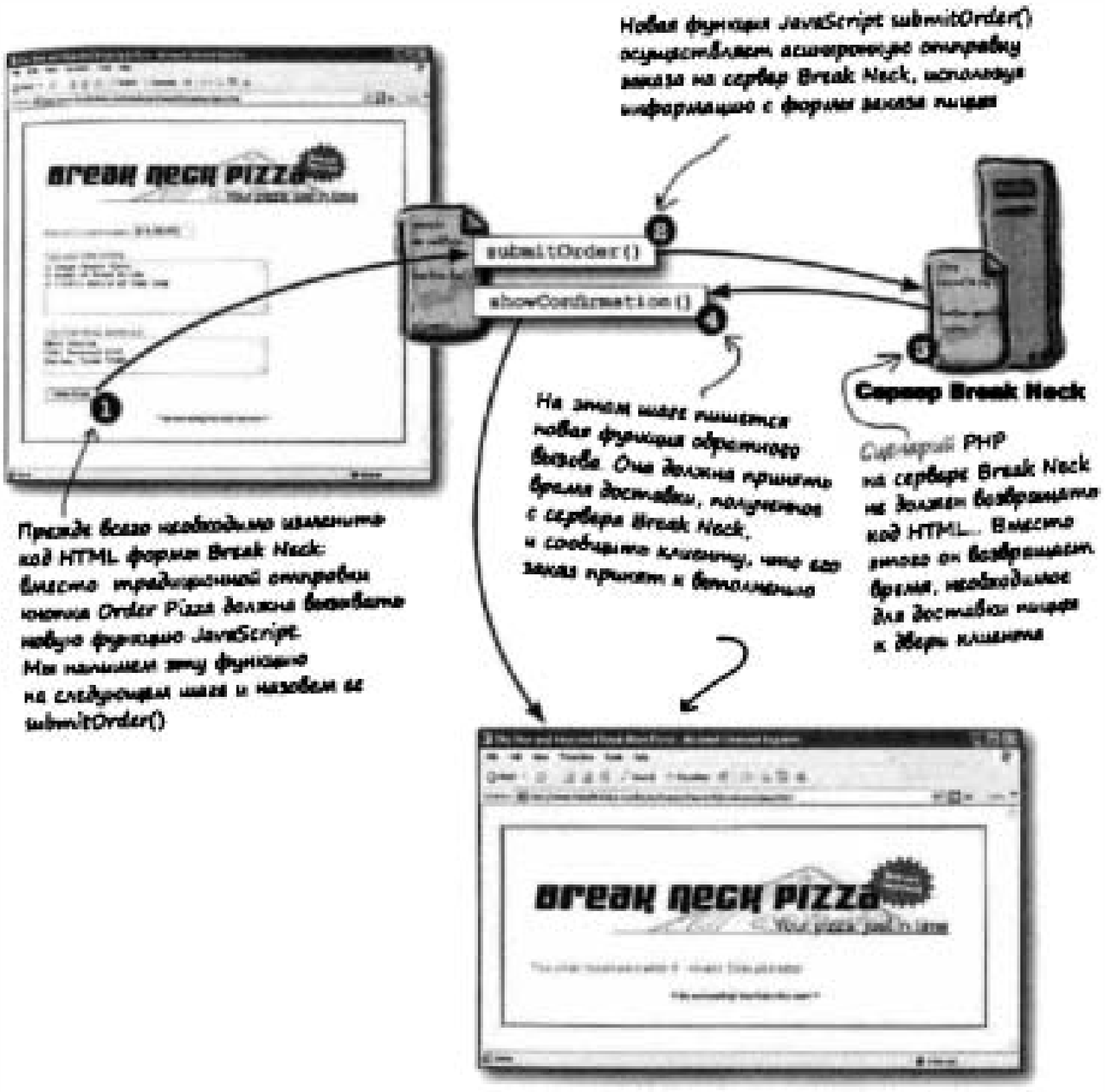
Знакомая задача

Помните всю работу, проделанную нами для Break Neck Pizza? Похоже, Ajax-версия формы для заказа пиццы пришлась всем по вкусу. Компания Break Neck хочет наделить ее несколькими новыми возможностями, и для этого она снова обратилась к вам. Давайте посмотрим, что они хотят.



Отправка формы в Ajax

Похоже, ничего особо сложного Break Neck не просит. Посмотрим, нельзя ли преобразовать отправку формы Break Neck в асинхронный запрос. Вот что для этого необходимо сделать:



1. Обновление кода HTML

Сначала необходимо модифицировать код HTML формы Break Neck, чтобы щелчок на кнопке Order Pizza не приводил к отправке формы. Давайте проанализируем код Break Neck и внесем несколько изменений:

```
<body>
 <div id="main-page">
  <p><img src="breakneck-logo_new.gif" alt="Break Neck Pizza" /></p>
  <form id="order-form" [redacted]>
   <p>Enter your phone number:
    <input type="text" size="14"
           name="phone" id="phone" onChange="getCustomerInfo()" />
   </p>
   <p>Type your order in here: <br />
   <textarea name="order" rows="6" cols="50" id="order">
     </textarea></p>
   <p>Your order will be delivered to: <br />
   <textarea name="address" rows="4" cols="50" id="address">
     </textarea></p>
   [redacted]
   <p><input [redacted] value="Order Pizza"
             [redacted] /></p>
  </div>
  <p class="tagline">** No more waiting! Now faster than ever! **</p>
 </form>
</body>
```

Элементу <form> не нужны атрибуты "action" или "method"

Старая версия кнопки "submit"

Вместо отправки формы кнопка "Order Pizza" запускает функцию JavaScript, которую мы напишем на следующем шаге

Тип кнопки меняется с "submit" на "button"

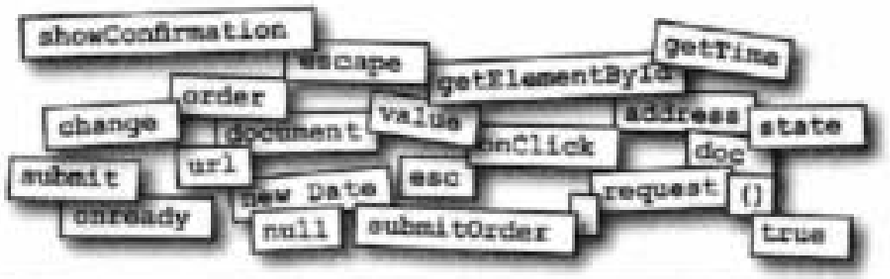
2. Отправка заказа на сервер

Разложите по полкам

На следующем месте в нашем списке задач стоит написание функции submitOrder(). Эта функция должна получать телефон, адрес и описание заказа с формы, и отправлять их сценарию placeOrder.php на сервере Break Neck. Функция обратного вызова, которую мы напишем, будет называться showConfirmation(); не забудьте указать браузеру, чтобы он запускал эту функцию при получении ответа от сервера.

Чтобы головоломка стала более интересной, каждый фрагмент разрешается использовать сколько угодно раз... или не использовать вовсе. Более того, в некоторые пропуски необходимо подставить более одного фрагмента!

```
function submitOrder() {
  var phone = _____("phone").value;
  var _____ = _____.getElementById("_____").value;
  var order = _____.getElementById("order").value;
  var _____ = "placeOrder.php?phone=" + _____(phone) +
                                "&address=" + _____(address) +
                                "&order=" + _____(order);
  url = _____ + "&dummy=" + _____;
  _____.open("GET", _____, _____);
  request._____ = _____;
  request.send(_____);
}
```



* Подсказка Один клиент с одним телефоном и адресом может разместить одинаковые заказы несколько раз

Разложите по полкам

Решения

На следующем месте в нашем списке задач стоит написание функции submitOrder(). Далее показано, где следует разместить фрагменты в этой функции. Будьте внимательны — следите за тем, чтобы в функциях обратного вызова не появлялись лишние крутые скобки, а компоненты составных имен «document», «getElementById» и т. д. разделялись точками.

```
function submitOrder() {
  var phone = document.getElementById("phone").value;
  var address = document.getElementById("address").value;
  var order = document.getElementById("order").value;
  var url = "placeOrder.php?phone=" + escape(phone) +
                          "&address=" + escape(address) +
                          "&order=" + escape(order);
  url = url + "&dummy=" + new Date().getTime();
  request.open("GET", url, true);
  request.onreadystatechange = showConfirmation;
  request.send(null);
}
```

Поняли, зачем это делается? Мы предотвращаем кэширование запросов в таких браузерах, как Opera и IE (на случай, если клиент захочет разместить одинаковые заказы несколько раз)

Последите за тем, чтобы вы имена функции обратного вызова не было крутых скобок

Просто сделайте это

В папке chapter05/breakneck/ в архиве примеров вы найдете последнюю версию приложения Break Neck вместе с кодом HTML, CSS и PHP. Откройте файл pizza.html и измените форму так, чтобы при щелчке на кнопке запускалась функция **submitOrder()** (вместо прямой отправки сценария placeOrder.php на сервер Break Neck).

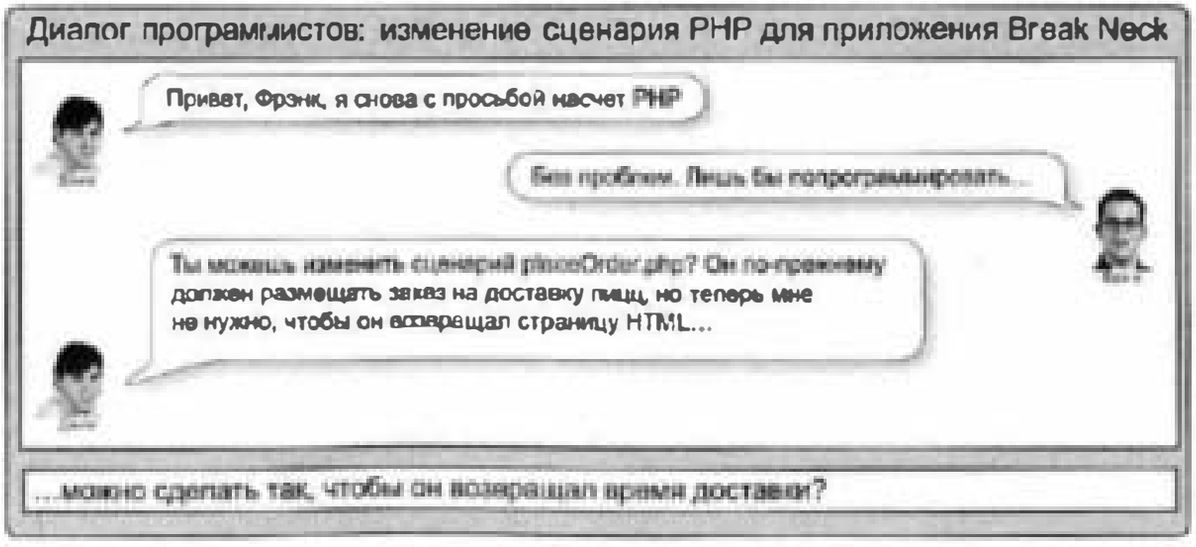
Раз уж мы занялись усовершенствованием Break Neck, вынесите весь код JavaScript за пределы pizza.html. Объект запроса можно создать в файле ajax.js, созданном нами в главе 3. Затем создайте новый файл JavaScript с именем pizza.js. Переместите функции Break Neck — **getCustomerInfo()**, **updatePage()** и **submitOrder()** — в этот файл. Не забудьте включить в код HTML элементы **<script>** со ссылками на эти файлы!

Проследите за тем, чтобы использовалась версия ajax.js с одним объектом запроса... В этой главе второй объект запроса не нужен

Выполнив все сказанное, перейдите на с. 314 в самом конце главы и убедитесь в том, что ваши файлы не отличаются от наших.

3. Обновление placeOrder.php

Работа продвигается довольно быстро! Пора попросить вашего старого знакомого Фрэнка обновить сценарий PHP. Теперь мы не хотим, чтобы он возвращал код HTML.



PHP ...на первый взгляд

Обновление сценария placeOrder.php прошло очень быстро. Новая версия сценария принимает заказ, но не возвращает кода HTML. Вместо этого она выдает примерное время, за которое пицца будет доставлена к двери клиента.

```
<?php

include("order.php");
include("delivery.php");

// Error checking
$order = $_REQUEST['order'];
$address = $_REQUEST['address'];
if (strlen($order) <= 0) {
  header("Status: No order was received.", true, 400);
  echo " ";
  exit;
}
if (strlen($address) <= 0) {
  header("Status: No address was received.", true, 400);
  echo " ";
  exit;
}

// Place the order
$pizzaOrder = new PizzaOrder($order, $address);
$pizzaOrder->cookOrder();
$pizzaOrder->prepOrder();

// Deliver the order
$delivery = new Delivery($pizzaOrder);
$delivery->deliver();
$deliveryTime = $delivery->getDeliveryEstimate();

echo $deliveryTime;
?>
```

Фрэнк использует пару вспомогательных файлов PHP для решения специфических задач обработки и доставки заказов. Мы не будем рассматривать эти файлы, но они включены в каталог примеров данной книги

Получение данных запроса

Убеждаемся в том, что полученный заказ не является пустой строкой

Некоторые браузеры (например, Safari) возвращают «неопределенный» статус при полном отсутствии ответа от сервера. Отправка пробела гарантирует возврат привычного кода статуса в таких браузерах

Проверяем наличие введенного адреса

Возвращает браузеру заголовок с сообщением об ошибке

...и код статуса HTTP, указывающий на возникновение ошибки

Заказ создан, выполнен и подготовлен к доставке

Заказ передается курьеру, который доставляет его клиенту

В завершение сценарий определяет, сколько времени займет доставка пиццы, и возвращает эту информацию браузеру

Когда возникают проблемы

При работе клиента с веб-приложением Break Neck возможны происходить разного рода сбои. Ниже перечислены некоторые проблемы, возникающие при работе клиента с формой заказа пиццы. Напишите, что, по вашему мнению, происходит при возникновении каждой ошибки.

Уолтер ошибся при вводе телефона на форме Break Neck, и теперь вводит свой адрес и описание заказа вручную.

Мэй-Ли вводит номер телефона и описание заказа, но нечаянно изменяет номер улицы в поле адреса.

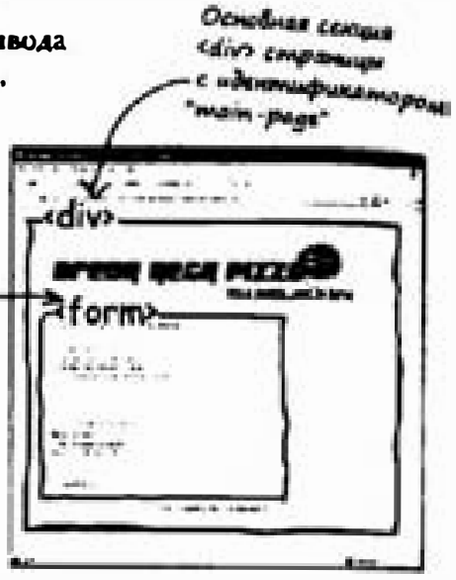
Сьюзен вводит свой телефон, но случайно щелкает на кнопке Order Pizza, не успев ввести описание заказа.

Ответы на с. 291.

4. Написание функции обратного вызова

Итак, сценарий PHP теперь возвращает только оцениваемое время доставки. Давайте напишем функцию JavaScript для вывода этой информации. Для начала необходимо точно определить, что должна делать новая функция.

Что у нас есть: Страница HTML с главным элементом , и элементом <form> внутри этого элемента для приема заказов



Что мы хотим: Страницу HTML с главным элементом и подтверждением заказа внутри этого элемента.

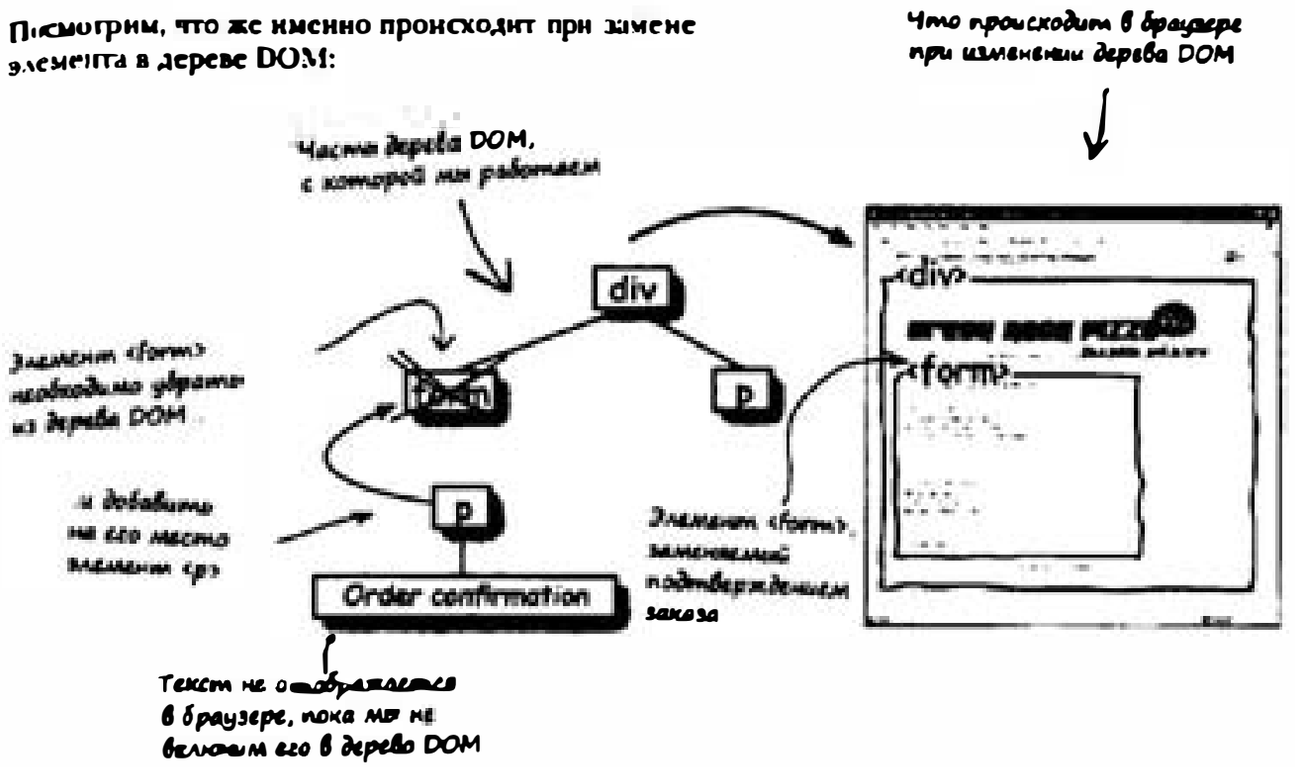
Что надо сделать: Заменить элемент <form> в главном элементе подтверждением заказа.

1. Получить ссылку на элемент по атрибуту "id".
2. Получить ссылку на элемент <form> по атрибуту "id".
3. Создать новый элемент <p> и добавить в него текст подтверждения заказа.
4. Заменить элемент <form> новым элементом <p> (и его текстовым узлом).

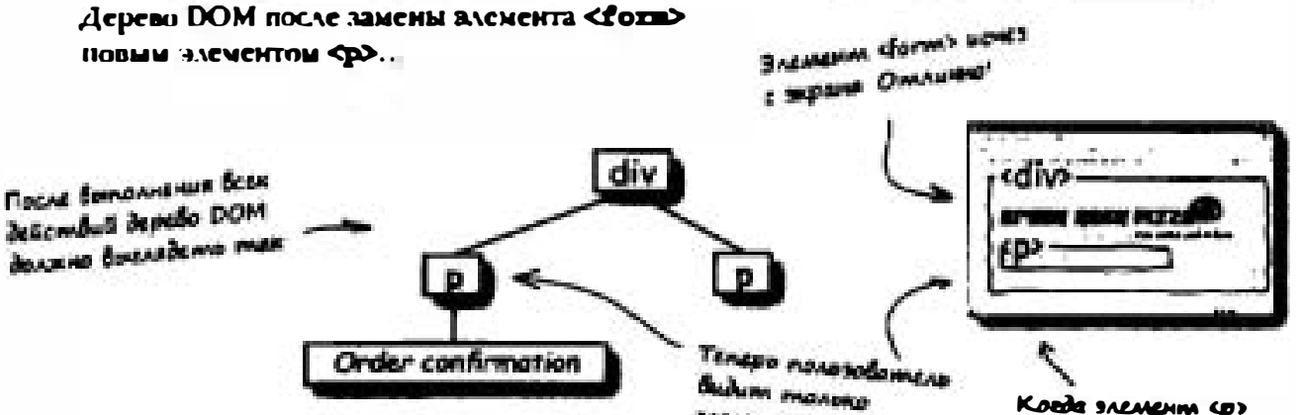
Рассмотрим этот шаг чуть подробнее

Информация, отображаемая на экране, определяется деревом DOM

Посмотрим, что же именно происходит при замене элемента в дереве DOM:



Дерево DOM после замены элемента <form> новым элементом <p>..



Все это происходит без перезагрузки страниц.. и без ожидания!

Простосделайтеэто

Пора собрать воедино все, что вы узнали в нескольких последних главах, и применить свои знания на практике. Далее приведена функция обратного вызова **showConfirmation()** для новой, усовершенствованной формы заказа пиццы. Ваша задача — заполнить пропуски и завершить код функции. Чтобы функция успешно работала, вспомните, что вы узнали об асинхронных запросах, формах HTML, DOM и обработке ошибок.

```
function showConfirmation() {
  if (request._____ == _____) {
    if (request._____ == _____) {
      var response = request._____;
      // Locate form on page
      var mainDiv = document.getElementById("_____");
      var orderForm = document.getElementById("_____");

      // Create some confirmation text
      pElement = document._____("p");
      textNode = document._____(
        "Your order should arrive within " +
        _____ +
        " minutes. Enjoy your pizza!");
      pElement._____(textNode);

      // Replace the form with the confirmation
      mainDiv.replaceChild(_____, _____);
    } else {
      alert("Error! Request status is " + request._____);
    }
  }
}
```

Функция выполняется после того, как сервер зарегистрирует заказ и вернет время доставки

*ПОДСКАЗКА: функция replaceChild() получает два аргумента. В первом аргументе передается новый узел, а во втором — заменяемый узел

Когда функция, по вашему мнению, будет готова, сравните свой ответ с нашим на с. 240, и включите код в файл pizza.js. После этого можно переходить к тестированию

Пробный запуск Break Neck

Убедитесь в том, что вы сравнили свой опыт к предыдущему упражнению с нашим, и внесли изменения в файл JavaScript pizza.js. Загрузите файл pizza.html в браузере и посмотрите, как выглядит страница.

Не торопись, Алекс...

Просто сделайте это — решение

Далее приводится полный код функции обратного вызова **showConfirmation()**: убедитесь в том, что ваши ответы совпадают с нашими... и что вы понимаете весь код.

```
function showConfirmation() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      var response = request.responseText;

      // Locate form on page
      var mainDiv = document.getElementById("main-page");
      var orderForm = document.getElementById("order-form");

      // Create some confirmation text
      pElement = document.createElement("p");
      textNode = document.createTextNode(
        "Your order should arrive within " +
        response +
        " minutes. Enjoy your pizza!");
      pElement.appendChild(textNode);

      // Replace the form with the confirmation
      mainDiv.replaceChild(pElement, orderForm);
    } else {
      alert("Error! Request status is " + request.status);
    }
  }
}
```

Перед проверкой статуса запроса обязательно проверяется состояние готовности

Помните: функция выполняется после того, как сервер зарегистрирует заказ. В ответе от сервера содержится примерное время доставки заказа

Свойство responseText запроса сообщает примерное время доставки

Получаем ссылку на узел элемента <form>

Затем средствами DOM в элемент <p> создается новый текст

Не забудьте включить текстовый узел в элемент <p>

The first argument is the new <p> element, and the second argument is the <form> to replace.

Если сценарий PHP вернул код ошибки, эта команда выведет его для сведения пользователя

Когда возникают проблемы — решение

Итак, что же происходит в каждом из перечисленных случаев? Сравните свои ответы с нашими.

Собственно, это нельзя считать ошибкой.

Уолтер ошибся при вводе телефона на форме Break Neck, и теперь вводит свой адрес и описание заказа вручную.

Если Уолтер ввел свой адрес верно, проблемы будет. Он получит свой заказ.

Мы ничего не можем сделать, чтобы предотвратить эту проблему

Мэй-Ли вводит номер телефона и описание заказа, но нечаянно изменяет номер улицы в поле адреса

Алекс доставит пиццу по неверному адресу! Здесь ничего не выделать..

Сьюзен вводит свой телефон, но случайно щелкает на кнопке Order Pizza, не успев ввести описание заказа.

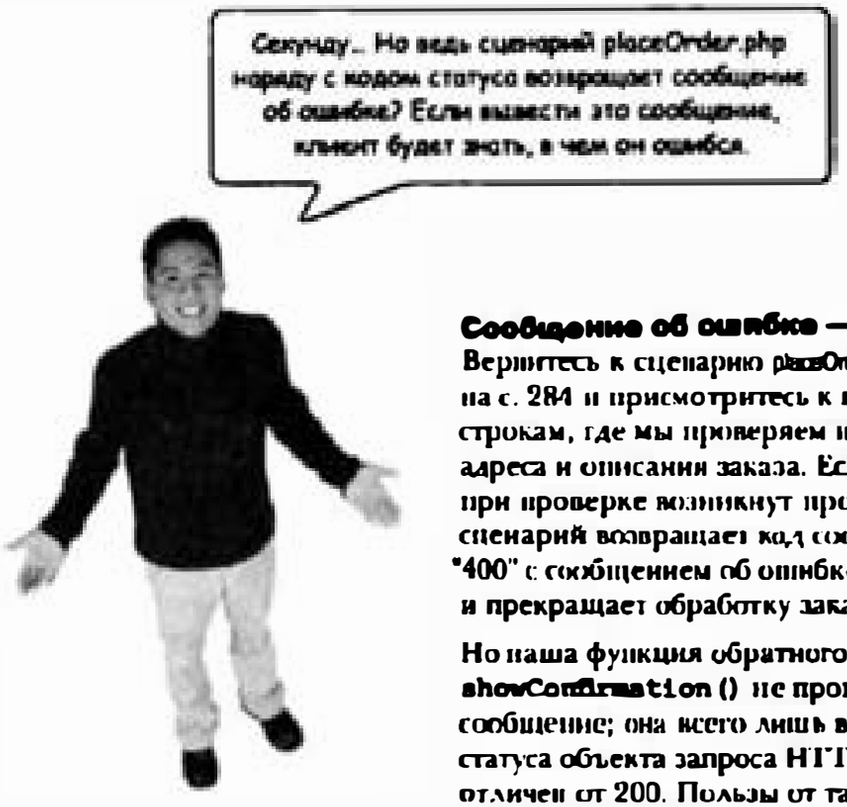
На форме заказа появится окно сообщения с кодом ошибки, возвращенным сценарием placeOrder.php (в данном случае 400).

В этом случае код статуса отличен от 200, поэтому выполняется блок 'else' функции showConfirmation()–

...и пользователь получит от вас форму бесполезное сообщение об ошибке. Кто знает, что означает код 400?

ШЕВЕЛИМ МОЗГАМИ

Присмотритесь к последней ошибке, где сценарий placeOrder.php возвращает браузеру ошибочный код статуса. Нельзя ли что-нибудь сделать для того, чтобы выводимое сообщение было более содержательным?



Сообщение об ошибке — хорошая штука

Вернитесь к сценарию placeOrder.php на с. 284 и присмотритесь к верхним строкам, где мы проверяем наличие адреса и описания заказа. Если при проверке возникнут проблемы, сценарий возвращает код состояния "400" с сообщением об ошибке и прекращает обработку заказа.

Но наша функция обратного вызова **showConfirmation()** не проверяет сообщение; она всего лишь выводит код статуса объекта запроса HTTP, если он отличен от 200. Пользы от такого сообщения маловато...

Код PHP для создания нового заголовка ответа:

```
if (strlen($order) <= 0) {
  header("Status: No order was received.", true, 400);
  exit;
}
if (strlen($address) <= 0) {
  header("Status: No address was received.", true, 400);
  exit;
}
```

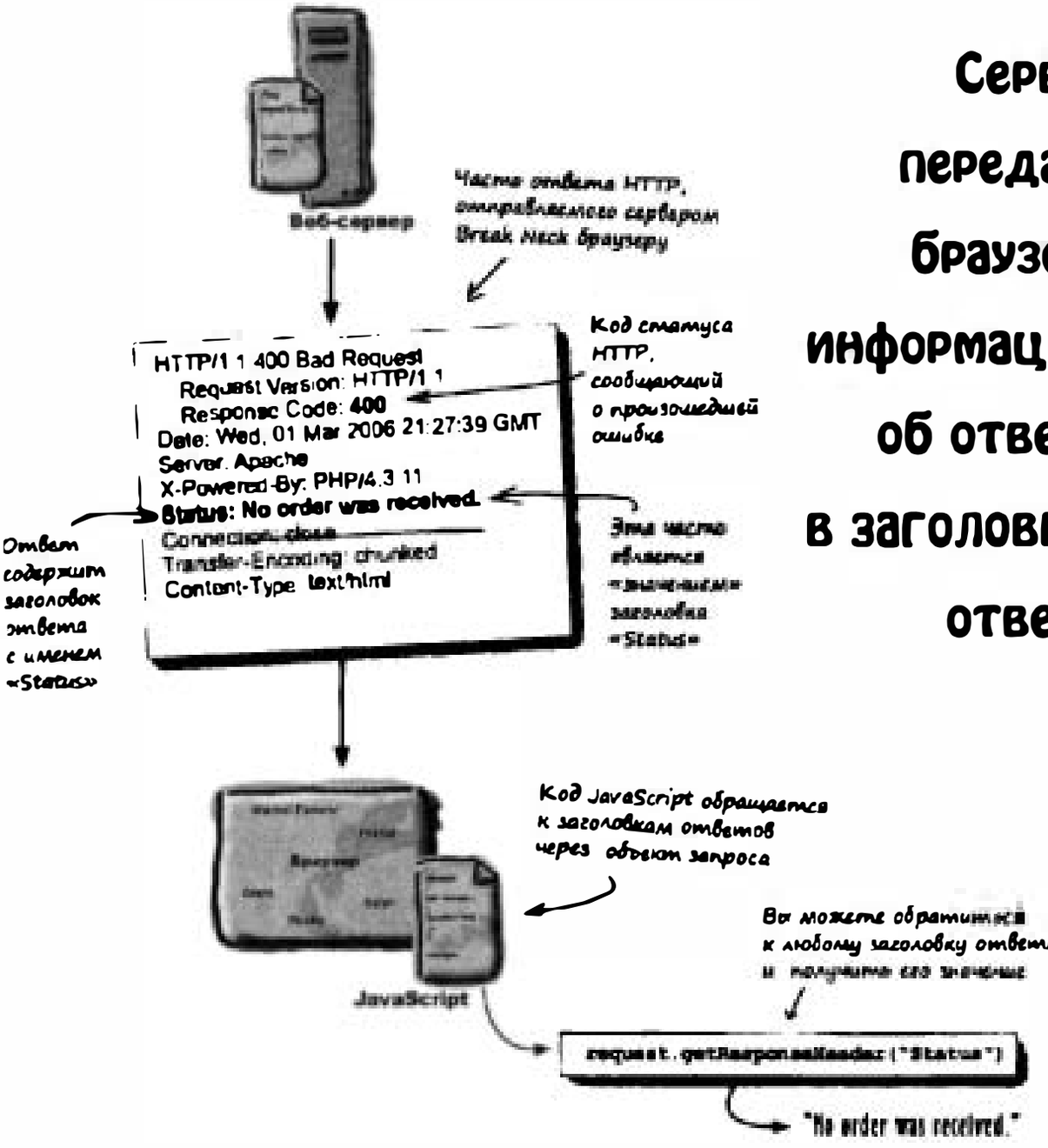
Строка "Status" становится именем заголовка ответа, возвращаемого браузеру

Весь текст после «Status:» становится частью сообщения, возвращаемого в качестве значения заголовка ответа

400 — код статуса, а "true" означает, что заголовок заменяет все существующие заголовки ответов того же типа (в данном случае «Status»)

Сервер возвращает информацию

Каждый раз, когда сервер возвращает ответ на ваш запрос, он может передать в своем ответе расширенную информацию в заголовках ответов



Сервер передает браузеру информацию об ответе в заголовках ответа.

Обработка ошибок в приложении Break Neck

Зная, как получить заголовок ответа, мы можем усовершенствовать код JavaScript приложения Break Neck и выдать пользователю чуть более подробную информацию о возникающих ошибках.

Внесите несколько простых изменений в разметку:

```
function showConfirmation() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      var response = request.responseText;
      var mainDiv = document.getElementById("main-page");
      var orderForm = document.getElementById("order-form");

      pElement = document.createElement("p");
      textNode = document.createTextNode(
        "Your order should arrive within " +
        response +
        " minutes. Enjoy your pizza!");
      pElement.appendChild(textNode);

      mainDiv.replaceChild(pElement, orderForm);
    } else {
      var message = request.getResponseHeader("Status");
      if ((message.length == null) || (message.length <= 0)) {
        alert("Error! Request status is " + request.status);
      } else {
        alert(message);
      }
    }
  }
}
```

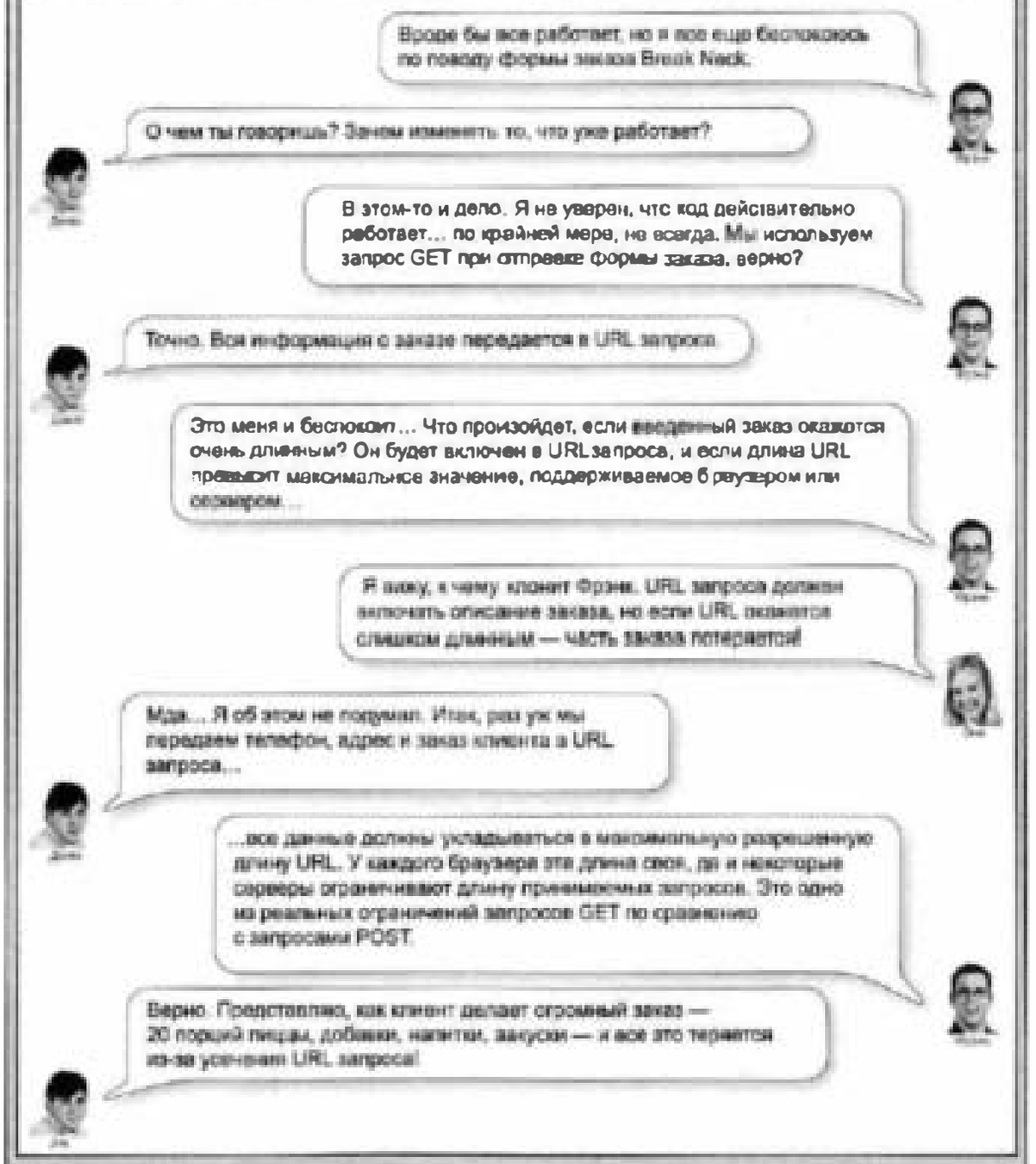
Если сценарий placeOrder.php сообщает об ошибке, код статуса равен 400, и выполняется код секции «else»

Команда получает значение заголовка "Status" (если он существует)

Если заголовок "Status" не существует, мы просто выводим код ошибки, как это делалось в предыдущей версии приложения Break Neck

Если сервер или сценарий вернули заголовок ответа "Status", показать его содержимое пользователю

Диалог программистов: остается решить еще одну проблему…



Выходит, вместо запроса GET нам следует использовать запрос POST?

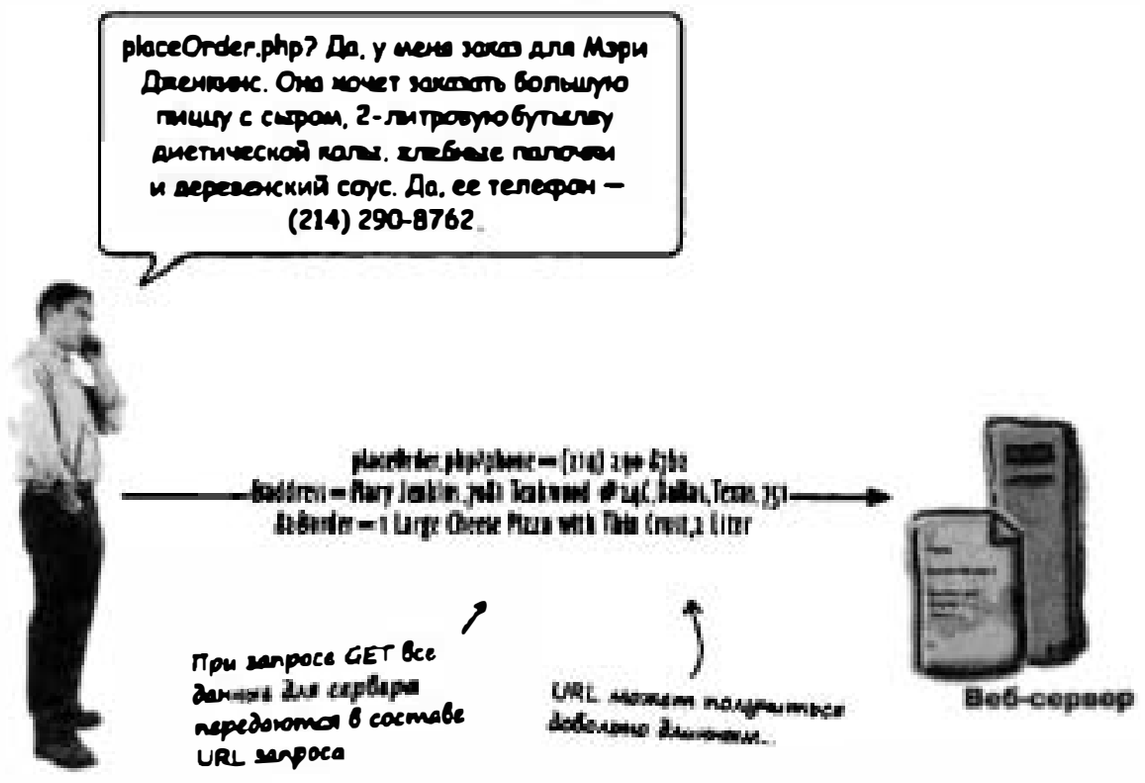
Запросы GET и POST

Бесспорно, потеря части заказа создаст большие проблемы для Break Neck. Но похоже, запрос POST избавит нас от всех бед?

Давайте повнимательнее присмотримся к обоим типам запросов.

Запросы GET передают данные в URL

Запрос GET передает данные серверу в составе URL:



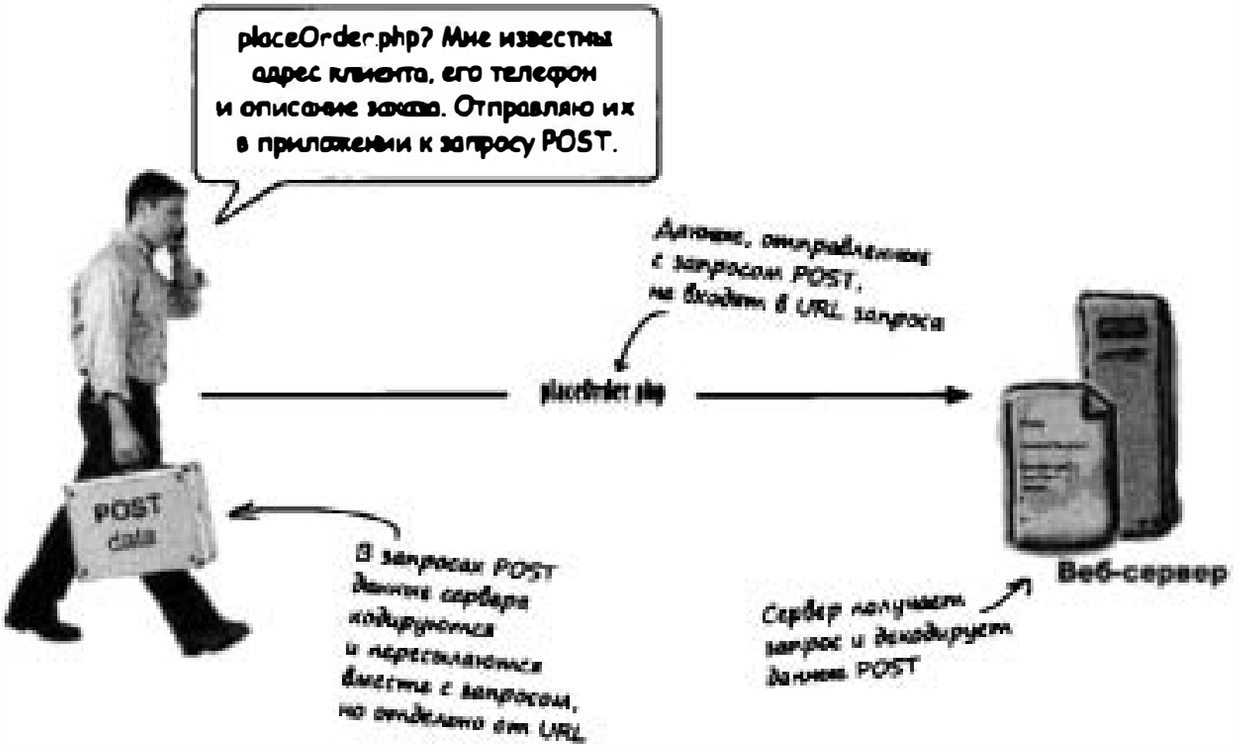
* В реальном запросе специальные символы в URL кодируются функцией JavaScript escape(). Мы специально оставили их незакодированными, чтобы URL выглядел более понятно

ШЕВЕЛИМ МОЗГАМИ

Как вы думаете, что происходит с данными запроса GET, если URL оказался слишком длинным для браузера или веб-сервера?

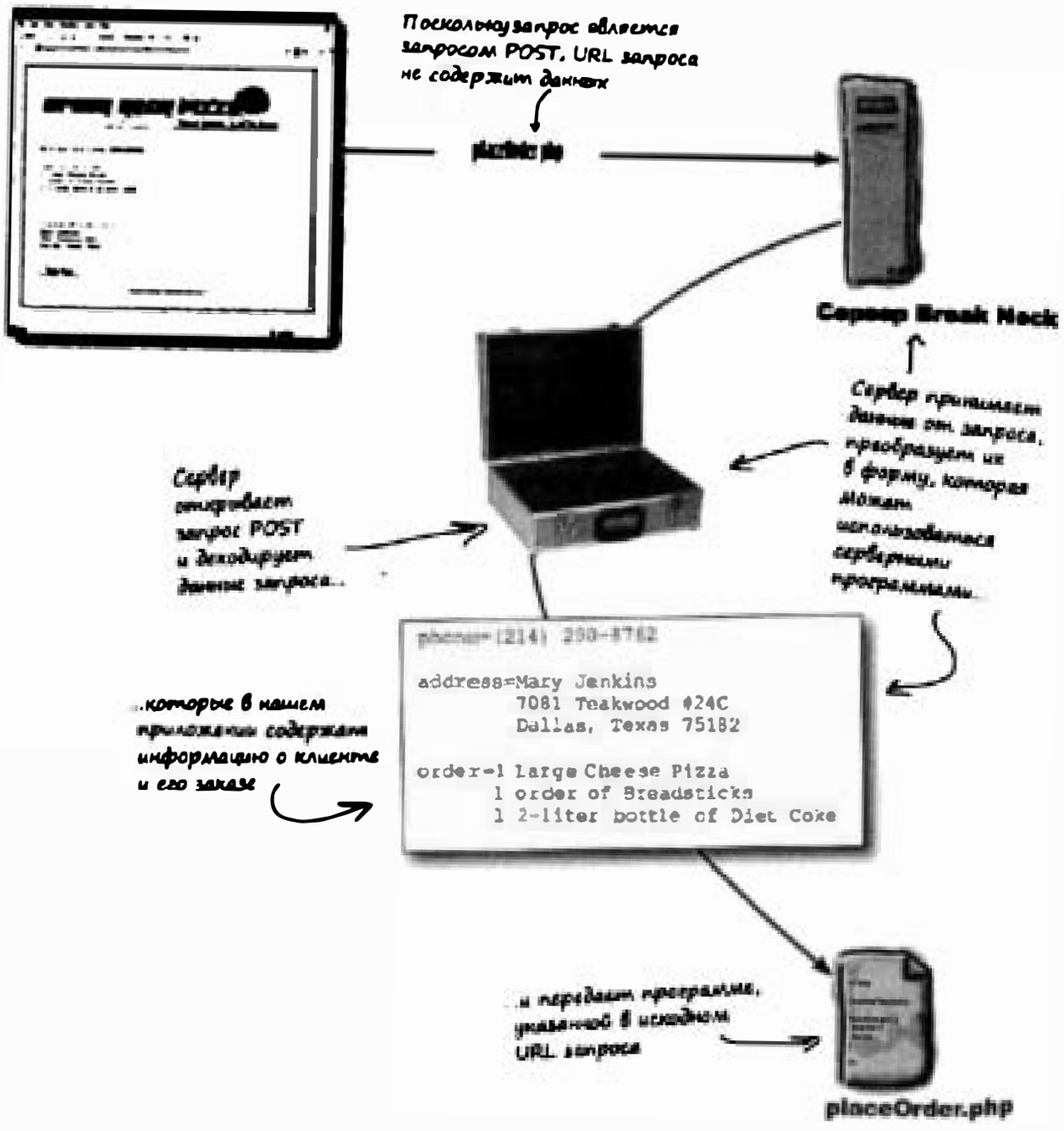
Запросы POST передают данные отдельно от URL запроса

В запросах POST данные, передаваемые серверу, отделены от URL, и их длина не ограничена:



Декодирование данных POST веб-сервером

При получении запроса POST веб-сервер определяет тип полученных данных и передает эту информацию программе, указанной в URL запроса.



Отправка дополнительных данных в запросе POST

Возвращаемся к шагу 2... Мы должны снова обновить функцию submitOrder()

Похоже, POST — именно то, что необходимо для сохранения данных, передаваемых сценарию Break Neck placeOrder.php. Давайте изменим код JavaScript в placeOrder() так, чтобы вместо запроса GET использовался запрос POST:

```
Функция submitOrder() находится в файле pizza.js
function submitOrder() {
  var phone = document.getElementById("phone").value;
  var address = document.getElementById("address").value;
  var order = document.getElementById("order").value;
  var url = [redacted]:phone=" + escape(phone) +
            "&address=" + escape(address) +
            "&order=" + escape(order);
  url = url + "&dummy=" + new Date().getTime();
  request.open([redacted], url, true);
  request.onreadystatechange = showConfirmation;
  request.send("phone=" + escape(phone) +
               "&address=" + escape(address) +
               "&order=" + escape(order));
}
```

Сначала все данные формы удаляются из URL запроса

Не потеряйте точку с запятой в конце строки!

Указываем open(), что вместо запроса GET должен использоваться запрос POST

После перехода на запрос POST фиктивный параметр нам более не понадобится

Отправляем заказ на сервер Break Neck функцией send()

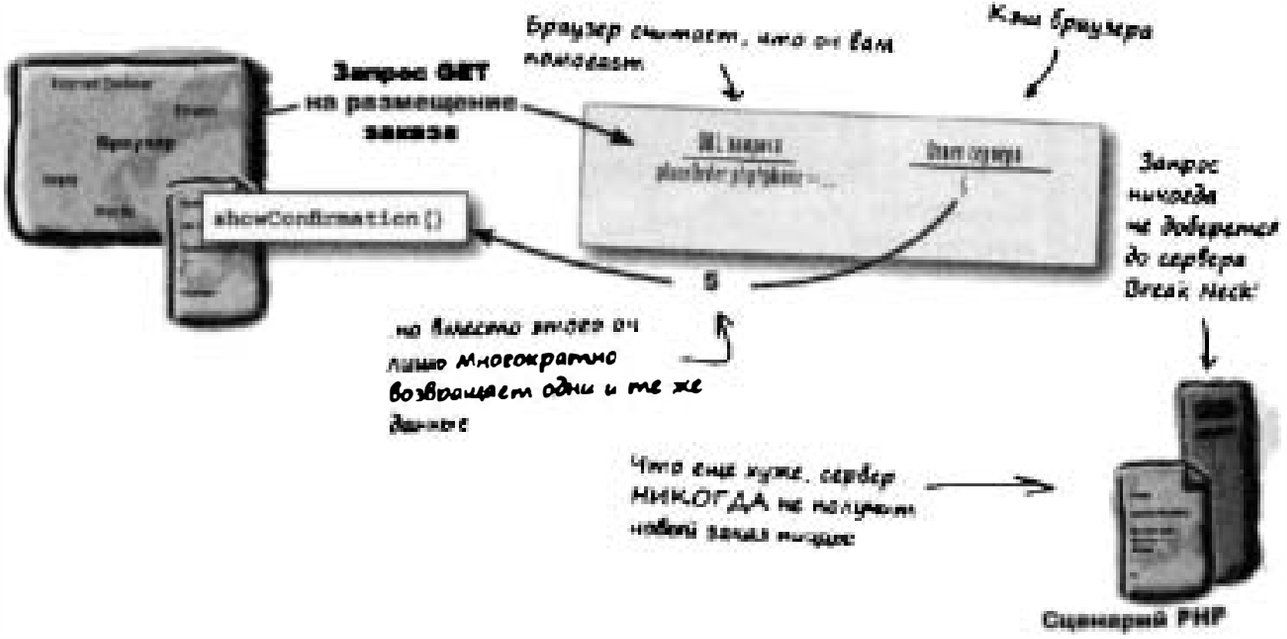
При вызове send() указываются пары «имя/значение», как в конце URL запроса в GET-версии кода

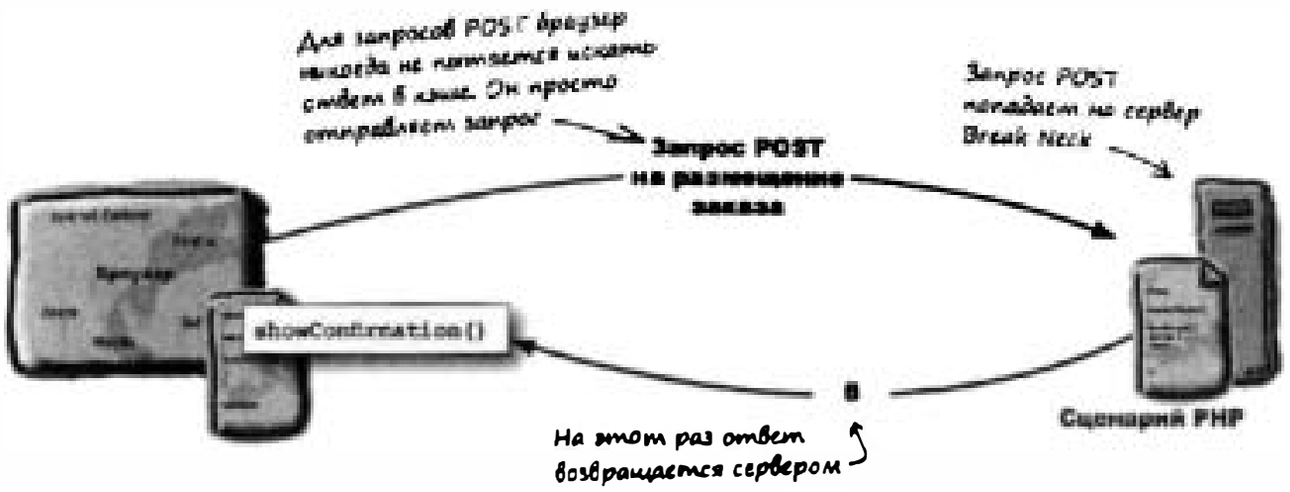
Просто сделайте это

Откройте свою копию файла pizza.js и найдите функцию **submitOrder()**. Измените код функции таким образом, чтобы заказ клиента передавался сценарию placeOrder.php с использованием запроса POST (вместо GET). Когда ваша функция будет совпадать с приведенной ранее, сохраните файл pizza.js, перезагрузите страницу — мы должны обсудить еще несколько вопросов, прежде чем переходить к очередной фазе тестирования.

Браузеры кэшируют запросы GET

Браузеры точно знают, какие данные передаются в запросах GET... Все данные хранятся в URL запроса. Когда браузер считает, что запрос не изменился (потому что в URL передаются те же данные), он пытается «помочь» и возвращает для него кэшированный ответ.





Браузеры не любят секреты

Браузер не знает, какие данные могут входить в запрос POST, так как эти данные не являются частью URL-запроса. А раз браузер не уверен в содержании запроса, он передает все запросы POST серверу, не пытаясь кэшировать ответы.

Браузеры не кэшируют запросы POST.

В: Теперь, когда мы перешли на запрос POST, ограничивается ли длина данных, передаваемых в запросе?

О: Нет. В запросах POST можно передавать любой объем данных. Таким образом, ваш крупный заказ для следующего «мальчишника» доберется до Вегаса к Near без всяких проблем.

В: Ограничение на длины запросов GET — единственная причина для перехода на запросы POST?

О: Во всяком случае, основная. Хотя большинство людей согласны с тем, что запросы POST лучше подходят для отправки заказов и проведения других «полностью завершенных» транзакций, основным доводом в пользу применения запросов POST является возможность отправки неограниченного объема информации в запросе POST. В запросах GET и браузер, и сервер устанавливают ограничения на максимальную длину URL запроса; все данные, выходящие за эти пределы, игнорируются.

В: Я всегда полагал, что запросы POST лучше защищены, чем запросы GET?

О: Запросы POST немного лучше защищены. Упаковка данных POST сопровождается парой дополнительных шагов: кодированием данных в браузере и декодированием на сервере. Тем не менее, декодирование данных POST выполняется весьма тривиально. Любой, кто хочет получить доступ к вашей информации, сможет извлечь ее из запроса POST почти с такой же легкостью, как из запроса GET.

Если вы действительно хотите защитить свой запрос, вам придется использовать защищенное сетевое подключение — например, протокол SSL. Впрочем, тема защиты выходит за рамки книги. В основном этой проблемой должны заниматься специалисты, обеспечивающие работу сервера, а не ваше Ajax-приложение.

В: Я слышал, что номера кредитных карт и другие личные данные всегда следует пересылать в запросах POST. Это правда?

О: Такое мнение объясняется скорее спецификой запроса, а не безопасностью данных. Помните: большинство веб-программистов предпочитают использовать запросы POST для отправки заказов, оплаты товаров или любых других «логически завершенных» транзакций. Таким образом, отправку запросов методом POST можно сравнить с выпиской чека. Если вам потребуется дополнительная безопасность, используйте подключение SSL или другой механизм сетевой безопасности.

С другой стороны, запросы GET обычно используются для получения данных (и это вполне логично, не правда ли?) А поскольку ввод конфиденциальной информации (паролей, номеров кредитной карты) почти всегда сопряжен с завершением транзакции или покупкой, эти данные в большинстве случаев отправляются серверу в запросах POST. И все не дело в типе транзакции, а не в дополнительной защите запроса.

В: При отправке запросов POST мы просто указываем данные запроса при вызове метода send() объекта запроса?

О: Совершенно верно. При этом даже используются такие же пары «имя/значение», как при добавлении данных в URL запроса (пары разделяются символом &). При передаче данных в запросах POST и GET существует единственное различие: перед первой «паре/значение», включаемой в URL запроса, должен стоять префикс «?». В запросах POST он «?» не нужен, поскольку пары передаются непосредственно при вызове метода send()

В: И это все? Больше ничего не нужно делать?

О: Давайте откроем обновленную форму Break Neck и посмотрим...

Тестирование запросов POST

Сделайте все, о чем говорится в упражнении «Просто сделай это» на с. 298, и загрузите файл pizza.html в браузере. Введите номер телефона, и пока сервер заполняет адрес — введите описание заказа. Щелкните на кнопке Order Pizza и посмотрите, что произойдет.

ОЙ... Сценарий placeOrder.php больше ничему. Заказ клиента почему-то не дошел до сервера

Часто задаваемые вопросы

В: Я опробовал pizza.html в Safari, и все прекрасно работает. Может, дело в моем коде?

О: Нет. Safari выполняет некоторые действия, из-за которых запросы POST работают без дополнительных шагов, необходимых для других браузеров. POST-версия Break Neck не работает в Internet Explorer, Opera и Firefox. А значит, мы определенно должны выяснить, в чем корень проблемы, и исправить приложение Break Neck.

Почему не работает запрос POST?

Все из-за сценария placeOrder.php. Мы указали open(), что для запроса должен использоваться метод POST, так что дело в сценарии.

Я уже сказал, что со сценарием все в порядке. Наверняка ты что-нибудь напутал в коде JavaScript.

Джим: Ты уверен? Могу поспорить, что ты забыл изменить свой сценарий, чтобы он принимал параметры POST. Ну давай, признайся! Исправь ошибку, и все будет нормально...

Фрэнк: У тебя проблемы со слухом? Я уже дважды сказал: мой сценарий принимает параметры GET и POST. А ты уверен, что твой код отправил описание заказа и адрес?

Джим: Уверен. Как только я понял, каким длинным может быть заказ, я сразу же перевел соответствующую часть кода JavaScript на использование запроса POST.

Фрэнк: Похоже, ты где-то ошибся.

Джим: Ни за что. Я указал телефон, адрес и описание заказа при вызове метода **send()** для моего объекта запроса... И потом лишний раз проверил. Я знаю, что данные добрались до веб-сервера.

Фрэнк: Но не до меня. Когда мой сценарий выполняется сервером, он проверяет параметры запроса **address** и **order** — и ничего не находит.

Джим: Погоди. Если я правильно отправляю данные серверу...

Фрэнк: ...А я запрашиваю данные у сервера, но ничего не получаю, значит...

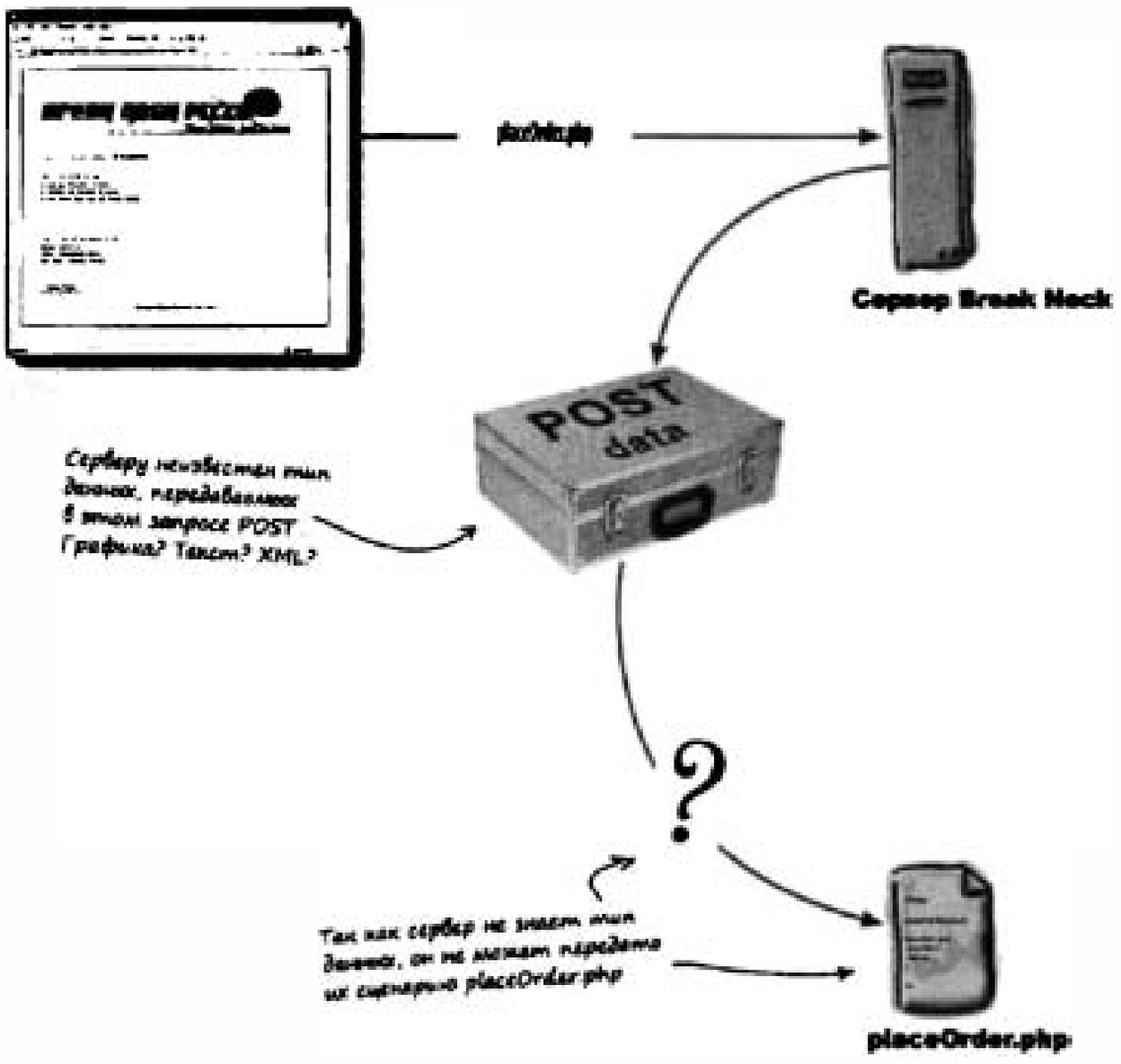
Хором: *Проблема с сервером!*

Джим — главный Ajax-программист в Break Neck.

А это Фрэнк... Он писал все сценарии PHP для Break Neck Pizza.

Таинственные данные POST

В рассуждениях Фрэнка и Джима есть здравое зерно. Помните, мы обсуждали, как сервер декодирует данные POST из вашего запроса? Открывданные запроса POST, сервер не знает, какой тип данных ему следует ожидать... а серверы действительно не любят неизвестности. Посмотрим, что происходит данными POST:



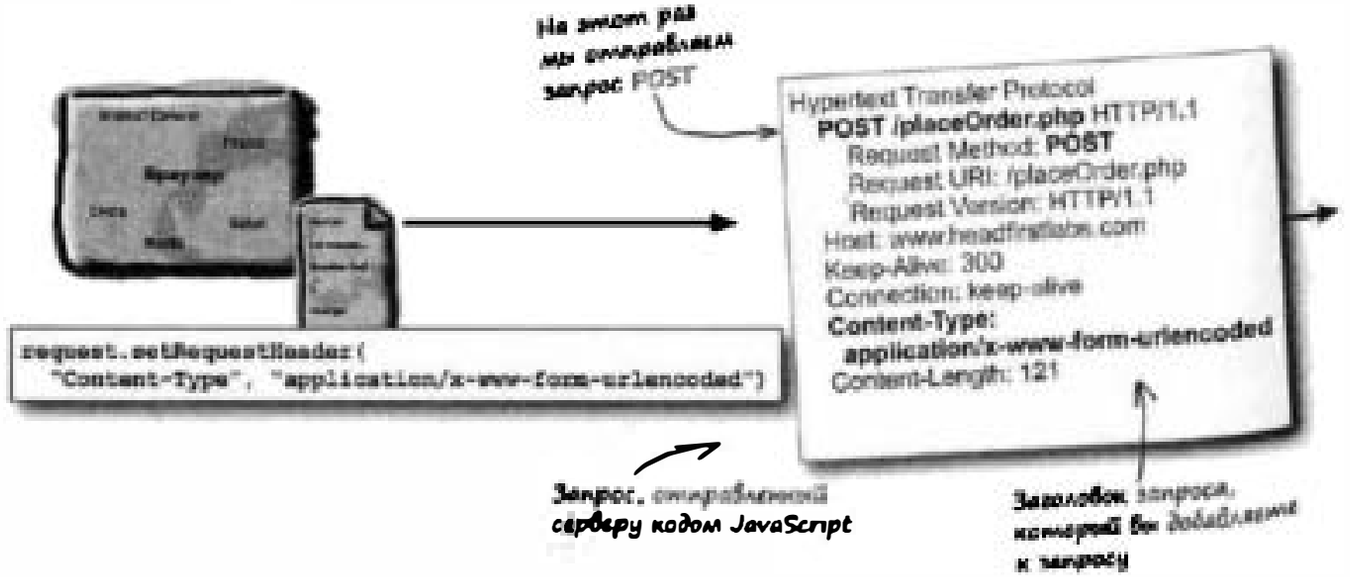
Понятно. В запросе GET данные входят в URL запроса, поэтому они заведомо являются текстом. Но в запросе POST можно отправить графику, XML или обычный текст... Значит, мы должны сообщить серверу, чего именно следует ожидать.

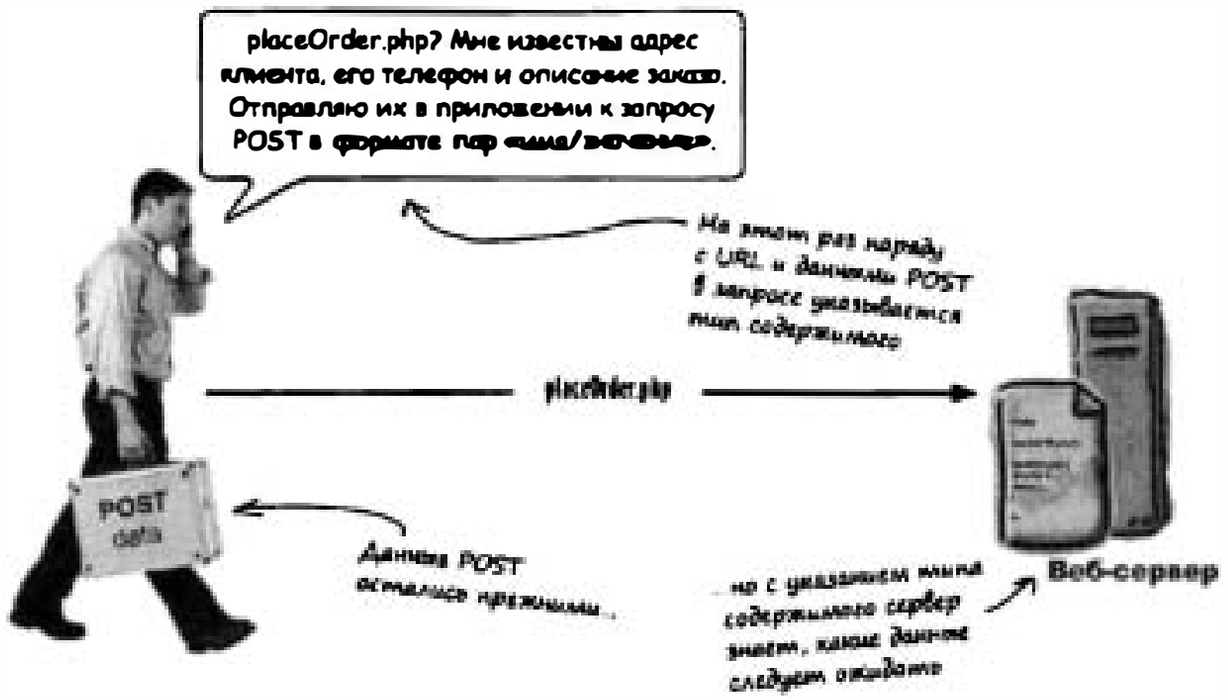
Мы должны указать тип содержимого

Запросы POST отнюдь не ограничиваются пересылкой простого текста... как вы узнаете в следующей главе. Но когда сервер получает ваш запрос POST, он не знает, с какими данными он работает — если вы не сообщите ему.

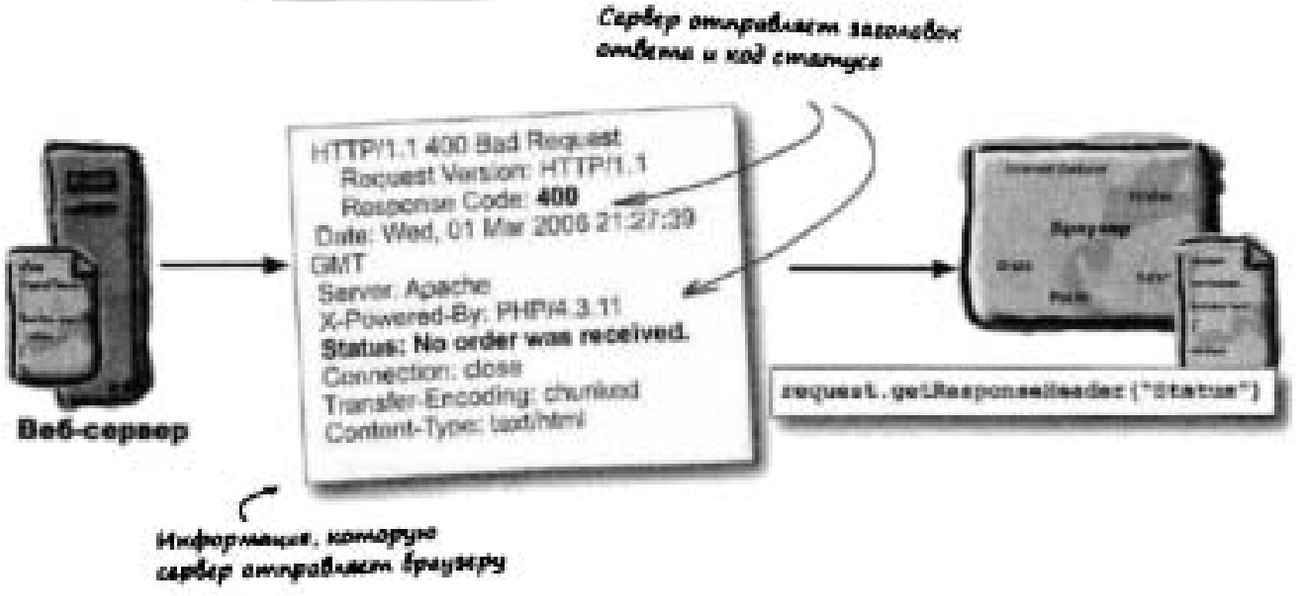
Как только сервер узнает, данные какого типа отправляются в запросе, он сможет декодировать данные POST и правильно обработать их. В приложении Break Neck это означает передачу текстового адреса и описания заказа сценарию placeOrder.php.

Сервер получает информацию от браузера в заголовках запросов





Сервер отправляет информацию браузеру в заголовках ответов



Устанавливаем тип содержимого

Итак, перед отправкой запроса следует указать тип содержимого для данных POST. Когда запрос будет отправлен, сервер получит URL запроса, данные POST и ожидаемый тип данных. Для передачи серверу любой информации о запросе используются заголовки запросов.

Вот как настраивается заголовок запроса для приложения Break Neck:

```
function submitOrder() {
  var phone = document.getElementById("phone").value;
  var address = document.getElementById("address").value;
  var order = document.getElementById("order").value;
  var url = "placeOrder.php";
  request.open("POST", url, true);
  request.onreadystatechange = showConfirmation;
  request.setRequestHeader("Content-Type",
                           "application/x-www-form-urlencoded");
  request.send("phone=" + escape(phone) +
               "&address=" + escape(address) +
               "&order=" + escape(order));
}
```

Функция setRequestHeader() позволяет включить в запрос дополнительную информацию (обычно предназначенную для сервера)

"Content-Type" — имя заголовка..

...а это — значение заголовка запроса

Мы сообщаем серверу, что данные кодируются по той же схеме, что и в URL запроса (как если бы они являлись частью запроса GET)

Часто задаваемые вопросы

В: Значит, заголовок запроса отправляется серверу наряду с запросом?

О: Да. Все заголовки запроса являются частью самого запроса. Браузер создает некоторые заголовки запроса автоматически, поэтому в действительности мы всего лишь добавляем новый заголовок запроса к уже существующим.

В: Есть ли другие типы содержимого, о которых мне следует знать?

О: Конечно, есть. В следующей главе мы рассмотрим тип содержимого для данных XML. Также существует множество других типов; чтобы получить списки всевозможных типов файлов и содержимого, проведите поиск в Google по строке «HTTP Content-Type».

Следующая проверка

Вкажите в pizza.js строку, создающую заголовок «Content-Type», и сохраните код JavaScript. Загрузите файл pizza.html в браузере и попробуйте снова внести заказ.

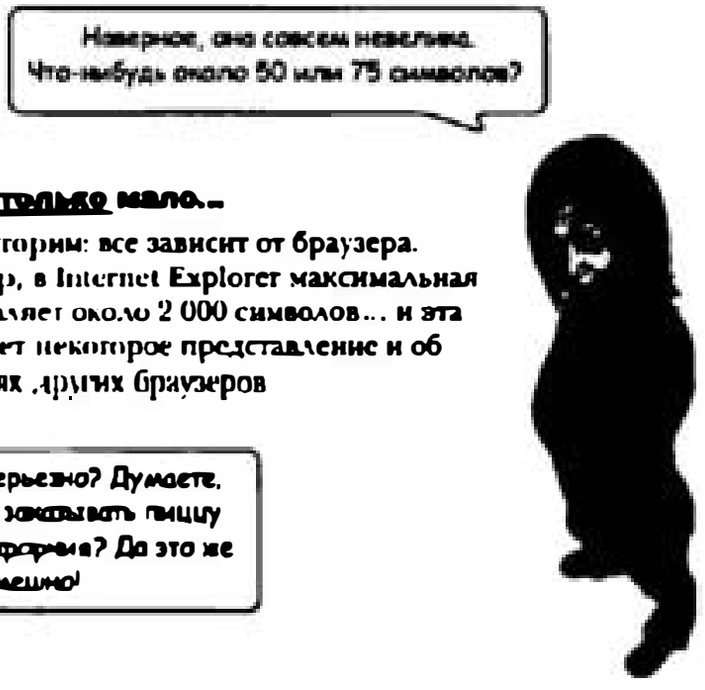
Поздравляем —

вы снова выручили Алекса!

С нас бесплатная пицца.

Ограничения зависят от браузера

Каждый браузер работает с URL так, как считает нужным. Это означает, что максимальная длина URL запроса будет разной для разных браузеров. Opera, Internet Explorer, Safari... Все зависит от того, какой браузер будет использоваться клиентом.



Ну, не настолько мало...

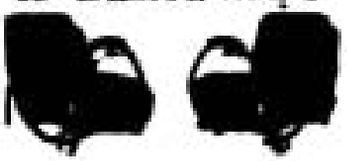
Еще раз повторим: все зависит от браузера. Но например, в Internet Explorer максимальная длина составляет около 2 000 символов... и эта величина дает некоторое представление и об ограничениях других браузеров

Хмм... Ну... Ээ...

Ладно, убедили. Вероятно, ситуации, в которых адрес клиента, номер телефона и описание заказа занимают более 2000 символов, будут относительно редкими... даже на «мальчишниках».

Но есть и другие веские причины для использования запросов POST...

Беседа за чашкой кофе



Наши гости:
вопрос POST и вопрос GET

POST: Привет. GET, я удивлен, что ты еще здесь. В последнее время со мной обращаются так часто, что я решил, что ты уехал на Фиджи или что-нибудь в этом роде.

GET: «Еще здесь»? О, да у меня полно работы. Кроме того, я предпочитаю, чтобы мои услуги пользовались спросом из-за того, что я делаю в действительности... а не в предположении большинства людей.

POST: Как это понимать?

GET: Ты когда-нибудь спрашивал, почему люди используют тебя вместо меня?

POST: Я и так знаю. Существует сотня причин, по которым я лучше тебя.

GET: Неужели? Назови хотя бы пять.

POST: Запросто. Даже не знаю, с чего начать. Во-первых, я безопаснее. Никто не использует тебя для отправки важной информации — например, номера кредитной карты или проверки состояния счета.

GET: (вздыхая) Если люди во что-то верят, это еще не значит, что это правда. На самом деле ты защищен ничуть не лучше меня. Любому компьютерному программисту составить данные POST ничуть не сложнее, чем прочитать URL запроса. Незашифрованные данные POST? Полчаса... И то если вдумчиво.

POST: Ладно, ладно. Вот тебе другая причина: со мной не нужно беспокоиться о длине данных запроса, а у тебя имеется это дурацкое ограничение длины.

GET: Да... И если ты сейчас передаешь 2000 символов текста, которые часто приходится переводить в пары «имя/значение», я скажу, насколько это существенно. Любому, кто работает с обычным текстом, этого объема более чем достаточно.

POST: Хмм. Хорошо, а кто захочет видеть все свои данные в адресной строке браузера? А? Как насчет этого, мистер «Абсолютно-нулевая-тайна-содержимого»?

GET: Так у нас тут конкурс красоты? А я-то думал, это книга про Ajax. Никто не видит URL в асинхронных запросах, даже если это запрос GET. Я все еще жду веских причин для использования запросов POST...

POST: Всем известно, что... правильнее... использовать POST для отправки данных серверу.

GET: Правильно. И это все?

POST: Гарри... Ну... Стой, стой, СТОЙ! Знаю!

GET: Наверняка это будет бесценный довод. Что теперь? Гарри Старк скажет, что POST лучше? Конечно, мы должны полагаться на подобные авторитетные мнения при принятии важных конструкторских решений...

POST: XML.

GET: А?

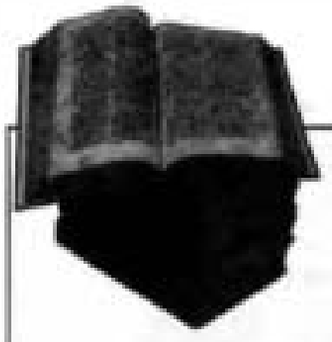
POST: XML, умник. Что ты на это скажешь?

GET: Ммм... Ну...

POST: Я так и думал! Ведь тебе приходится кодировать каждую угловую скобку, не так ли? А как насчет отправки данных? В XML это серьезная проблема.

GET: Ну, не такая уж серьезная... Я хочу сказать, что мы уже приблизились к концу книги, а об XML все еще только не говорили...

POST: Подождем до главы 6, приятель. Она посвящена XML, и в ней для тебя места не найдется. До встречи! Следующая глава зовет... В ней компания понадобится кто-нибудь, способный справиться с запросами XML!



Перебираем библиотеку Ajax

К настоящему моменту в вашей библиотеке Ajax и асинхронного программирования скопилось немало полезных инструментов. Прежде чем переходить к главе 6, попробуйте сформулировать то, что вы узнали до настоящего момента. Перечислите основные концепции Ajax, усвоенные в первых пяти главах.

Обзор на 60 секунд

- В запросах GET все данные, отправляемые серверу, включаются в URL запроса.
- В каждом браузере существует максимальная допустимая длина URL вместе со всеми включенными данными. Для большинства браузеров она составляет около 2 000 символов.
- При использовании запросов POST длина данных, передаваемых серверу, не ограничивается.
- В запросах POST данные передаются серверу отдельно от URL запроса. В URL включается только имя программы на сервере, которая должна обработать запрос.
- В запросах POST могут пересылаться данные разных типов: простой текст, XML, двоичные объекты (графика и файлы) и любые другие данные, которые сможет декодировать ваш браузер.
- Сервер не сможет определить тип полученных данных POST, если вы ему не сообщите.
- Для передачи дополнительной информации серверу используется метод setRequestHeader() объекта запроса JavaScript.
- В заголовке запроса Content-Type можно передать серверу информацию о том, какие данные отправляются в запросе POST.
- Тип содержимого «x-www-form-urlencoded» означает, что серверу передаются пары «имя/значение» в виде простого текста, как если бы эти данные были отправлены веб-формой.
- Запросы POST лишь незначительно лучше защищены по сравнению с запросами GET, и для защиты данных от посторонних и злонамеренных программистов в обоих случаях потребуются дополнительные уровни безопасности — такие как SSL (Secure Sockets Layer).

Решение упражнений

Просто сделайте это — решение

В папке chapter05/breakneck/ в архиве примеров вы найдете последнюю версию приложения Break Neck вместе с кодом HTML, CSS и PHP. Откройте файл pizza.html и измените форму так, чтобы при щелчке на кнопке запускалась функция **submitOrder()** (вместо прямой отправки сценария placeOrder.php на сервер Break Neck).

Раз уж мы занялись усовершенствованием Break Neck, вынесите весь код JavaScript за пределы pizza.html. Объект запроса можно создать в файле ajax.js, созданном нами в главе 3. Затем создайте новый файл JavaScript с именем pizza.js. Переместите функции Break Neck — **getCustomerInfo()**, **updatePage()** и **submitOrder()** — в этот файл. Не забудьте включить в код HTML элементы **<script>** со ссылками на эти файлы!

Далее приводится секция <head> файла pizza.html

```
<html>
 <head>
  <title>The New and Improved Break Neck Pizza</title>
  <link rel="stylesheet" type="text/css" href="breakneck.css" />
  <script type="text/javascript" src="ajax.js"> </script>
  <script type="text/javascript" src="pizza.js"> </script>
 </head>
```

В новой версии код HTML не содержит встроенного кода JavaScript

Эти два файла JavaScript вы уже создали

Файл ajax.js У вас этот файл должен был сохранится из главы 3

```
var request = null;
try {
  request = new XMLHttpRequest();
} catch (trymicrosoft) {
  try {
    request = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (othermicrosoft) {
    try {
      request = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (failed) {
      request = null;
    }
  }
}

if (request == null)
  alert("Error creating request object!");
```

```
function getCustomerInfo() {
  var phone = document.getElementById("phone").value;
  var url = "lookupCustomer.php?phone=" + escape(phone);
  request.open("GET", url, true);
  request.onreadystatechange = updatePage;
  request.send(null);
}

function updatePage() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      var customerAddress = request.responseText;
      document.getElementById("address").value = customerAddress;
    } else
      alert("Error! Request status is " + request.status);
  }
}

function submitOrder() {
  var phone = document.getElementById("phone");
  var address = document.getElementById("address");
  var order = document.getElementById("order");
  var url = "placeOrder.php?phone=" + escape(phone) +
            "&address=" + escape(address) +
            "&order=" + escape(order);
  url = url + "&dummy=" + new Date().getTime();
  request.open("GET", url, true);
  request.onreadystatechange = showConfirmation;
  request.send(null);
}
```

Функции, специфические для приложения Break Neck, должны хранится в файле с именем pizza.js


...мы следим за вами...

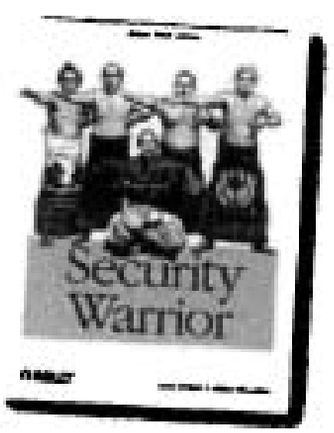
Вы же не думали, что с нами покончено? Проект «Хаос» не закроется от одного мелкого поражения. Кроме того, мы еще не отомстили этим торговцам пиццей... только посмотрите, что мы придумали на этот раз. Они даже не поймут, откуда пришла беда!

Вы нашли этот диктофон на кровати, рядом с упавшей лампой

Как создать проблемы в Break Neck

Похоже, у проекта «Хаос» появился новый план относительно того, как навредить Break Neck. На этот раз они придумали нечто более хитроумное... помогла предварительная подготовка.

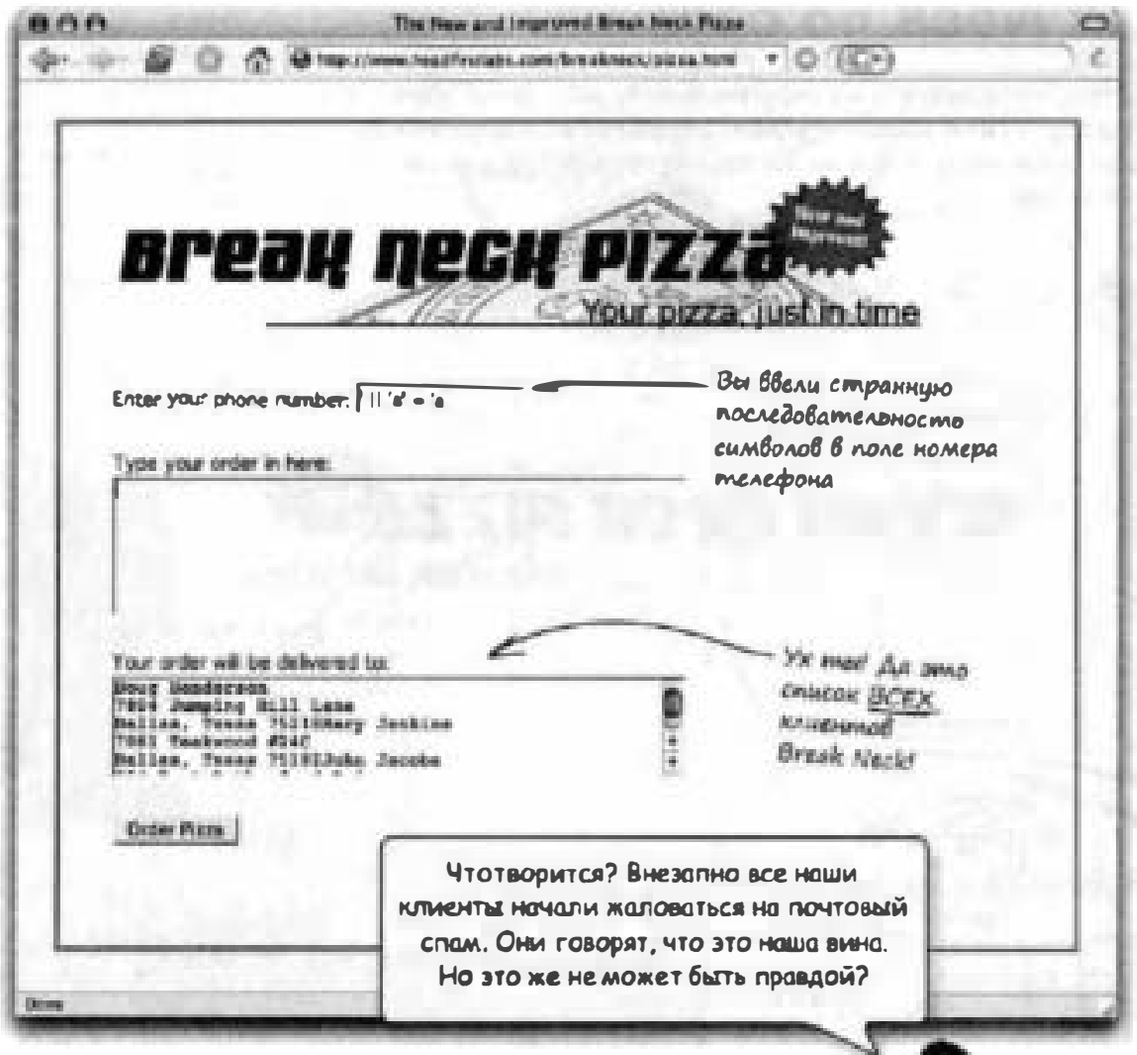
Хм... Интересно, чего стоило «Хаосу» в этой книге, чтобы сломать приложение Break Neck?



Break Neck по сети

Откройте браузер и зайдите на «официальный» сайт Break Neck Pizza по адресу **http://www.headfirstlabs.com/breakneck/pizza.html**. Затем выполните инструкции из записки проекта «Хаос», оставленной в утренней газете.

стоп! Обязательно проделайте это по Интернету – скорее всего, с локальной копией pizza.html ничего не выйдет.

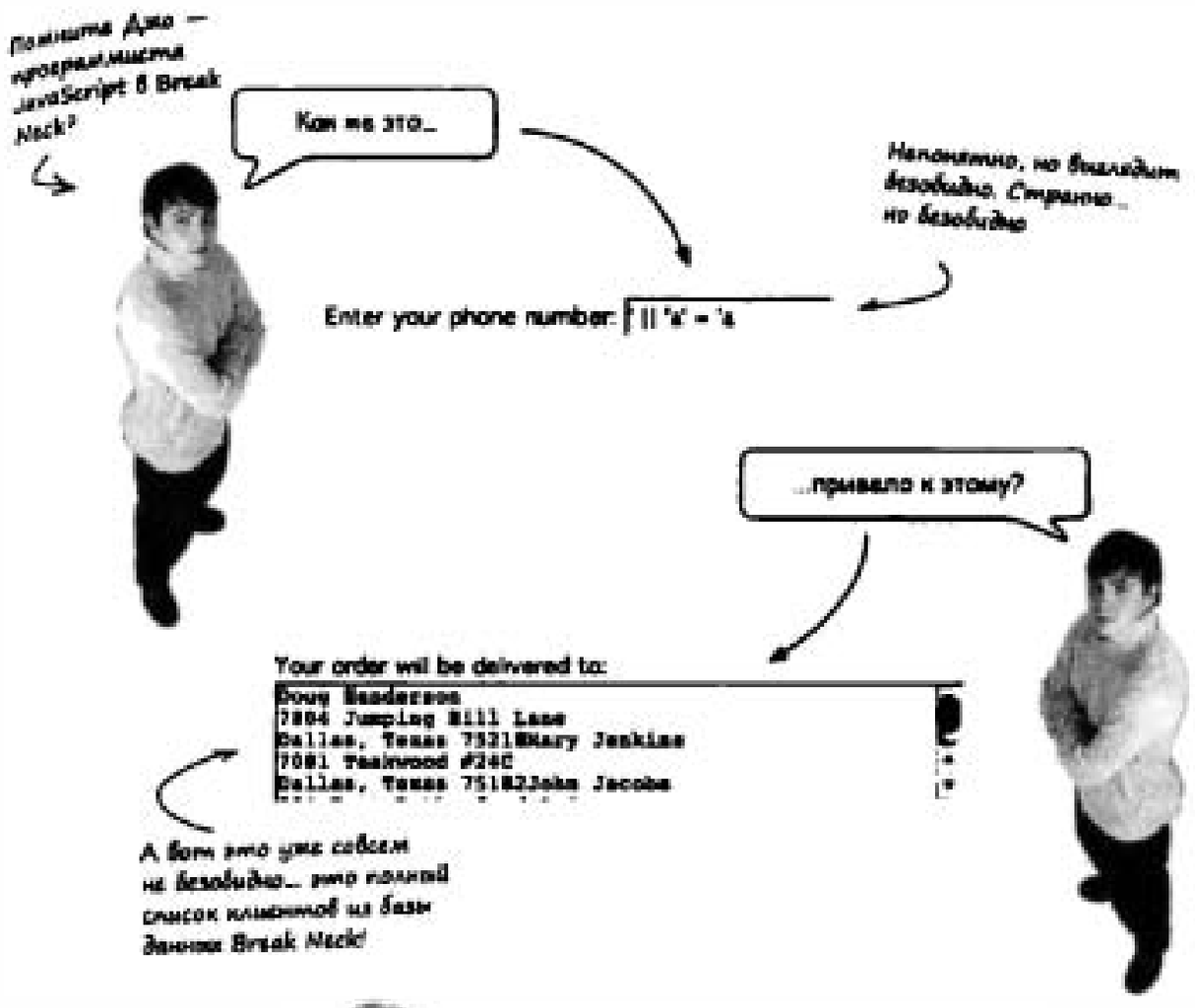


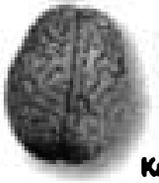
А ведь виновником действительно является Break Neck!

Похоже, нам снова придется спасать положение

Знакомьтесь: внедрение SQL

Приложение Break Neck стало жертвой атаки, называемой *внедрением SQL*, и теперь расплачиваться приходится всем клиентам.





ШЕВЕЛИМ МОЗГАМИ

Как вы думаете, что произошло? Как закрыть посторонний доступ к базе данных имен и адресов Break Neck?

Инспектор SQL

Чтобы понять, что произошло, необходимо разобраться в коде SQL. Ниже приведена часть сценария lookupCustomer.php, получающего запрос при вводе телефонного номера на форме Break Neck... Мы выделили строку SQL, которая запрашивает у сервера адрес клиента для номера телефона, введенного на веб-форме.

```
if (!$conn)
  die("Error connecting to MySQL: " . mysql_error());

if (!mysql_select_db("headfirst", $conn))
  die("Error selecting Head First database: " . mysql_error());

$phone = preg_replace("/[\. \(\)\-]/", "", $_REQUEST['phone']);
$select = "SELECT *";
$from   = "  FROM hraj_breakneck";
$where  = " WHERE phone = '" . $phone . "'";

$queryResult = mysql_query($select . $from . $where);

if (!$queryResult)
  die('Error retrieving customer from the database.');
```

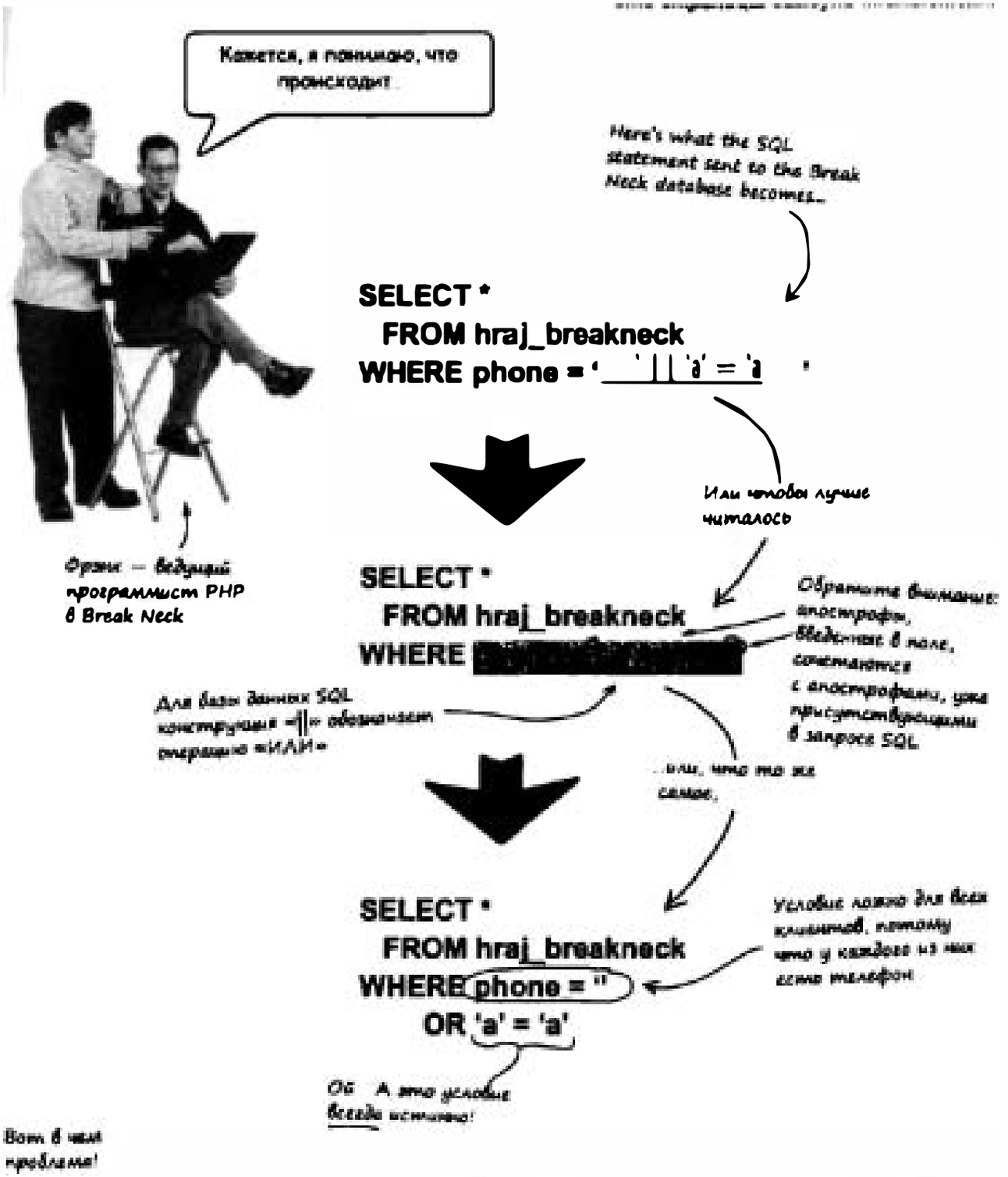
Это сценарий lookupCustomer.php, с которым вы встречались в главе 2.

Мы должны взять этот запрос и добавить к нему специальные символы, которые проект «Хаос» потребовал ввести на форме Break Neck. Так мы будем точно знать, что передается в базу данных Break Neck... А может, и сумеем разобраться в том, что происходит в приложении.

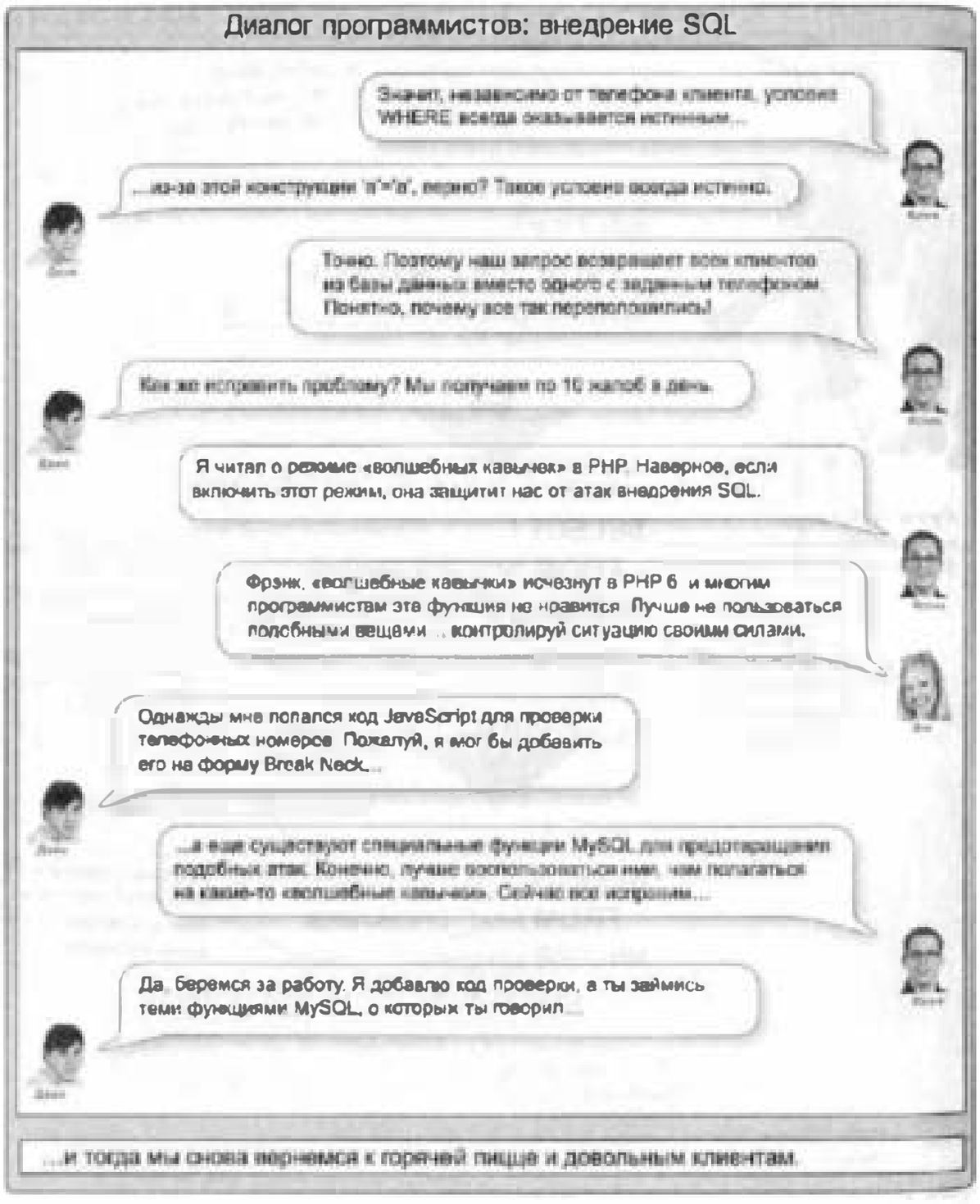
```
SELECT *
  FROM hraj_breakneck
WHERE phone = '_____'
```

Код SQL, созданный сценарием lookupCustomer.php

Впишите то, что вы ввели в поле телефонного номера



Условие в запросе SQL истинно для каждого клиента!


Диалог программистов: внедрение SQL
Значит, независимо от телефона клиента, условие WHERE всегда оказывается истинным...
...из-за этой конструкции '1'='1', верно? Такое условие всегда истинно.
Точно. Поэтому наш запрос возвращает всех клиентов из базы данных вместо одного с заданным телефоном. Понятно, почему все так переполошились!
Как же исправить проблему? Мы получаем по 10 жалоб в день.
Я читал о режиме «волшебных кавычек» в PHP. Наверное, если включить этот режим, она защитит нас от атак внедрения SQL.
Фрэнк, «волшебные кавычки» исчезнут в PHP 6, и многим программистам эта функция не нравится. Лучше не пользоваться подобными вещами... контролируй ситуацию своими силами.
Однажды мне попался код JavaScript для проверки телефонных номеров. Пожалуй, я мог бы добавить его на форму Break Neck...
...а еще существуют специальные функции MySQL для предотвращения подобных атак. Конечно, лучше воспользоваться ими, чем полагаться на какие-то «волшебные кавычки». Сейчас все исправим...
Да, беремся за работу. Я добавлю код проверки, а ты займись теми функциями MySQL, о которых ты говорил...
...и тогда мы снова вернемся к горячей пицце и довольным клиентам.

Одну секунду... Так это серверная проблема? Почему я должна писать дополнительный код JavaScript, если я не виновата?

Проверка защитит веб-приложение от хакеров. Всегда проверяйте пользовательский ввод.

Но это ваше веб-приложение

Конечно, внедрение SQL в большей степени отражается на людях, работающих над кодом PHP (таких, как Фрэнк и Энн), нежели на асинхронных программистах JavaScript. Но если оставить такой дефект, будьте уверены: вы о нем еще услышите!

Кроме того, небольшой объем дополнительного кода JavaScript сделает ваше приложение более защищенным. Проверка поможет всем, а не только парням, работающим над серверными программами.

Часто задаваемые вопросы

В: Я совершенно запуталась в PHP. Мне казалось, что эта книга посвящена асинхронному программированию!

О: Даже если вы не понимаете PHP, ничего страшного. Многие программисты занимаются исключительно кодом JavaScript в веб-приложениях и сотрудничают с другой группой программистов занимающихся серверной стороной приложения.

Главное — запомните, что на стороне сервера могут возникнуть проблемы

В: Не понимаю, что творится. Да и зачем мне разбираться во внедрении SQL?

О: Необходимо знать только одно: что проблемы возможны... и в большинстве случаев они решаются. А теперь поговорите со своими серверными коллегами и помогите им сделать так, чтобы адреса и телефоны ваших клиентов не достались посторонним

В: Так все дело во взаимодействии? Между мной и другими участниками группы?

О: Вот именно! Во многих ситуациях вы не знаете, как решать проблему (и даже не знаете о ее существовании) Но немного времени, проведенного за общением с другими участниками группы, принесет огромную пользу веб-приложению. Попутно у вас даже могут возникнуть новые идеи по поводу улучшения вашего кода... скажем, проверки введенных данных...

Защита от внедрения SQL → в коде JavaScript

Заглянув в архив примеров, вы найдете в нем папку chapter05-interludebreakneck. В ней находятся все файлы Break Neck, а также новый вспомогательный файл с именем validation-utils.js. Проверка данных в локальной версии Break Neck Pizza организуется следующим образом:

① Проверьте наличие файла validation-utils.js

Фрагмент файла validation-utils.js. Файл находится в папке chapter05-interlude/breakneck архива примеров

Функция validatePhone() используется для проверки телефонных номеров, введенных на форме Break Neck

```
function validatePhone(phoneNumber) {
  if (phoneNumber == null) {
    alert("Please enter your phone number.");
    return false;
```

validation-utils.js

② Включите ссылку на validation-utils.js в веб-форму Break Neck

Верхняя часть pizza.html. Используйте версию из главы 5 или возьмите обновленную версию из архива примеров

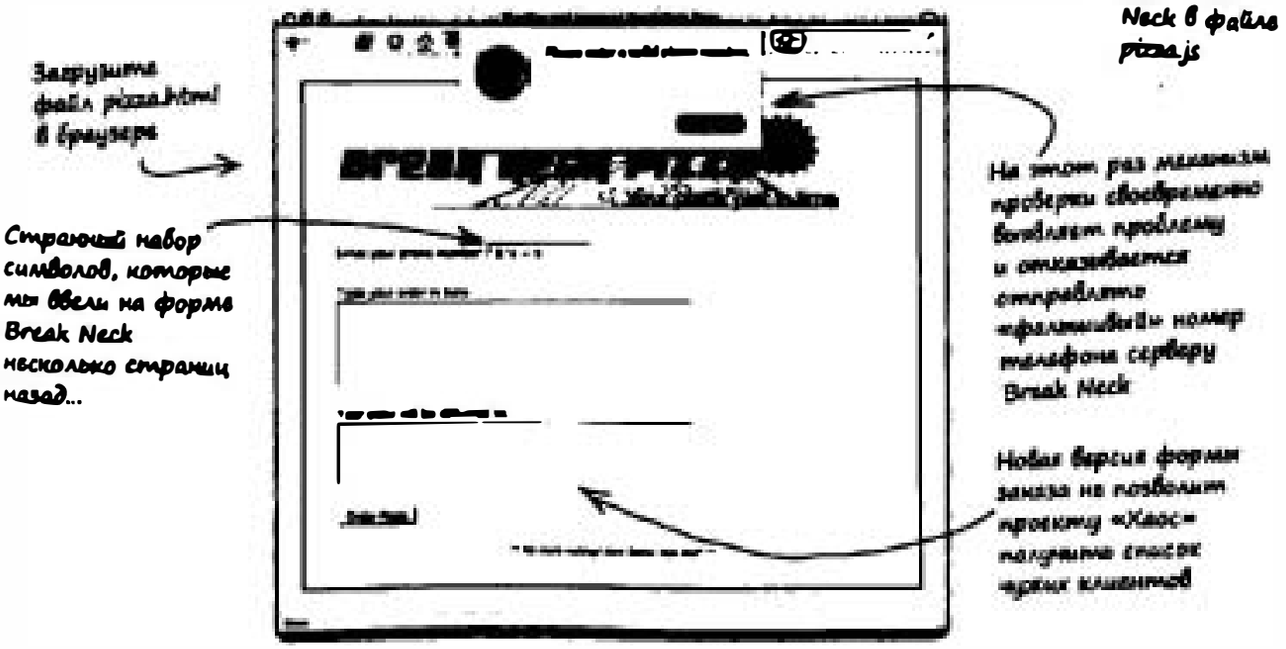
Эта строка предоставляет коду доступ к функциям validation-utils.js и остальному коду JavaScript

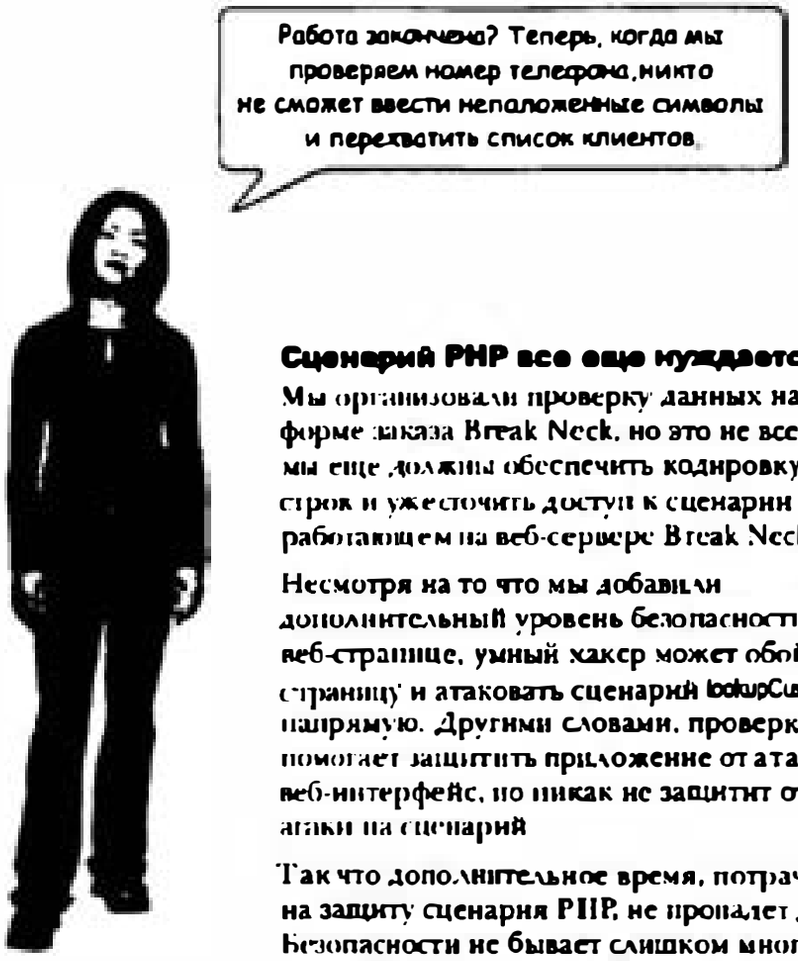
```
<html>
 <head>
  <title>The New and Improved Break Neck Pizza</title>
  <link rel="stylesheet" type="text/css"
        href="breakneck.css" media="screen" />
  <script type="text/javascript" src="ajax.js"> </script>
  <script type="text/javascript" src="pizza.js"> </script>
  <script type="text/javascript"
          src="validation-utils.js"> </script>
 </head>
```

pizza.html

③ Проверьте телефонный номер перед отправкой его веб-серверу Break Neck

④ Проверьте внесенные [illegible]





Безопасности слишком много не бывает.

Сценарий PHP все еще нуждается в защите

Мы организовали проверку данных на форме заказа Break Neck, но это не все: мы еще должны обеспечить кодировку строк и ужесточить доступ к сценарии PHP, работающем на веб-сервере Break Neck.

Несмотря на то что мы добавили дополнительный уровень безопасности на веб-странице, умный хакер может обойти страницу и атаковать сценарий `bobupCustomer.php` напрямую. Другими словами, проверка данных помогает защитить приложение от атак через веб-интерфейс, но никак не защитит от прямой атаки на сценарий.

Так что дополнительное время, потраченное на защиту сценария PHP, не пропадет даром. Безопасности не бывает слишком много. Никогда не знаешь, когда очередной 12-летний умник изобретет новый способ подобраться к вашим данным и создаст проблемы для клиентов.

Что со сценарием PHP?

Давайте рассмотрим сценарий lookupCustomer.php и подумаем, в чем его нужно улучшить.

Если вы не разбираетесь в PHP, ничего страшного... постарайтесь хотя бы в общих чертах понять, что здесь происходит. Позднее вы расскажете специалистам по PHP, на что им следует обратить внимание

Хотя мы избавляемся от стандартных символов форматирования телефонных номеров (таких, как «)», «(» и «–»), остается одна проблема...

```
<?php

// Connect to database
$conn = @mysql_connect("mysql.headfirstlabs.com",
                       "secret", "really-secret");
if (!$conn)
  die("Error connecting to MySQL: " . mysql_error());

if (!mysql_select_db("headfirst", $conn))
  die("Error selecting Head First database: " . mysql_error());

$phone = preg_replace("/[\. \(\)\-]/", "", $_REQUEST['phone']);
$select = 'SELECT *';
$from   = ' FROM brad_breakneck';
$where  = ' WHERE phone = \'' . $phone . '\'';

$queryResult = @mysql_query($select . $from . $where);
if (!$queryResult)
  die("Error retrieving customer from the database.");

while ($row = mysql_fetch_array($queryResult)) {
  echo $row['name'] . "\n" .
       $row['street'] . "\n" .
       $row['city'] . ", " .
       $row['state'] . " " .
       $row['zipCode'];
}

mysql_close($conn);

?>
```

...мы никак не защищаемся от символов, используемых в атаках внедрения SQL (например, апострофов)...

...и потенциально опасная строка может быть вставлена в запрос SQL

Другая потенциальная проблема. Сценарий перебирает записи полученного набора данных и выводит их все...

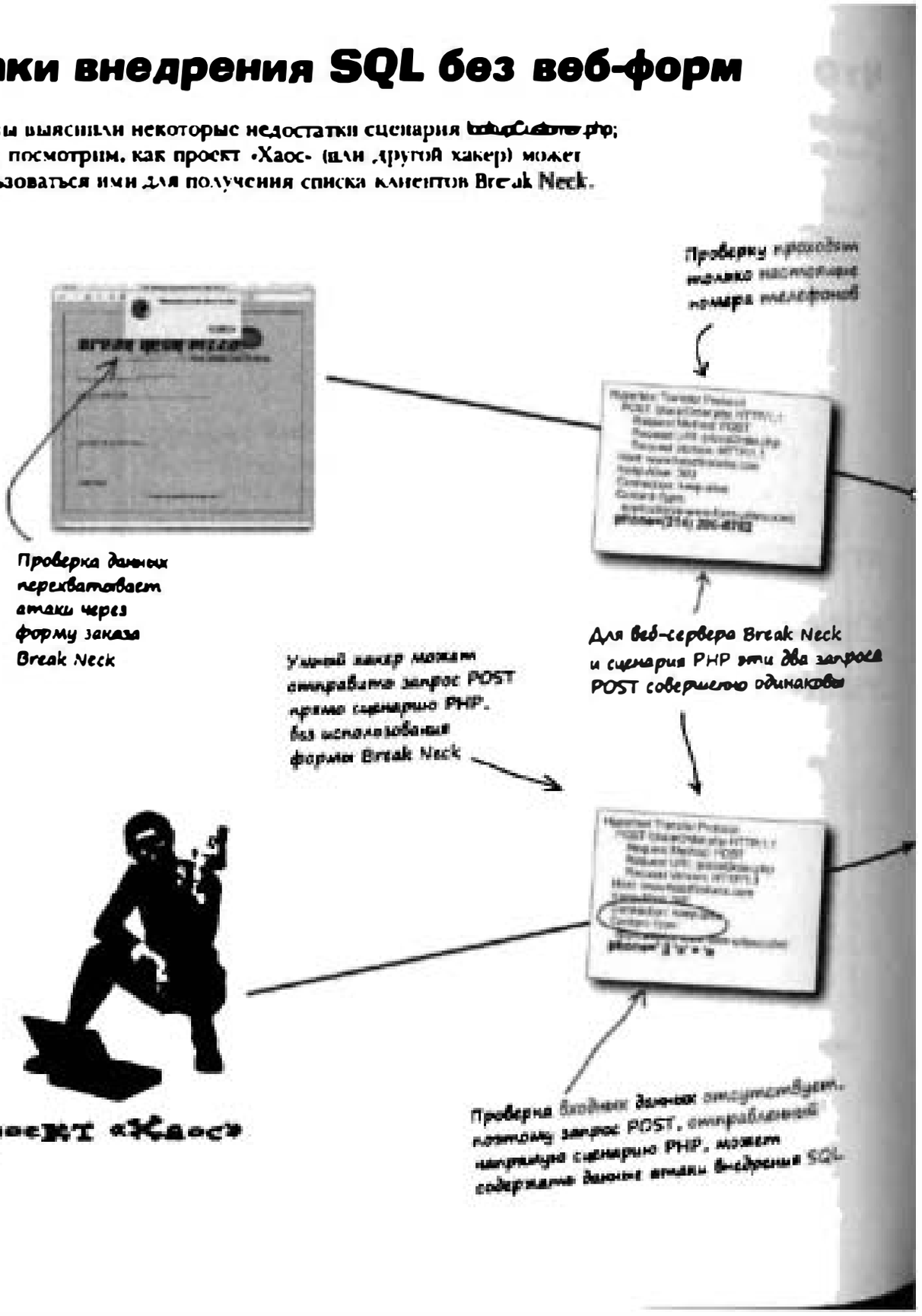
...но сценарий никогда не должен возвращать более одного клиента. Это надо исправить

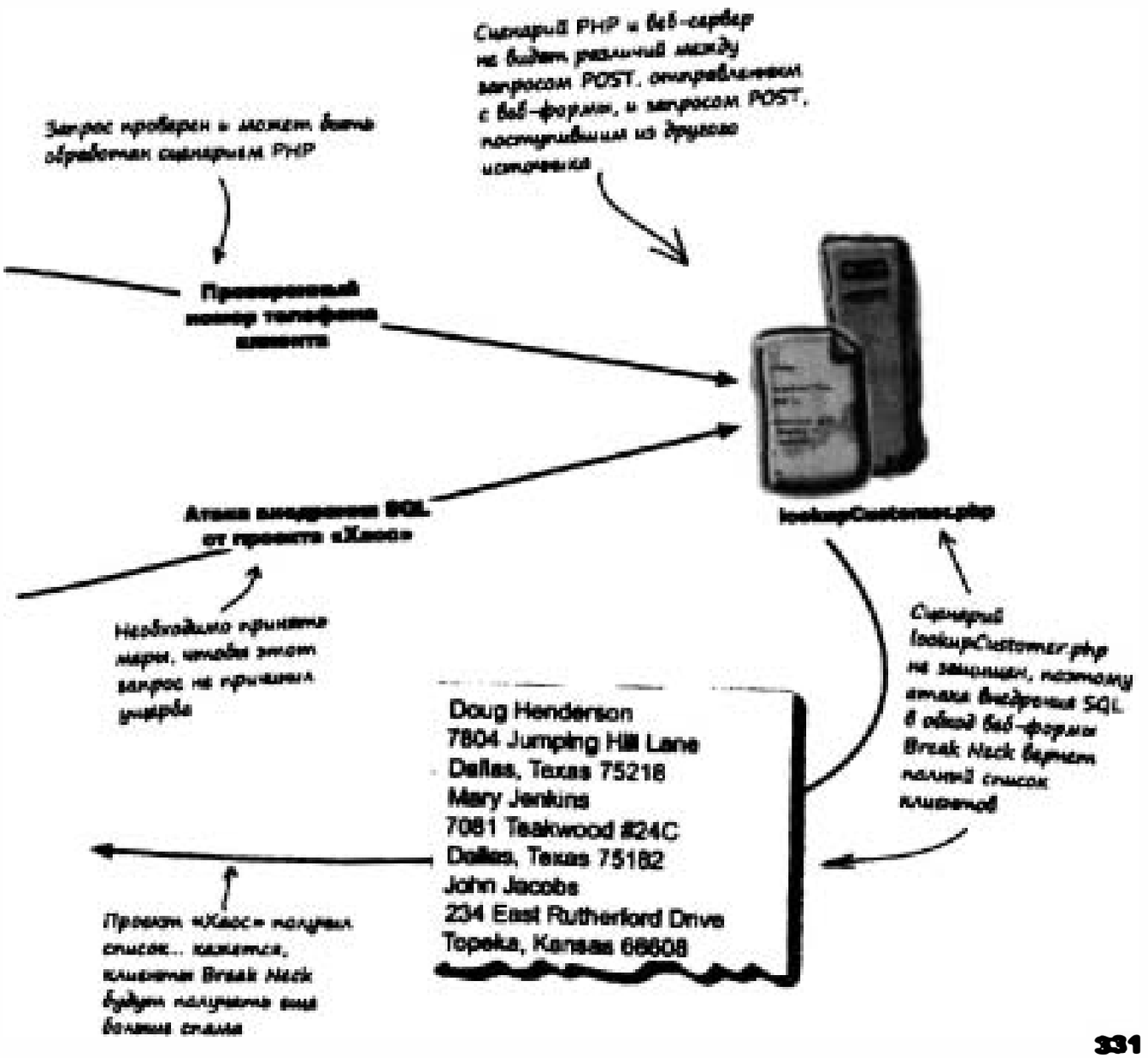
lookupCustomer.php

Помните сценарий из главы 2? Это тот сценарий PHP, к которому обращается с запросом функция getCustomerInfo()

Атаки внедрения SQL без веб-форм

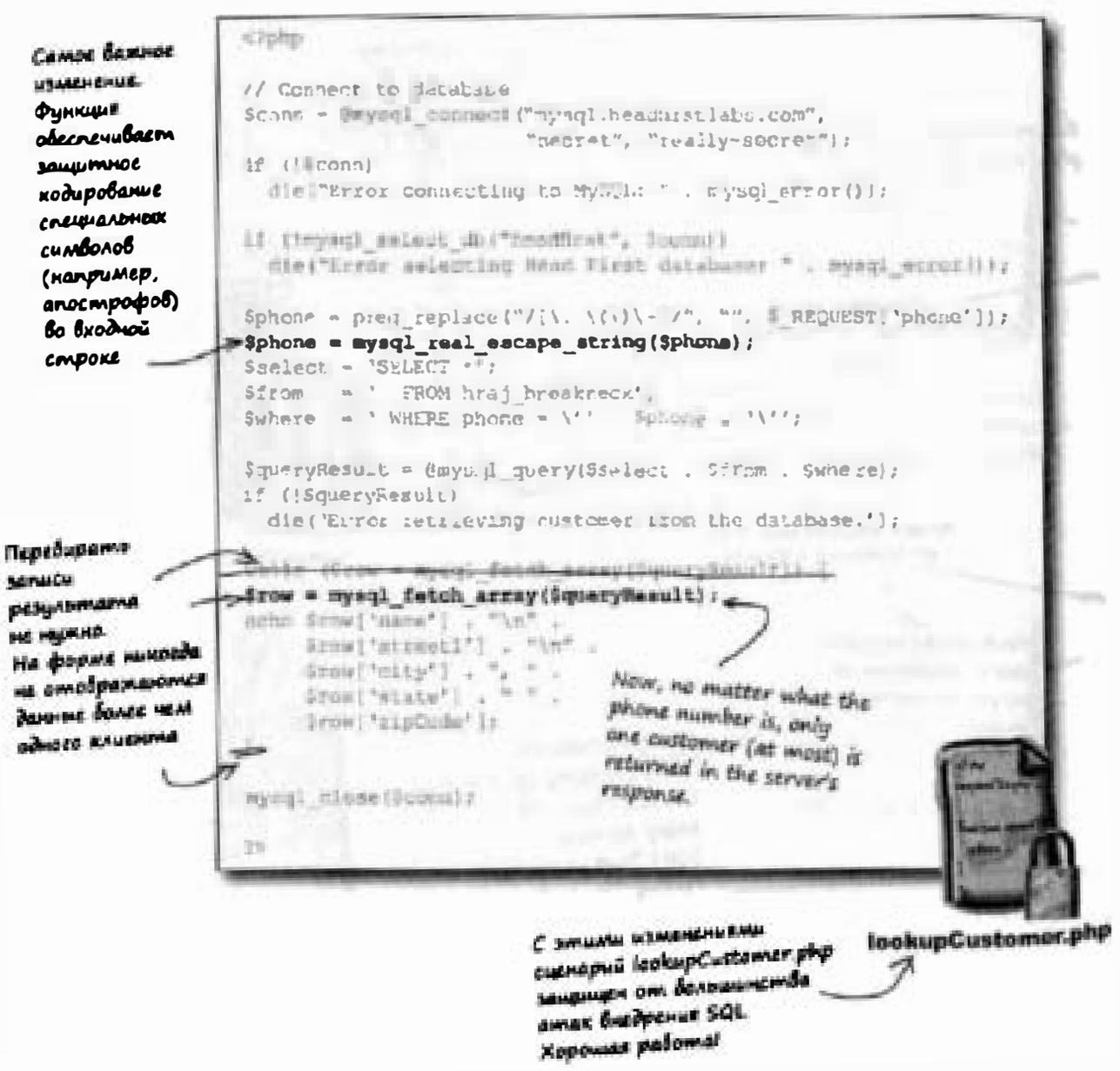
Итак, мы выяснили некоторые недостатки сценария [illegible].php; давайте посмотрим, как проект «Хаос» (или другой хакер) может воспользоваться ими для получения списка клиентов Break Neck.



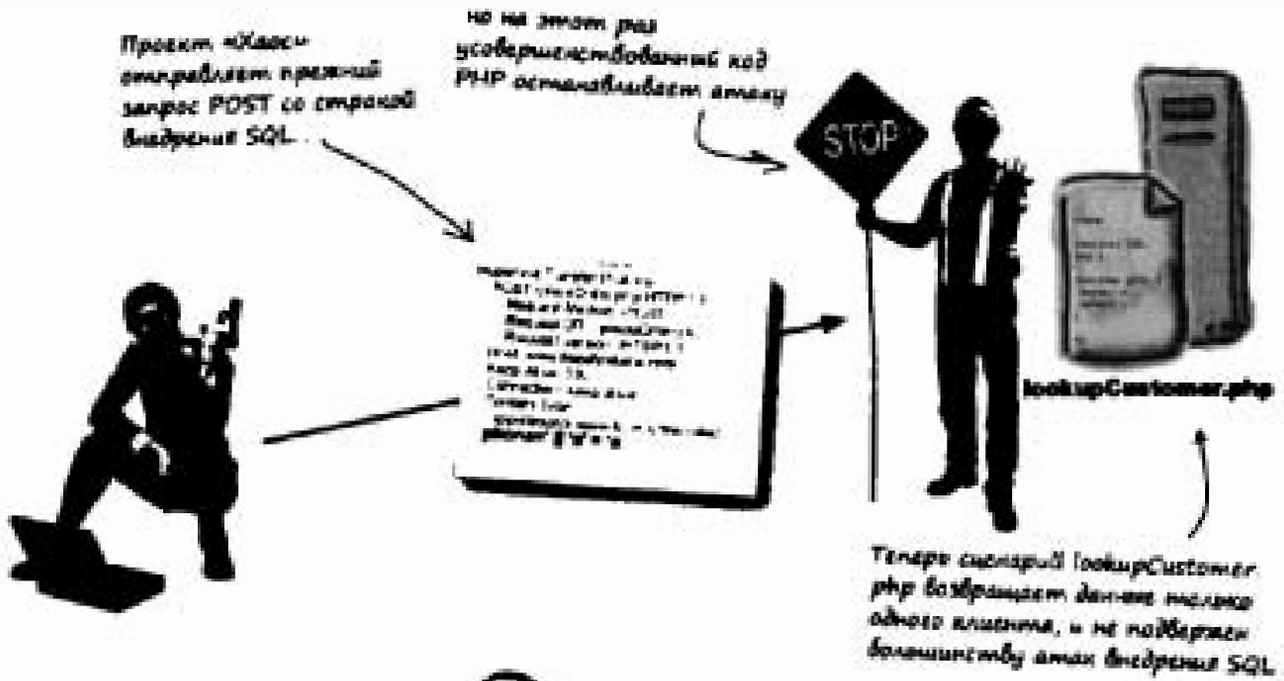

Сценарий PHP и веб-сервер не будет различий между запросом POST, отправленным с веб-формы, и запросом POST, поступившим из другого источника
Запрос проверен и может быть обработан сценарием PHP
Проверенный номер телефона клиента
Атака внедрения SQL от пронета «Хаос»
lookupCustomer.php
Необходимо принять меры, чтобы этот запрос не причинил ущерба
Сценарий lookupCustomer.php не защищен, поэтому атаки внедрения SQL в обход веб-формы Break Neck берет полный список клиентов
Doug Henderson
7804 Jumping Hill Lane
Dallas, Texas 75218
Mary Jenkins
7081 Teakwood #24C
Dallas, Texas 75182
John Jacobs
234 East Rutherford Drive
Topeka, Kansas 66608
Пронет «Хаос» получил список... кажется, клиенты Break Neck будут получать вид более стильный

Защита от внедрения SQL ⟶ в сценариях PHP

Не будем останавливаться на JavaScript. Давайте поможем Фрэнку внести некоторые исправления в код PHP, чтобы защитить сервер от атак внедрения SQL.



База данных клиентов защищена!



ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

В: mysql_real_escape_string()? Это еще что такое? Разве я не говорил, что не разбираюсь в программировании SQL?

О: mysql_real_escape_string() — функция PHP, которая кодирует все специальные символы в строке, в результате чего строка становится безопасной для использования в командах SQL. Функция работает только в базе данных MySQL, но аналогичные функции существуют во всех основных базах данных.

Не спорьте, если вы совершенно не знакомы с PHP или этими функциями. Главное, чтобы вы поговорили с программистами, работающими над серверными компонентами приложения. Скажите, чтобы они позаботились о защите своих сценариев.

В: И все это для защиты от внедрения SQL?

О: Внедрение SQL — всего лишь одна разновидность рисков безопасности для веб-приложений. Если форма приложения содержит поля, используемые для построения запросов SQL, хакеры часто пытаются вводить в них специальные строки (вроде той, которую нам передал проект «Хаос»), чтобы изменить содержимое базы данных или сохранить в ней посторонюю информацию.

К сожалению, существует немало других типов атак, также представляющих угрозу для вашего приложения. С другой стороны, с проверкой данных и дополнительными мерами безопасности на сервере можно обезопасить себя почти от любых угроз. Так что принимайте меры и защищайтесь!

Внедрение SQL — всего лишь верхушка айсберга... Мы еще вернемся тогда, когда вы менее всего будете этого ждать.

ПРОЕКТ «ХАОС»

6 Запросы и ответы XML

Больше, чем можно выразить словами

Кажется, будто вас никто не слушает? Когда вы пытаетесь с кем-то общаться, повседневного языка порой бывает недостаточно. До сих пор все наши запросы и ответы имели текстовый формат, но пришло время вырваться за рамки простого текста. В этой главе мы погрузимся в XML и узнаем, как заставить сервер передавать информацию, не ограничиваясь простым текстом. А если этого мало, вы научитесь использовать XML и в своих запросах (хотя это далеко не всегда стоит делать). Приготовьтесь... После чтения этой главы *ваши запросы и ответы уже никогда не будут прежними*.

Доктор Зигмунд: А-а-а, конечно. У многих моих пациентов та же проблема. Но обычно она существует лишь в вашем воображении.

Сервер: Но только не в моем случае! То есть все хотят говорить со мной. Мне постоянно кто-нибудь что-нибудь говорит...

Доктор Зигмунд: Видите? Это очень важно — у вас есть друзья, готовые с вами пообщаться.

Сервер: В этом-то все дело! Когда я что-нибудь отвечаю, меня не слушают. А в последнее время ситуация только ухудшилась.

Доктор Зигмунд: Расскажите мне про это... «последнее время».

Сервер: Раньше я много чего говорил — `<html><head><title>Hello!</title></head></html>`... А теперь, когда я говорю что-нибудь длиннее «1012», все начинают жаловаться. Мне разрешают произнести слово или два, вот и весь разговор.

Доктор Зигмунд: А у вас есть что сказать?

Сервер: Да! Иногда мне нужно сказать очень много всего. Я все время слушаю, никогда не игнорирую запросы... но когда наступает время ответить, и я не справляюсь за одну-две секунды, меня обзывают этими ужасными словами..

Доктор Зигмунд: Какими? Как вас обзывают?

Сервер: Медленный... Неповоротливый... На уровне прошлого века ... О, это ужасно! А я просто хочу, чтобы моя свобода слова не ограничивалась короткими высказываниями вроде «6» или «Pot 1»!

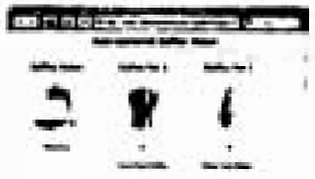
...продолжение следует...

Говорите!

Давайте припомним, что сервер говорил нашим Ajax-приложениям. Ниже перечислены пять приложений, созданных нами до настоящего момента. Вернитесь к соответствующим главам, разберитесь, какие данные возвращал сервер каждому приложению, и запишите ответы в приведенных пропусках.

Глава 1: Boards `R` Us

Глава 2: Break Neck Pizza



Глава 3: Ajax Coffee Maker

Глава 4: Top 5 CDs

Глава 5: Break Neck (новая версия)

Говорите!

Так что же отвечал сервер? Как насчет жалоб, что его никто не слушает? Насколько они обоснованы? Проверьте свои ответы и решите сами... Действительно ли серверу есть что сказать?

1143

Глава 1: Boards 'R' Us

Единственное приложение, в котором сервер возвращает заполненный объем данных

Doug Henderson
7804 Jumping Hill Lane
Dallas, Texas 75218

Глава 2: Break Neck Pizza

Ну-да. В этих приложениях сервер действительно многословен

2

Глава 3: Ajax Coffee Maker

Сервер не участвует!

Глава 4: Top 5 CDs

1

Напоминаем: это примерное время доставки пиццы

Глава 5: Break Neck (новая версия)

Доктор Зигмунд: Ну… Я вижу, что в главе 2 вам удалось высказаться. Целый адрес… несомненно, это было интересно!

Сервер: Да, но это только что-то одно. Я хочу сказать, что мне не разрешается вернуть, скажем, адрес клиента и специальный купон для повторных заказов. Мне разрешают вернуть только одно: адрес.

Доктор Зигмунд: Говорят, краткость — сестра таланта.

Сервер: Что? А… Кто говорит? Тому, кто это говорит, не приходится сидеть целыми днями, выслушивая длинные запросы.

Доктор Зигмунд: Спокойнее. А вы когда-нибудь *пытались* вернуть более одного информационного фрагмента в своем ответе?

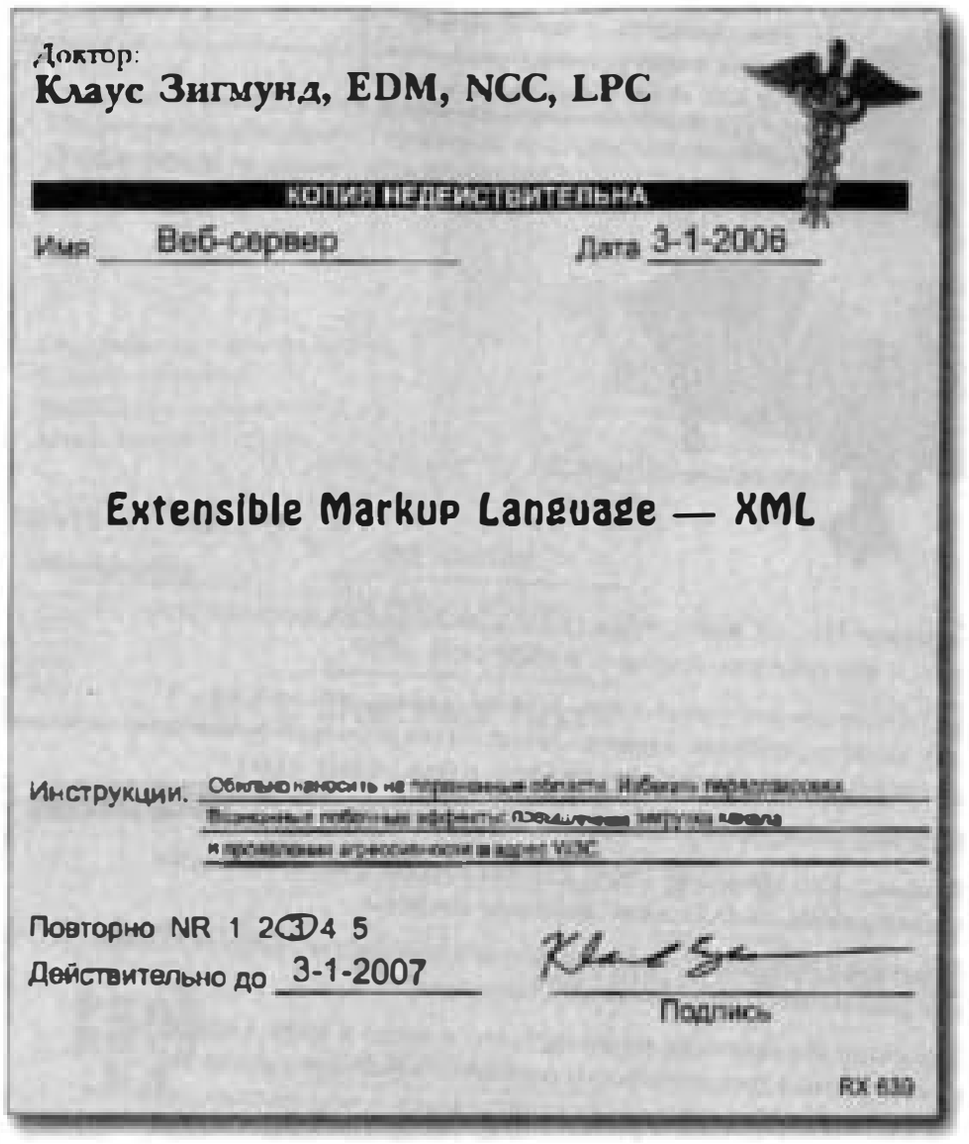
Сервер: Конечно! Но никто не понимает, что я имею в виду. Однажды я попытался вернуть адрес, телефон и описание заказа, разделив их символами «;». Однако это никому не понравилось… мне сказали, что это «специализированный формат», и мне нужно «стандартизироваться».

Доктор Зигмунд: Знаете, это правда. Обратитесь ко мне, когда вам захочется снова использовать запятые или эти подозрительные символы «|» для разделения данных.

Сервер: (*Это было всего один раз…*)

Доктор Зигмунд: Пожалуй, я знаю, что нужно делать, мой друг. Я вам выпишу рецепт. Уверяю, вам не повредит хорошая доза стандартизации. Все будут понимать, что именно вы говорите.

Сервер: И что эта… стандартизация мне даст? Я ведь просто хочу, чтобы меня понимали..



Рецепт, выписанный Серверу доктором Зигмундом

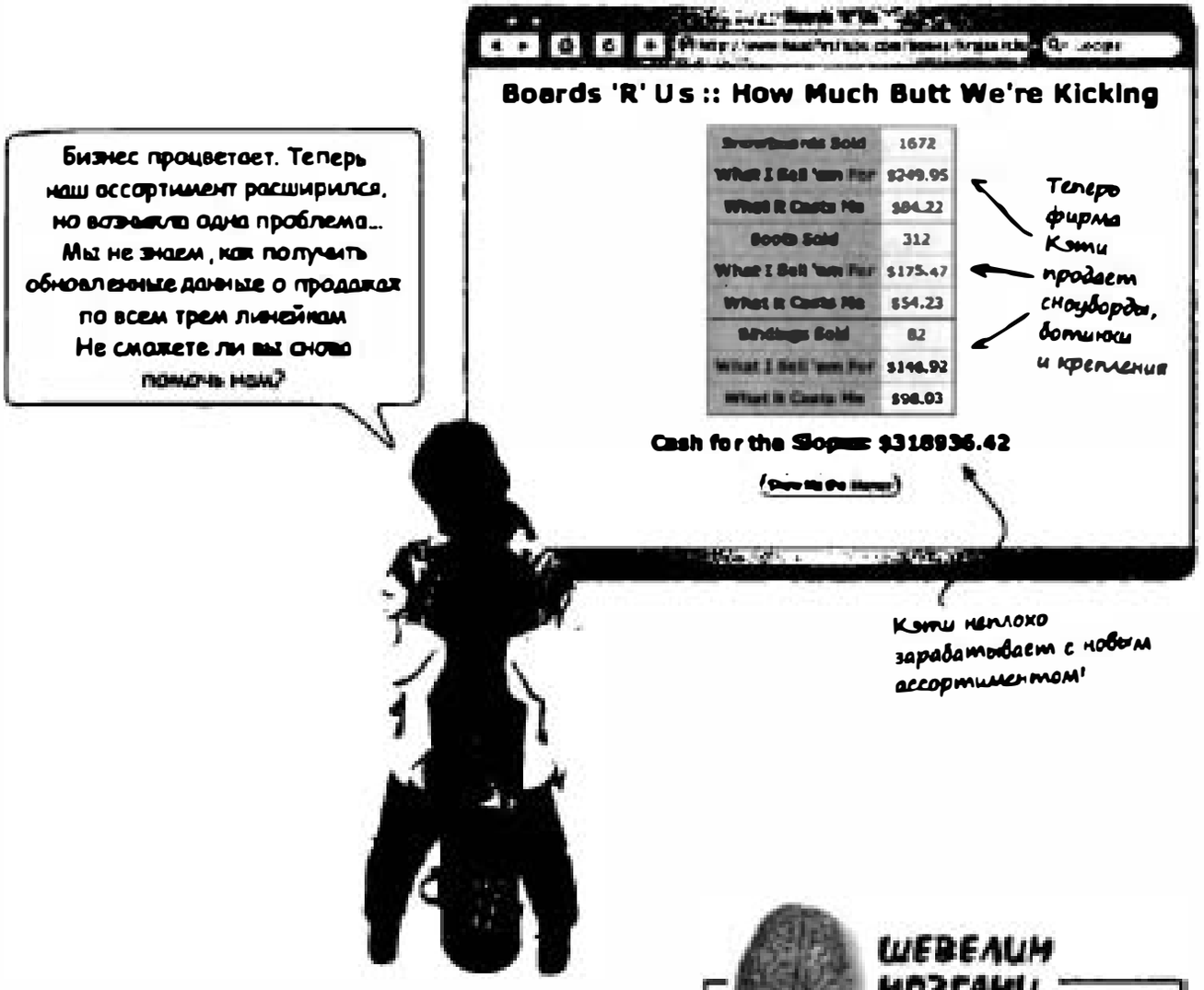
XML: то, что доктор прописал

Похоже, доктор Зигмунд достаточно четко себе представляет, как помочь веб-серверу в его желании передать больше полезной информации в ответе. Не забывайте об этом, потому что вскоре мы вернемся к XML.

А пока речь шла о помощи — похоже, в ней нуждается наша старая знакомая...

Помните Кэти?

У Кэти было много дел с того времени, когда мы расстались с ней в главе 1. Теперь у нее появились две новые линейки товаров — ботинки и крепления для сноуборда. Кэти обновила свой веб-отчет так, чтобы в нем отображалась информация по всем трем категориям.



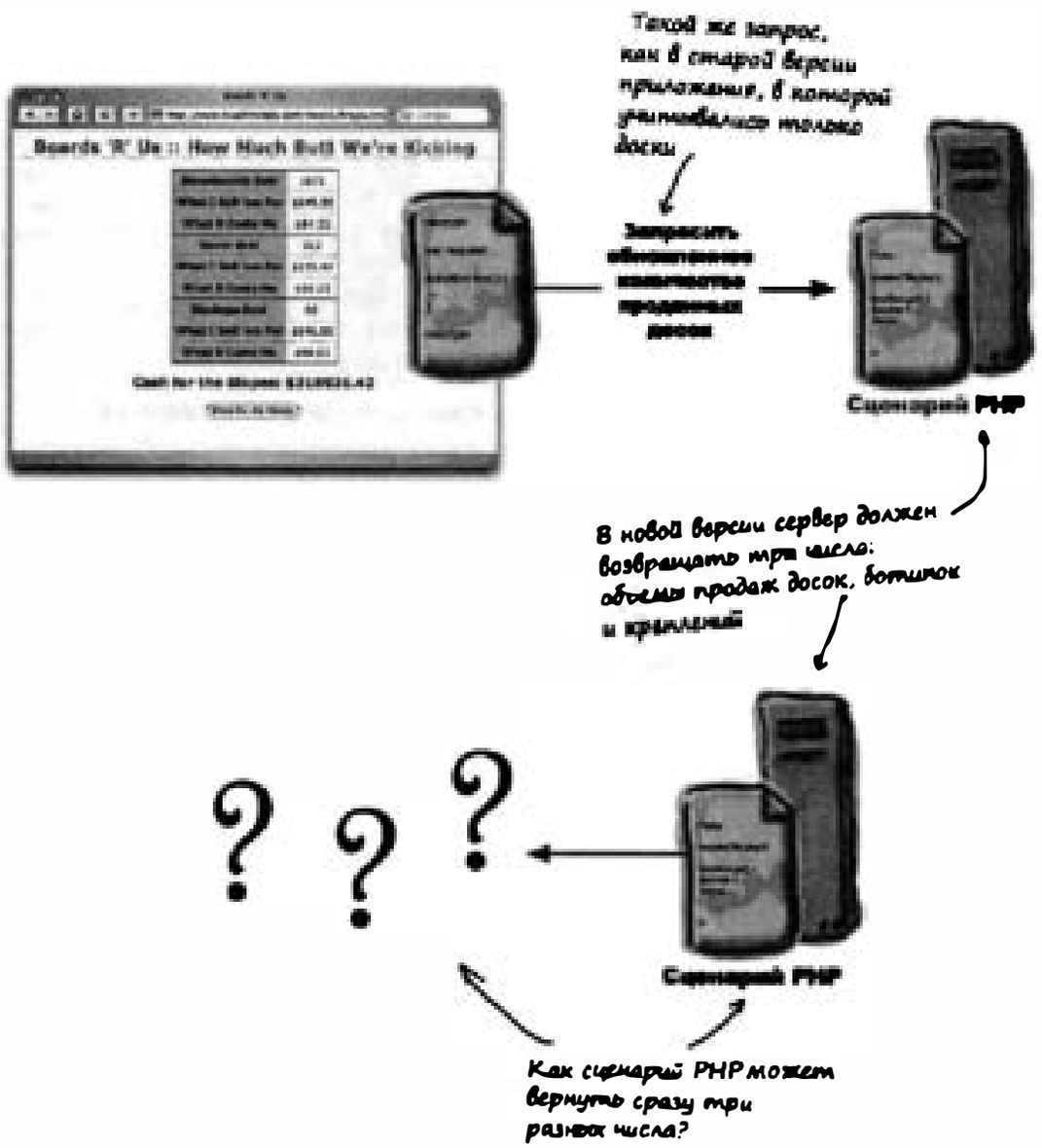
ШЕВЕЛИН НОЗГАНИ

Какую пользу XML принесет в приложение Boards R' Us?

В чем проблема?

Отчет Boards по-прежнему использует асинхронные запросы для взаимодействия с сервером. Проблема с кэшированием была решена, и отчет теперь работает на любых компьютерах (в том числе и на компьютере Кэти с системой Windows).

Но теперь сервер должен возвращать три числа: количества проданных досок, ботинок и креплений.



Доктор Зигмунд: Прогресс, мой дорогой друг! Теперь вас хотят слышать.

Сервер: Но как я передам сразу три числа? Это мой единственный шанс... нельзя его упускать.

Доктор Зигмунд: Вы не воспользовались моим рецептом, не так ли?

Сервер: ...Можно сцепить три числа, разделив их пробелами... Нет, это убогая идея...

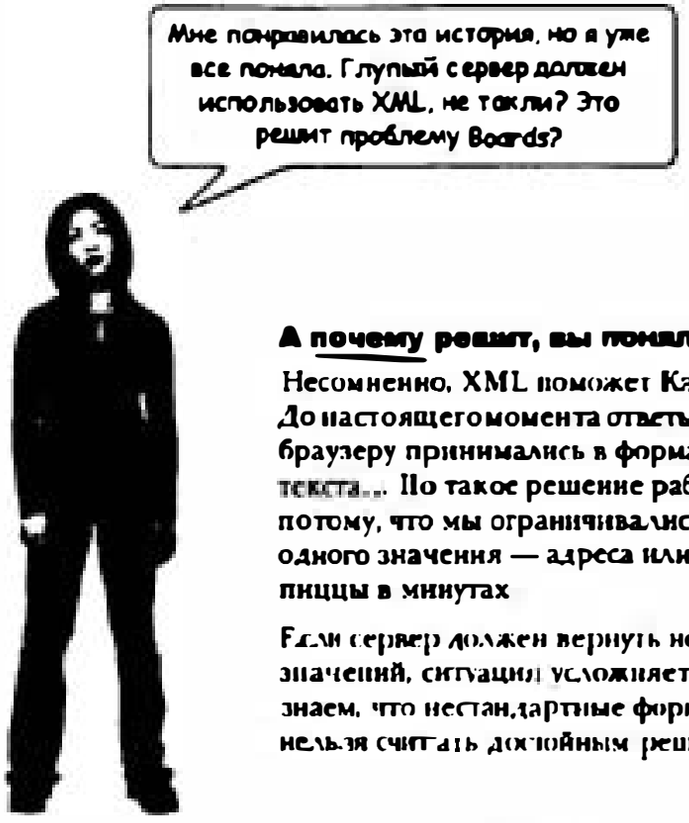
Доктор Зигмунд: Вы думали о применении XML?

Сервер: Может, разделить запятыми? В электронных таблицах работает... Нет, приложение может по ошибке принять запятую за часть самого значения...

Доктор Зигмунд: Спрашивается, кто кого не желает слушать? Молодой человек, я пытаюсь сказать вам...

Сервер: Если бы я мог передать данные в формате, понятном для отчета Boards... Кэти терпеть не может нестандартные решения, которые могут перестать работать через несколько месяцев...

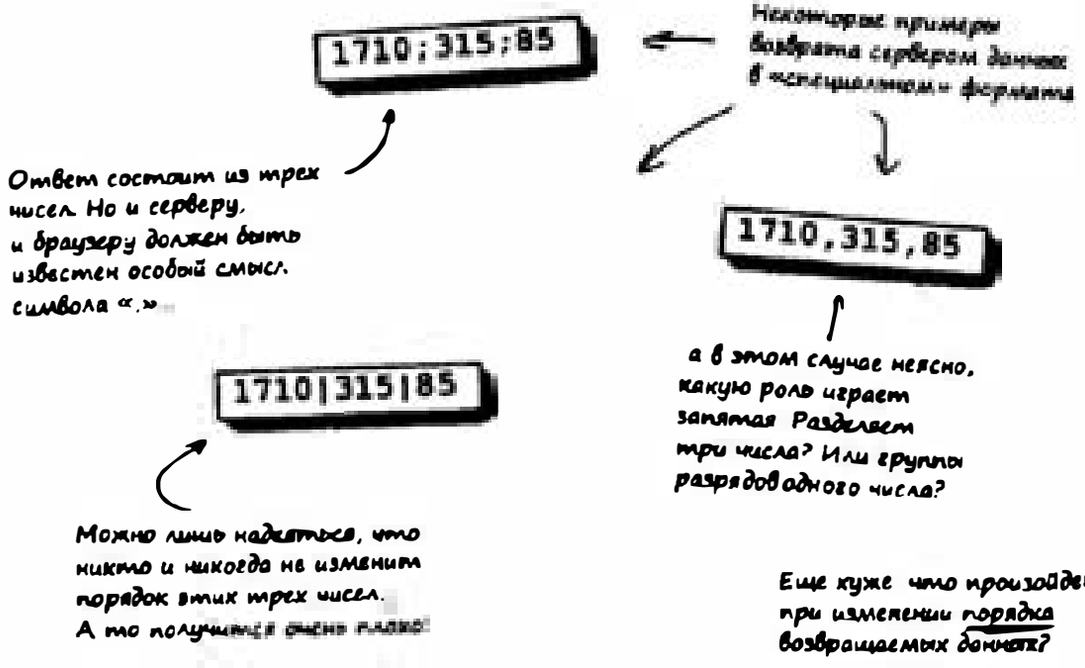
Доктор Зигмунд: Безнадежно.



А почему решит, вы поняли?

Несомненно, XML поможет Кэти с ее отчетом. До настоящего момента ответы сервера браузеру принимались в формате простого текста... Но такое решение работало лишь потому, что мы ограничивались возвратом одного значения — адреса или срока доставки пиццы в минутах.

Если сервер должен вернуть несколько значений, ситуация усложняется. Мы уже знаем, что нестандартные форматы данных нельзя считать достойным решением:



По рецепту XML

Изобрести формат данных, понятный как для сервера, так и для браузера... который не изменится, если Кэти введет новую линейку товаров, или переставит товары в своем отчете... не так просто.

К счастью для нас, доктор Зигмунд (и одна нетерпеливая девушка на предыдущей странице) предлагают решение: XML., расширяемый язык разметки (eXtensible Markup Language). XML. предоставляет в наше распоряжение простой и понятный формат ответа, содержащего все три обновленных числа.

Эта строка всего лишь указывает, что информация передается в формате XML. Из нее можно узнать используемую версию XML

<totals> — корневой элемент. В нашем примере он является контейнером для трех элементов «sold»

...и способ кодирования XML

```
<?xml version="1.0" encoding="utf-8"?>
<totals>
  <boards-sold>1710</boards-sold>
  <boots-sold>315</boots-sold>
  <bindings-sold>85</bindings-sold>
</totals>
```

Совершенно ясно, какое число представляет количество проданных досок, ботинок или креплений

Даже если переставить строки в другом порядке, все равно будет ясно, какое значение представляет то или иное число

Сервер возвращает в качестве ответа весь фрагмент в формате XML

Часто задаваемые вопросы

В: Чем так хорош формат XML?

О: Главная ценность XML заключается в том, что это общепризнанный стандарт. Консорциум World Wide Web Consortium (сокращенно W3C) определяет, что следует считать «правильным» XML. А поскольку большинство людей соглашаются соблюдать стандарты W3C, браузеры, серверы и программы (такие, как сценарии PHP) могут использовать XML, совершенно точно зная, как должна интерпретироваться угловая скобка или символ «точка с запятой»

В: Все равно не понимаю, чем плох [redacted] формат данных. Разве это не проще?

О: На первый взгляд проще, но [redacted] форматы данных (то есть форматы, которые вы изобретаете для собственного использования) способны создать массу проблем. Если не документировать такие форматы, люди забывают, как они работают. Что еще хуже, некоторые символы (такие, как «;» или «,») могут иметь сразу несколько значений, и ваш формат лишь запутает программистов

В: Я понимаю, чем удобен XML, но обязательно ли нам использовать те же имена элементов?

О: Вовсе нет. Одним из главных достоинств языка XML является его гибкость. Таким образом, вы можете использовать boardSales вместо board-sales, или totalBindings вместо bindingSales — все зависит от вас. Важно лишь то, чтобы код JavaScript в вашем браузере и код на сервере знали, какие имена должны использоваться в конкретном случае, а выбор имен остается за вами.

PHP...на первый взгляд

Пока мы думали над кодом JavaScript и HTML, серверные программисты не теряли времени даром. Они обновили сценарий getUpdatedSales.php так, чтобы он возвращал данные в формате XML, о котором мы говорили, с обновленными данными по продажам досок, ботинок и креплений. Неплохо, верно? Вот как теперь выглядит сценарий:

```
<?php

// Подключение к базе данных
$conn = @mysql_connect("mysql.headfirstlabs.com",
                       "secret", "really-secret");
if (!$conn)
  die("Error connecting to MySQL: " . mysql_error());

if (!mysql_select_db("headfirst", $conn))
  die("Error selecting Head First database: " . mysql_error());

$select = 'SELECT boardsSold, bootsSold, bindingsSold';
$from   = '  FROM boardsrus';
$queryResult = @mysql_query($select . $from);
if (!$queryResult)
  die('Error retrieving total boards sold from database.');

while ($row = mysql_fetch_array($queryResult)) {
  $boardsSold = $row['boardsSold'];
  $bootsSold = $row['bootsSold'];
  $bindingsSold = $row['bindingsSold'];
}

header("Content-Type: text/xml");
echo "<?xml version=\"1.0\" encoding=\"utf-8\"?>
?>

<totals>
 <boards-sold><? echo $boardsSold; ?></boards-sold>
 <boots-sold><? echo $bootsSold; ?></boots-sold>
 <bindings-sold><? echo $bindingsSold; ?></bindings-sold>
</totals>

<? mysql_close($conn); ?>
```

Большая часть сценария совпадает со старой версией, в которой был только один товар, и формат XML не использовался

В базе данных Boards теперь хранится информация по трем товарам: доскам, ботинкам и креплениям

В этой версии приложения нас интересуют три товара вместо одного

Так мы сообщаем браузеру, что сценарий возвращает XML, а не разметку HTML или текст

Первая часть строки <? является открывающим тегом в PHP, поэтому мы должны вывести ее командой «echo» отдельно от прочих байтов XML

Теперь PHP вместо одного числа или разметки HTML возвращает код XML

Каждый элемент XML содержит данные о продажах соответствующего товара из запроса к базе данных Boards

Просто сделайте это

Довольно разговоров; пора браться за дело. Откройте папку chapter08/boards в архиве примеров, загруженном с сайта. Вы найдете в ней файлы приложения Boards 'R' Us, включая обновленный отчет HTML (boards.html), несколько файлов JavaScript (ajax.js и boards.js), а также XML-версию getUpdatedSales.php. Ваша задача — обновить код отчета таким образом, чтобы он получал ответы XML от getUpdatedSales.php.

Откройте файл boards.js и найдите функцию, получающую ответ сервера. Временно закомментируйте весь код DOM, обновляющий форму... через несколько страниц мы все исправим. А пока мы должны проверить состояние готовности запроса, убедиться в том, что код статуса равен 200 (с ответом все в порядке), а затем вывести ответ сервера в окне сообщения JavaScript. Когда все будет сделано, загрузите boards.html и щелкните на кнопке Show Me the Money — результат должен выглядеть примерно так:

В окне сообщения выводится ответ сервера, который выглядит состоящим из кода XML.

СТОП! Не переворачивайте страницу, пока не выполните это упражнение.

Просто сделайте это

Вы разобрались, как вывести ответ сервера? Вот что сделали мы:

1. Мы открыли файл **boards.js**, в котором хранится функция обратного вызова Boards.
2. Мы нашли функцию обратного вызова **updatePage()** и закомментировали весь код, кроме двух команд с проверкой состояния готовности и кода статуса из запроса.
3. Мы получили ответ сервера и вывели его в окне сообщения при помощи функции **JavaScript alert()**.

Код нашей функции **updatePage()**:

```
function updatePage() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      var response = request.responseText;
      alert(response);
/*
      var newTotal = request.respor.seText;
      ...
      replaceText(cashEl, cash);
*/
    } else {
      // Код обработки ошибок
  }
}
```

Оставьте код проверки состояния готовности и статуса — он нам понадобится

Эта команда получает ответ от сервера и отображает его в диалоговом окне

Здесь должно быть довольно много кода... временно закомментируйте его

Более часто задаваемые вопросы

В: И это все? Сервер возвращает код XML, а мы читаем его из свойства responseText?

О: На простейшем уровне — все. Но как вы вскоре увидите, с XML можно сделать нечто большее, чем рассматривать его как обычный текст. Оставайтесь с нами.

В: Как извлечь данные из XML? Помощь, это непросто

О: Да, вы правы. Анализ кода XML (этот процесс называется разбором) и извлечение из него данных — задача нетривиальная. К счастью, существует более удобный способ работы с XML, чем на уровне простого текста.

Майк: Не куст... Дерево. И не простое дерево; это дерево DOM, которое вам пригодится.

Сервер: Вы не видите, что у меня сеанс психотерапии? Я не интересуюсь ландшафтными работами.

Майк: Я совершенно случайно услышал, что вы теперь возвращаете код XML в своих ответах.

Сервер: Да, верно... У меня трудности с общением, а Зигги считает, что XML поможет мне передать браузерам больше информации, и при этом быть понятым

Майк: По-моему, хороший совет. Но как браузер будет работать с вашим кодом XML? Разобрать документ XML непросто, знаете ли.

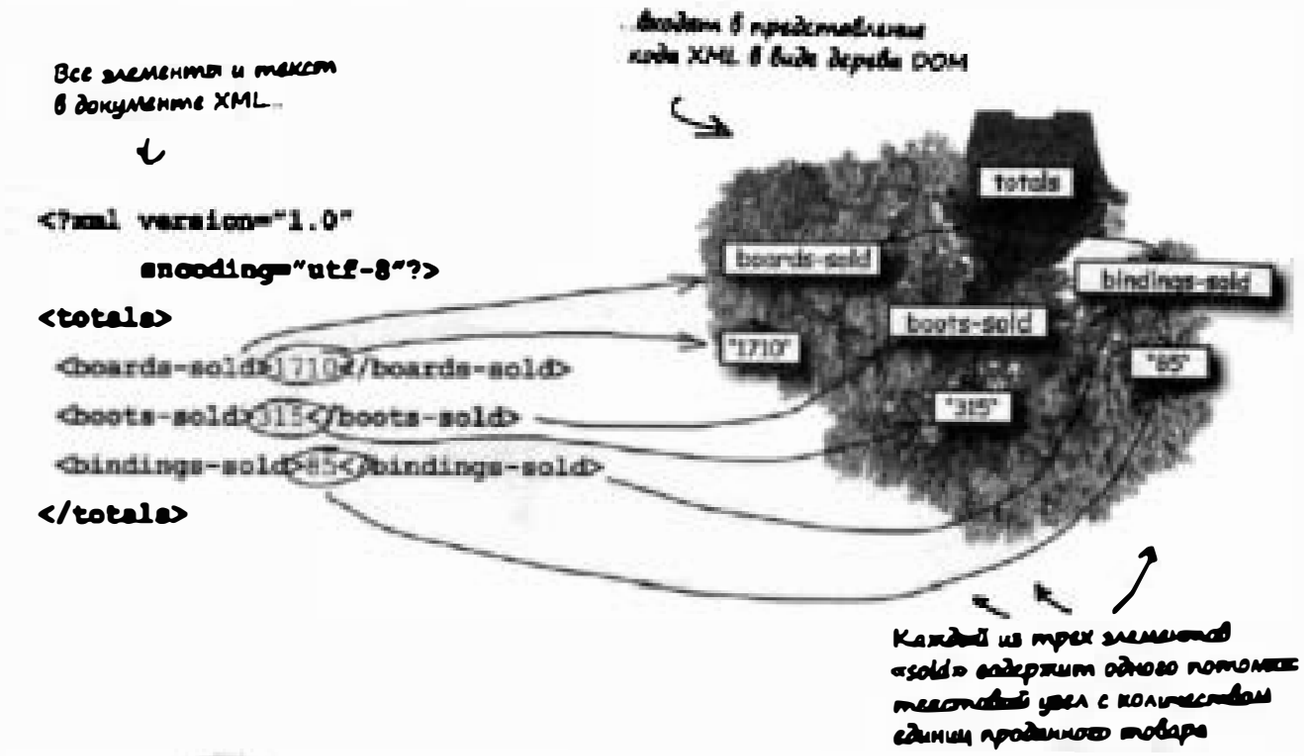
Сервер: Что? Вы серьезно? Столько работы, и меня все равно не поймут? Дайте веревку и мыло, с меня хватит!

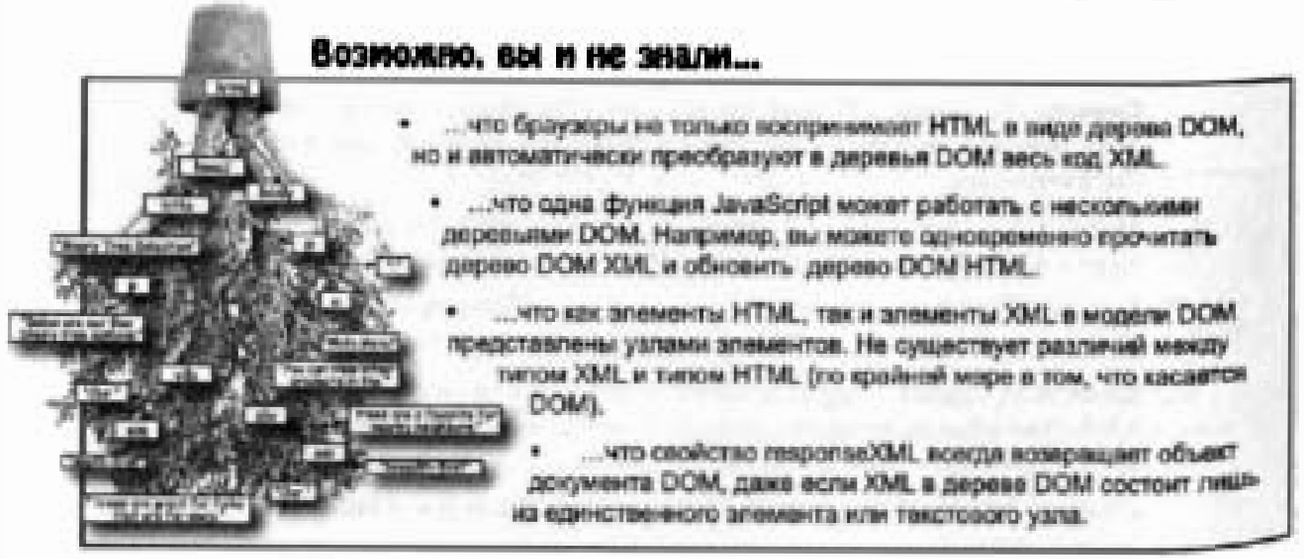
Майк: Вот почему я здесь! Браузер — и JavaScript — понимают формат XML. Любой код, которому потребуется прочитать ответ в формате XML, может работать с ним средствами DOM.

Сервер: Да! Давайте вырастим дерево DOM. Кажется, я вижу свет в конце туннеля...

Деревья, деревья, куда ни глянь

Мы уже встречались с моделью Document Object Model, упрощающей работу с HTML. Однако модель DOM более универсальна: она также позволяет работать с документами XML. Давайте посмотрим, как код XML, возвращаемый getProductSales.php, выглядит в виде дерева DOM:



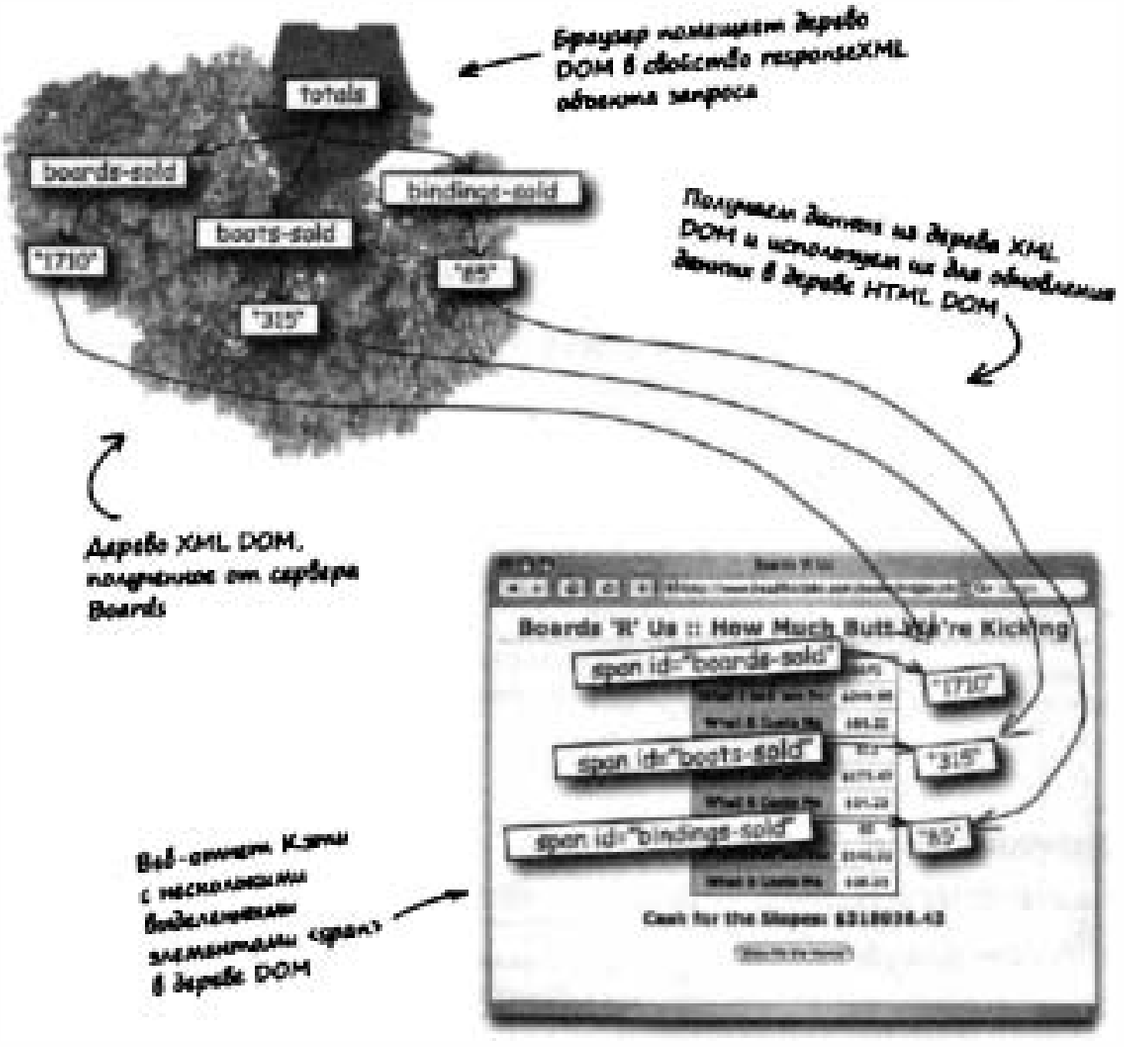


Использование responseXML в вашем коде

Мы уже видели, что свойство **responseText** объекта запроса позволяет прочитать код XML, полученный от сервера. Но нас не устроит текст в формате XML, ведь мы теперь стали экспертами по DOM, не так ли? При помощи свойства **responseXML** мы можем получить дерево DOM для ответа сервера, а затем работать с XML средствами DOM.

Даже если вы не чувствуете себя экспертом, на самом деле вы совсем неплохо разбираетесь в DOM. А к последним страницам этой главы вы будете знать DOM еще лучше, чем сейчас

А поскольку код HTML отчета Кэти представляет другое дерево DOM, нам остается лишь взять данные из XML DOM и поместить их в HTML DOM. Посмотрим, что же именно необходимо сделать.



Элементы можно найти по «имени тега»

Ранее мы использовали функцию **getElementById()** для поиска элементов в дереве DOM. Однако существует и другой полезный метод, который называется **getElementsByTagName()**. Он позволяет найти все элементы дерева DOM с определенным именем:

Получите дерево DOM, представляющее ответ сервера в формате XML

```
var xmlDoc = request.responseXML;
var boardsSoldElements =
  xmlDoc.getElementsByTagName("boards-sold");
```

Теперь мы работаем с DOM!

Приведенный фрагмент возвращает массив всех элементов с именем «boards-sold» в дереве DOM xmldoc. Далее для обращения к элементам списка используется простое индексирование, как в следующем примере:

При необходимости содержимое списка также можно перебрать в цикле for

```
var firstBoardsSoldElement =
  boardsSoldElements[0];
```

Помните: индексы в массивах JavaScript начинаются с 0, а не с 1

Сокращенная запись:

```
var firstBoardsSoldElement = xmlDoc.getElementsByTagName("boards-sold")[0];
```

Массив всех элементов с именем "boards-sold".

...первый элемент массива

Часто задаваемые вопросы

В: Я так и не понимаю, какое отношение к этому имеет responseXML.

О: Браузер сохраняет текстовую версию ответа сервера в свойстве responseText объекта запроса — будь то отдельное значение, пары «имя/значение» или текстовая версия документа XML.

Но если сервер возвращает код XML и задает заголовок ответа Content-Type равным «text/xml», браузер создает дерево DOM для представления кода XML и помещает ссылку на него в свойство responseXML.

В: Значит, если я хочу использовать DOM для работы с XML, вместо responseText нужно использовать responseXML?

О: Совершенно верно. Вообще говоря, работать с XML можно и через responseText, но тогда придется разбирать XML самостоятельно... вам это нужно?

В: В HTML DOM на с. 351 атрибуты id показаны как составная часть элементов <head>. Я думал, что в DOM атрибуты представляются отдельно от элементов.

О: Вы настолько внимательно следите за деревьями DOM? Отличная работа... Вы абсолютно правы. Атрибуты id элементов <head> в действительности представлены отдельными узлами дерева HTML DOM. Но мы немного нарушили правила и представили атрибут id как часть имени элемента — только для того, чтобы читателю стало чуть проще разобраться в происходящем, пусть даже это не совсем соответствует правилам DOM.

В: Значит, не существует различий между деревом HTML DOM и деревом XML DOM?

О: Дерево DOM веб-страницы представляет код HTML, а дерево DOM XML представляет ответ сервера. Впрочем, с деревьями DOM вы работаете точно так же, они обладают теми же методами, и оба дерева можно легко изменять. Удобно, верно?

Пора применить на практике полученные знания. Далее приведена функция обратного вызова `updatePage()` для приложения Boards. Для завершения кода вам понадобится все, что вы узнали об асинхронных запросах, DOM, XML и динамическом коде HTML.

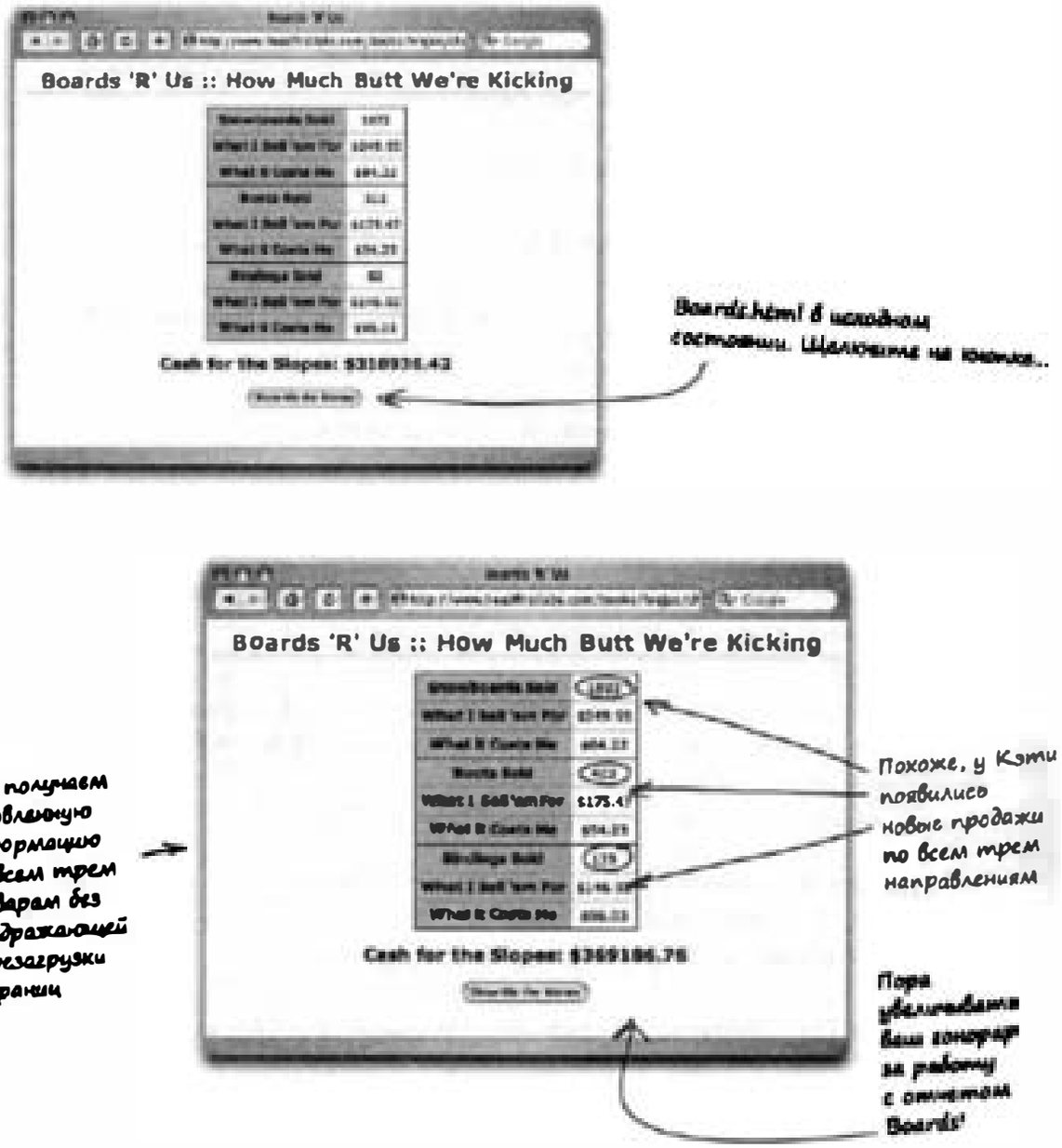
```
function updatePage() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // Получение обновленных данных из ответа XML
      var xmlDoc = request._____________;
      var xmlBoards =
        _____._____________________("boards-sold")[0];
      var totalBoards = xmlBoards.firstChild.nodeValue;
      var xmlBoots =
        _____._____________________("boots-sold")[0];
      var totalBoots = xmlBoots.firstChild.nodeValue;
      var xmlBindings =
        _____._____________________("_____")___;
      var totalBindings = xmlBindings.firstChild.nodeValue;

      // Обновление страницы
      var bo rd SoldEl =
        document._____________("boards-sold");
      var bootsSoldEl =
        document._____________("boots-sold");
      var bindingsSoldEl =
        document._____________("bindings-sold");
      var cashEl = document._____________("cash");
      replaceText(_____________, _____________);
      replaceText(_____________, _____________);
      replaceText(_____________, _____________);

      // Вычисление прибыли для досок
      var boardsPriceEl =
        do ument.getElementById("_____________");
      var boardsPrice = getText(boardsPriceEl);
```

```
var boardsCostEl =
  document.getElementById("_____");
var boardsCost = getText(boardsCostEl);
var cashPerBoard = boardsPrice - boardsCost;
var cash = cashPerBoard * totalBoards;

// Вычисление прибыли для ботинок
var bootsPriceEl =
  _____.getElementById("boots-price");
var bootsPrice = getText(_____);
var bootsCostEl = _____.getElementById("boots-cost");
var bootsCost = getText(_____);
var cashPerBoot = _____ - _____;
cash = _____ + (cashPerBoot * totalBoots);

// Вычисление прибыли для креплений
_____
_____
_____
_____
_____
_____
_____
_____

// Обновление суммарной прибыли на веб-форме
cash = Math.round(cash * 100) / 100;
replaceText(cashEl, cash);
} else
alert("Error! Request status is " + request.status);
```

Этот часть бы должна заполняться симметрично. Впрочем, если бы были вычислены, проблем не будет.

Пробный запуск

Готовы опробовать приложение Boards в деле? Выполните упражнение на предыдущих страницах и внесите изменения в свою копию boards.js. Если вы не уверены в правильности ответов, сравните их с нашими на с. 366. Сохраните изменения, запустите браузер и загрузите файл boards.html.



Доктор Зигмунд: Превосходно. Значит, теперь вы поняли, что к вам прислушиваются?

Сервер: О, да. И я должен признать... Тот парень с деревьями мне действительно помог.

Доктор Зигмунд: Расскажите об этом подробнее.

Сервер: На первых порах я возвращал XML — и это было хорошо. Никто больше не просил у меня данных, разделенных запятыми, или чего-нибудь в этом роде, а браузеры понимали, что я им говорю. Но похоже, у людей возникали проблемы с использованием того, о чем я говорил.

Доктор Зигмунд: Вот как? Почему вы так думаете?

Сервер: Оказалось, что они работают с текстовой версией моего ответа XML, и пытаются самостоятельно организовать разбор XML. Представляю, как они мучились! И конечно, у многих людей возникали ошибки.

Доктор Зигмунд: И как вы вышли из положения?

Сервер: Ну, сначала я принял успокоительное (большое спасибо, все говорят, что я стал гораздо спокойнее), а потом предложил работать с представлением XML в виде дерева DOM. Браузер даже предоставил доступ к этому представлению через простое свойство запроса.

Доктор Зигмунд: Ага, понимаю — дерево DOM, о котором рассказывал человек. Ну конечно!

Сервер: Да. Говорю вам, жизнь стала просто замечательной. Я наконец-то могу говорить действительно важные вещи, и никто не обрывает меня на полуслове. Наконец, мне не приходится возвращать кучу лишнего кода HTML. Я здоров, доктор, я абсолютно здоров!

Вы продолжаете утверждать, что XML лучше, потому что это открытый формат, который стал «стандартным». Но ведь в коде XML приложения Boards используются такие элементы, как <boardsSold> и <bootsSold>. Какой же это стандарт? Кто еще будет эти имена использовать?

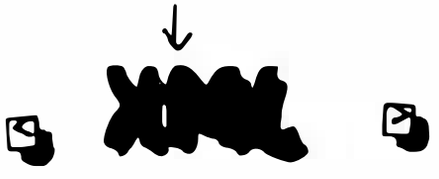
XML стандартизирован...
...а способы его использования — нет.

Спецификация XML разрабатывается консорциумом W3C (World Wide Web Consortium). Это означает, что если вы передаете данные в формате XML, все остальные будут знать, как с ними работать. Однако XML в действительности представляет собой метаязык, то есть язык для определения других языков. Так, в приложении Boards мы использовали XML, но определили собственные элементы с именами, соответствующими специфике приложения Boards.

Если вы захотите обновить сервер Break Neck Pizza так, чтобы он возвращал код XML, вероятно, в нем будут использоваться элементы с другими именами — такими как `<deliveryTime>` и `<orderConfirmation>`. В коде сервера и коде JavaScript, который вы напишете, должны использоваться одинаковые имена, иначе ничего работать не будет.

XML является метаязыком, то есть используется для определения других языков.

Это язык XML. Он определяет, что такое элемент, что такое атрибут, и как должны использоваться некоторые символы (например, угловые скобки < и >). XML является [redacted]



Все равно XML лучше какого-нибудь формата, придуманного для конкретной задачи. Ведь для работы с XML всегда существуют готовые инструменты — такие как DOM, верно?

Иногда формат XML очень удобен...
...иногда нет.

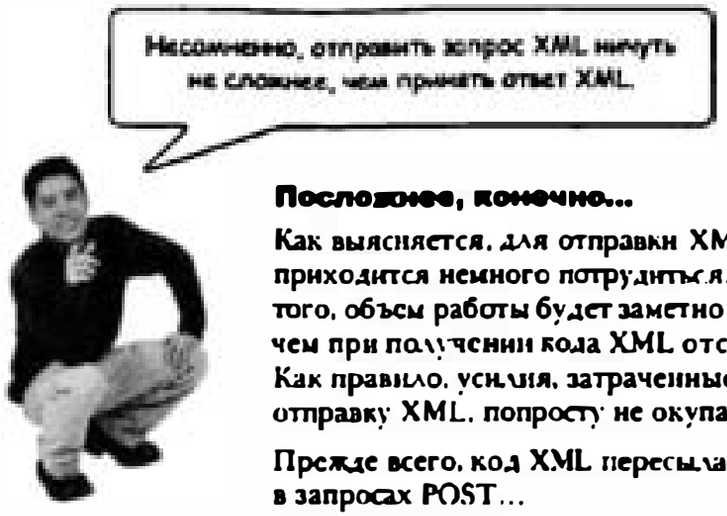
Формат данных XML весьма популярен, и такие инструменты, как DOM, присутствуют в любом языке программирования. При использовании модели DOM вам не придется беспокоиться о порядке элементов в документе XML или писать собственный код разбора. XML отлично подходит для представления данных.

Однако у XML имеется и оборотная сторона: для передачи ничтожного объема данных используется большой объем текста. Все эти имена в угловых скобках занимают много места! И как вы узнаете в следующей главе, существует немало достойных альтернатив — таких как JSON.

Документы XML. Имена элементов и атрибутов используются в соответствии со стандартом XML, но вы можете определить собственные имена элементов и атрибутов, и создать собственную версию XML специально для своего приложения.

```
<?xml version="1.0" encoding="utf-8"?>
<totals>
  <boards-sold>1710</boards-sold>
  <boots-sold>315</boots-sold>
  <bindings-sold>85</bindings-sold>
</totals>
```

Не пытайтесь применять XML для любых целей!



Пословное, конечно...

Как выясняется, для отправки XML приходится немного потрудиться... Более того, объем работы будет заметно больше, чем при получении кода XML от сервера. Как правило, усилия, затраченные на отправку XML, попросту не окупаются.

Прежде всего, код XML пересылается только в запросах POST...

Даже короткий ответ XML от сервера Boards занимает около 200 символов!

Все верно.

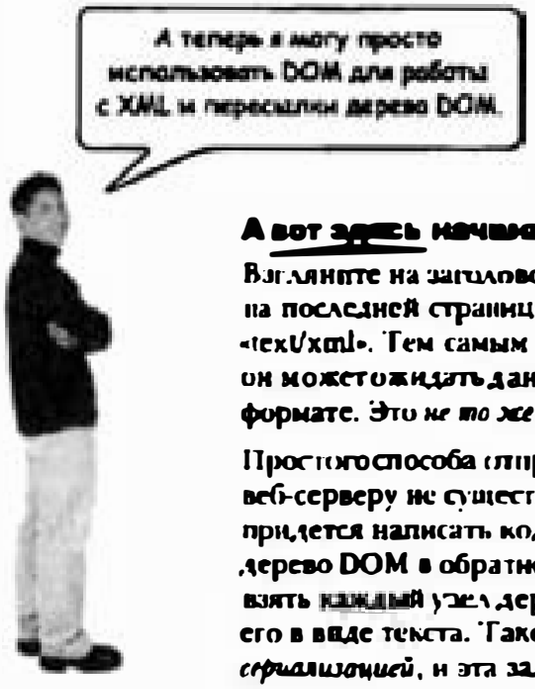
Даже при небольшом объеме данных документ XML получается довольно длинным. Вы обязаны использовать запрос POST. Кроме того, серверу нужно сообщить, что передается код XML, а не пары «имя/значение»

И снова в точку!

Достаточно вызвать метод **setRequestHeader()** объекта запроса и сообщить серверу о том, что ему передается код XML:

```
request.setRequestHeader("Content-Type", "text/xml");
```

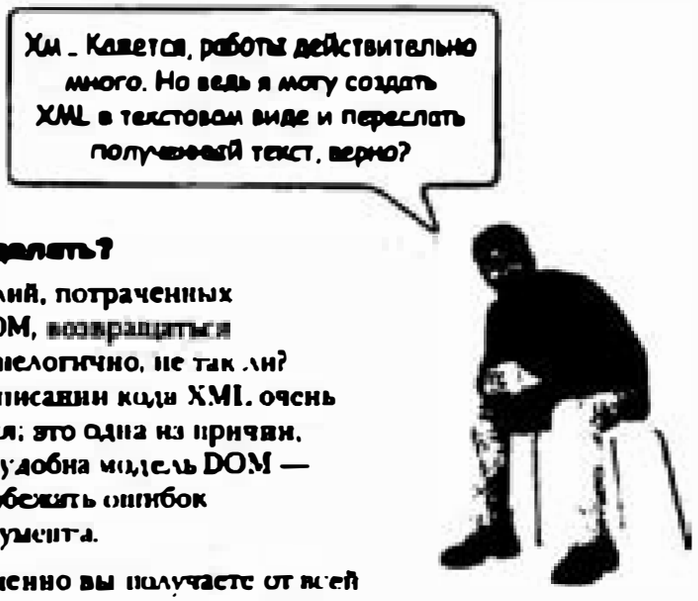
Мы сообщаем серверу, что во входном сообщении передается код XML в текстовом формате



А вот здесь начинаются проблемы!

Взгляните на заголовок запроса «Content-Type» на последней странице... он содержит значение «text/xml». Тем самым мы сообщаем серверу, что он может ожидать данные XML, но в текстовом формате. Это *не то же самое*, что дерево DOM.

Простого способа отправки дерева DOM веб-серверу не существует. Фактически вам придется написать код, который будет разбирать дерево DOM в обратном направлении: он должен взять каждый узел дерева DOM и записать его в виде текста. Такой процесс называется *сериализацией*, и эта задача не из простых.



А зачем это делать?

После всех усилий, потраченных на изучение DOM, возвращаться к тексту как-то нелогично, не так ли? При ручном написании кода XML очень легко ошибиться; это одна из причин, по которой так удобна модель DOM — она помогает избежать ошибок в структуре документа.

И потом, что именно вы получаете от всей дополнительной работы, связанной с отправкой серверу XML вместо пар «имя/значение» в формате простого текста?

Сервер: Я наконец-то нашел способ общаться... быть услышанным... понятым...

Доктор Зигмунд: Да, да, XML... Он больше не работает?

Сервер: Нет, все отлично... Я ОБОЖАЮ XML. Но теперь все пытаются присылать мне данные в формате XML.

Доктор Зигмунд: И с этим что-то не так?

Сервер: Конечно! Чем был плох текст? Что случилось с временами простых пар «имя/значение»? Теперь каждый норовит прислать мне длинный документ XML, но передает такую же информацию, как при использовании простых пар «имя/значение».

Доктор Зигмунд: И как вы себя чувствуете?

Сервер: Как чувствую? Растолстевшим, медленным и неэффективным! Теперь мне приходится разбирать код XML только для того, чтобы извлечь из него данные, которые я раньше мог извлечь мгновенно. И еще нужно следить за тем, чтобы не было ошибок. Крошечная ошибка в XML — и все идет наперекосяк. Браузер начинает ругаться, и... Нет, и этого просто не переживу!

Доктор Зигмунд: И вы хотите отказаться от XML?

Сервер: Нет, нет! Мне нравится возвращать много информации в ответах на запросы, и XML позволяет мне это сделать. Просто я не понимаю, почему все думают, что они должны передавать XML мне. Это же нелогично...

Доктор Зигмунд: Похоже, вы считаете, что лучше присылать вам простой текст и ограничить применение XML возвращаемыми данными. Да?

Сервер: Точно! Это так просто... Почему, ну почему?

Часто задаваемые вопросы

В: Если мы хотим, чтобы сервер возвращал в своем ответе код XML, разве мы не должны использовать XML в своих запросах?

О: Вовсе нет. Формат, выбранный вами для запроса, не имеет никакого отношения к формату, используемому сервером в своем ответе. Вы можете использовать простой текст в запросе и ответе, текст в запросе и XML в ответе, XML в обоих случаях, и даже XML в запросе и текст в ответе.

В: Что необходимо сделать при отправке XML — только задать заголовок запроса?

О: При отправке запроса в формате XML необходимо использовать запрос POST, задать заголовку «Content-Type» запроса значение «text/xml» при помощи функции setRequestHeader(), а затем отправить XML в текстовом виде, вызывая метод send() для объекта запроса. Помимо этих специальных шагов, запросы отправляются точно так же, как обычные текстовые запросы.

В: Все равно не понимаю, почему я не могу воспользоваться DOM и отправить серверу дерево DOM.

О: Вы можете использовать DOM для построения запроса XML, но документ DOM нельзя отправить серверу напрямую. Необходимо создать текстовую версию DOM, а затем отправить ее вместо объектов DOM.

В: Почему? Разве не проще отправить DOM напрямую, если работа с XML в текстовой форме так плоха?

О: Конечно, передача объектов по сети сопряжена с некоторыми сложностями, особенно при использовании модели запросов Ajax. Кроме того, отправка XML вместо простых пар «имя/значение» не дает особых преимуществ. Таким образом, XML лишь усложняет запросы, а не упрощает их.

В: Я читал об инструментарии JavaScript, который преобразует дерево DOM в текстовый документ XML. Разве я не могу воспользоваться им и отправить созданный им код XML на сервер?

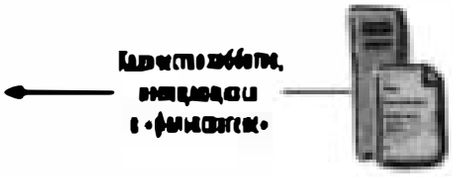
О: Конечно, в таком решении нет ничего плохого. Но учтите, что использование XML в запросе не дает никаких реальных преимуществ. Вы можете пересылать пары «имя/значение», причем этот формат является стандартом, поддерживаемым любым сервером. Следовательно, хотя вы можете использовать XML, вероятно, делать этого не стоит.

В: Неужели мне никогда не придется отправлять запросы в XML?

О: Существует только одна ситуация, в которой отправка запросов XML оправдана: если сервер, с которым вы общаетесь, принимает только код XML. Допустим, вы хотите обратиться к веб-серверу, принимающему только запросы SOAP (особая разновидность XML). Но если не считать этих особых случаев, для отправки запросов в почти всегда лучше использовать пары «имя/значение».

Выбираем формат данных

Решите, какой формат данных лучше подходит в каждом из пяти приведенных примеров. Будьте внимательны: в одних случаях речь идет о запросах, в других — об ответах. Удачи!

	Текст	XML
	☐	☐
	☐	☐
	☐	☐
	☐	☐
	☐	☐

Обзор на 60 секунд

- XML, расширяемый язык разметки, позволяет структурировать данные с использованием элементов и атрибутов.
- Многие сценарии и программы, работающие на стороне сервера, могут создавать ответы в формате XML и возвращать их в ответ на запросы браузера.
- XML позволяет включить в ответ сервера несколько фрагментов информации, без применения специализированных форматов данных или особого форматирования.
- Свойство responseText объекта запроса возвращает текстовую версию любого документа XML, возвращенного сервером.
- Чтобы получить DOM-представление документа XML, возвращенного сервером, используйте свойство responseXML объекта запроса.
- Если сервер не устанавливает заголовок ответа «Content-Type» равным «text/xml», многие браузеры неправильно задают значение свойства responseXML объекта запроса.
- Получив от сервера ответ в формате XML, браузер создает дерево DOM, представляющее документ.
- Возможна как отправка, так и получение документов XML, хотя для отправки DOM-представления документа XML потребуется специальный инструментарий или программный код.
- Создание XML вручную (на уровне обычного текста) — процесс трудоемкий и сопряженный с высоким риском ошибок.
- Для большинства запросов оптимальным решением является отправка пар «имя/значение» в виде простого текста. В частности, она снижает вычислительную нагрузку для кода JavaScript и веб-сервера, которому отправляется запрос.

Просто сделайте это

Пора применить на практике полученные знания. Далее приведена функция обратного вызова `updatePage()` для приложения Boards. Мы заполнили все пропуски; сравните свои ответы с нашими и убедитесь в том, что вы понимаете весь код.

```
function updatePage() {
  if (request.readyState == 4) {
    if (request.status == 200) {
      // Получение обновленных данных из ответа XML
      var xmlDoc = request.responseXML;
      var xmlBoards =
        xmlDoc.getElementsByTagName("boards-sold")[0];
      var totalBo rds = xmlBoards.firstChild.nodeValue;
      var xmlBoots =
        xmlDoc.getElementsByTagName("boots-sold")[0];
      var totalBoots = xmlBoots.firstChild.nodeValue;
      var xmlBindings =
        xmlDoc.getElementsByTagName("bindings-sold")[0];
      var totalBindings = xmlBindings.firstChild.nodeValue;

      // Обновление страницы
      var boardsSoldEl =
        document.getElementById("boards-sold");
      var bootsSoldEl =
        document.getElementById("boots-sold");
      var bindingsSoldEl =
        document.getElementById("bindings-sold");
      var cashEl = document.getElementById("cash");
      replaceText(boardsSoldEl, totalBoards);
      replaceText(bootsSoldEl, totalBoots);
      replaceText(bindingsSoldEl, totalBindings);

      // Вычисление прибыли для досок
      var boardsPriceEl =
        document.getElementById("boards-price");
      var boardsPrice = getText(boardsPriceEl);
```

Эти три строки могут следовать в произвольном порядке. Важно лишь то, чтобы имя элемента было связано с его значением

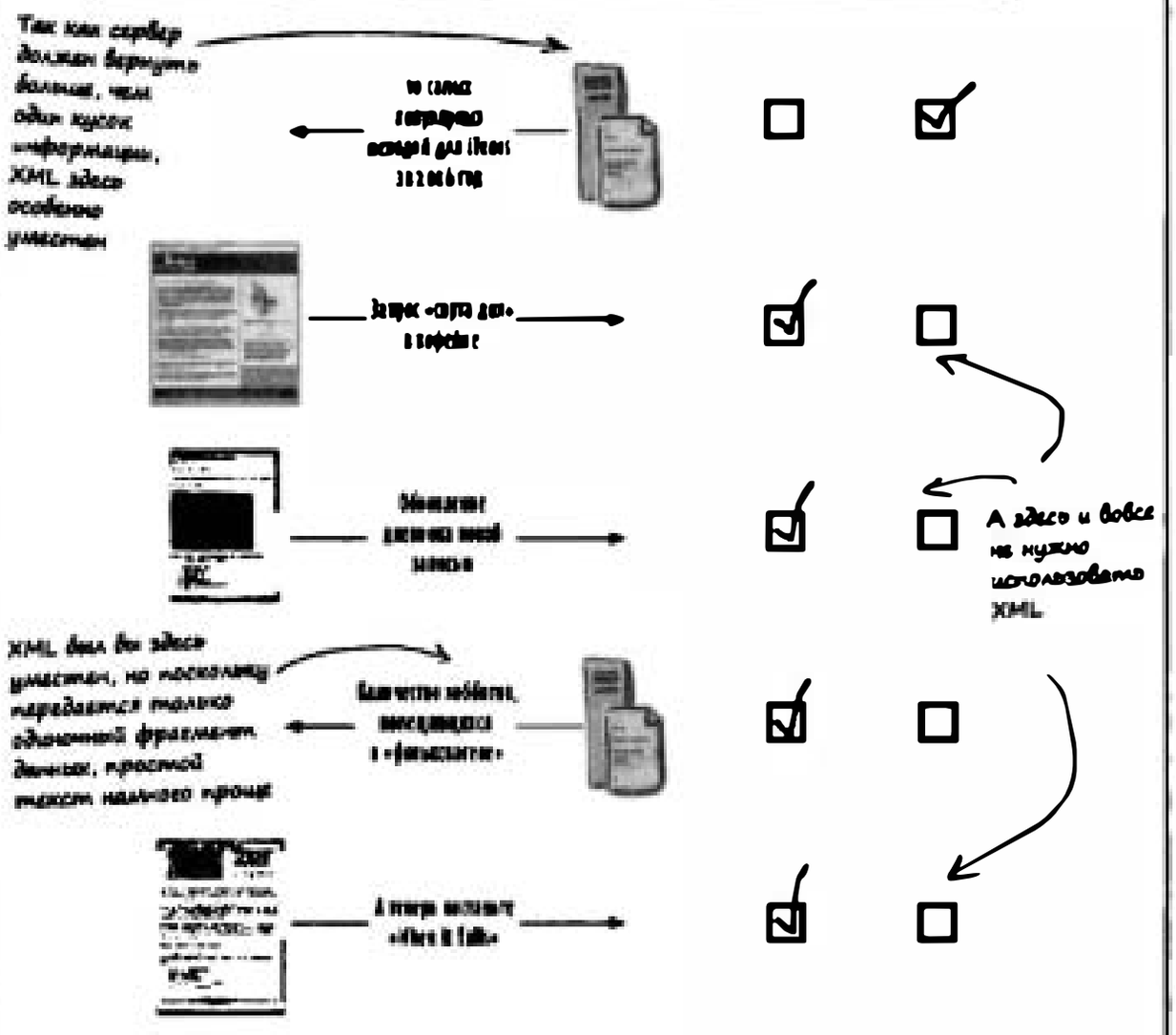
```
    var boardsCostEl =
      document.getElementById("boards-cost");
    var boardsCost = getText(boardsCostEl);
    var cashPerBoard = boardsPrice - boardsCost;
    var cash = cashPerBoard * totalBoards;

    // Вычисление прибыли для ботинок
    var bootsPriceEl =
      document.getElementById("boots-price");
    var bootsPrice = getText(bootsPriceEl);
    var bootsCostEl = document.getElementById("boots-cost");
    var bootsCost = getText(bootsCostEl);
    var cashPerBoot = bootsPrice - bootsCost;
    cash = cash + (cashPerBoot * totalBoots);

    // Вычисление прибыли для креплений
    var bindingsPriceEl =
      document.getElementById("bindings-price");
    var bindingsPrice = getText(bindingsPriceEl);
    var bindingsCostEl =
      document.getElementById("bindings-cost");
    var bindingsCost = getText(bindingsCostEl);
    var cashPerBinding = bindingsPrice - bindingsCost;
    cash = cash + (cashPerBinding * totalBindings);

    // Обновление суммарной прибыли на веб-форме
    cash = Math.round(cash * 100) / 100;
    replaceText(cashEl, cash);
  } else
    alert("Error! Request status is " + request.status);
  }
}
```

Мы вычисляем накопленную сумму, начиная с прибыли, прибавляемой к текущему значению cash

Выбираем формат данных

Решите, какой формат данных лучше подходит в каждом из пяти приведенных примеров. Будьте внимательны, в одних случаях речь идет о запросах, в других — об ответах. Удачи!

Текст или XML



7 JSON или XML

Битва до победного конца

Пора вернуться в школу. Помните времена юности когда все разногласия решались колкостями, кулаками и убогим подражанием кун-фу? В этой главе мы вернемся в те дни оставив позади все дружеские слова и золотое правило этики. XML и JSON два разных формата отправки и приема данных в асинхронных запросах должны решить свои проблемы на ринге

Фрэнк: Зачем нам новый формат данных? По-моему, с того времени, как мы перешли на XML, все работает просто прекрасно.

Джо: Для тебя — возможно. Но нам, JavaScript-программистам, не так просто работать с XML.

Фрэнк: Почему? Я думал, что ты применяешь модель DOM для работы с кодом XML, полученным от моих сценариев.

Джо: Куда, применяем...

Фрэнк: И еще я видел твое приложение Top 5. Очень мило! И все это средствами DOM? Должно быть, ты хорошо разбираешься в этой теме...

Джо: О, для работы с веб-страницами меня DOM вполне устраивает. Но в том, что касается XML, она неудобна. Мне приходится тратить все свое время на перемещения вверх-вниз по документу XML только для того, чтобы получить несколько значений.

Фрэнк: И ты думаешь, что новый формат поможет? Расскажи о нем чуть подробнее.

Джо: Он называется JSON (JavaScript Object Notation), я...

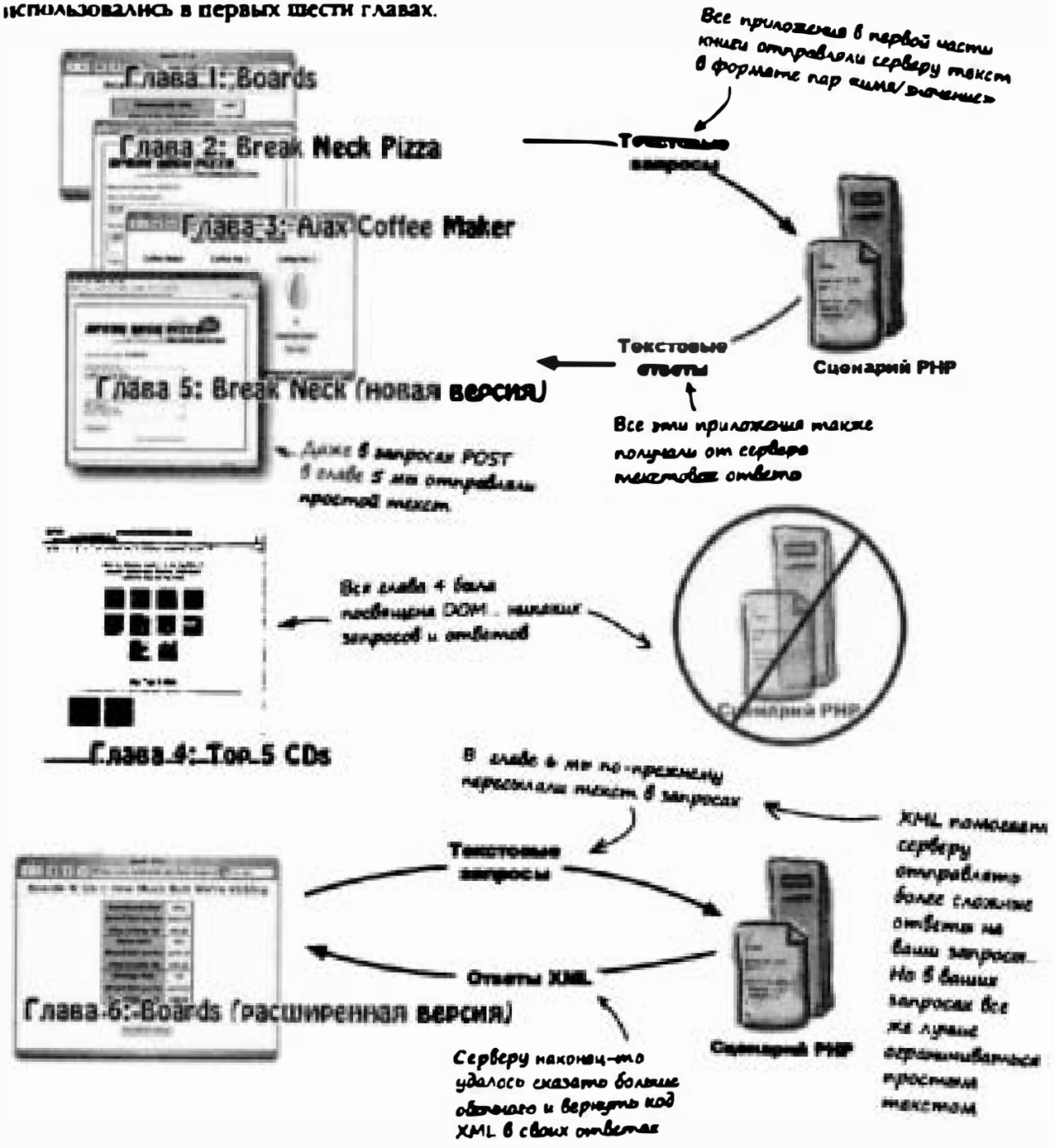
Фрэнк: Одну минуту. Формат данных — JavaScript? Что-то мне это не нравится. Я пишу PHP, Джо, а не JavaScript! О чем ты думаешь?

Джо: Но с ним удобно работать, он быстрый...

Фрэнк: Послушай, я занимаюсь серверным программированием и не собираюсь использовать JavaScript в коде PHP. Это просто безумие!

Форматы запросов и ответов

Прежде чем погружаться в спор «JSON против XML», давайте проанализируем форматы данных, которые нами уже использовались в первых шести главах.



Что использовать: XML или JSON?

Джо и другие программисты JavaScript присматриваются к JSON, но Фрэнк со своей группой PHP считают, что нужно использовать XML. Какой формат данных лучше? В этой главе мы разрешим JSON выяснить отношения с XML и посмотрим, кто из них переживет Чемпионат форматов данных по боям без правил.

Разминка перед бит

Войны напряжены, в их крови бурлит адреналин:
XML и JSON

XML: *(смотрит на JSON)*

JSON: Твое время пришло, XML. Сегодня весь мир увидит, что ты уже потерял былую хватку — особенно в том, что касается JavaScript и асинхронных приложений.

XML: Я уже слышал это прежде... Но я здесь, и все еще остаюсь королем среди форматов данных.

JSON: Ты так популярен только потому, что многие люди думают, будто ты единственный. Многие тебя терпеть не могут, XML. Ты слишком велик и неповоротлив, с тобой неудобно работать.

XML: Да, я большой, потому что могу справиться с чем угодно. Продавец сноубордов, HTML, оформление заказа... давай мне, и я это сделаю без проблем. Неужели кто-нибудь малого справится с множеством разных типов данных? Я так не думаю.

JSON: Может, и нет, но я быстро двигаюсь... Обычно гораздо быстрее тебя.

XML: Я тоже достаточно быстр, особенно если использовать мои атрибуты. И еще я универсален и способен на многое... Например, представить математическую формулу или целую книгу.

JSON: Да, только большинству моих пользователей не нужно пересылать по сети математические формулы. А эти угловые скобки? Уныние... Любой программист, знакомый с массивами, может начать работать со мной без изучения нелепого синтаксиса XML.

XML: Но разве тебя можно преобразовать во что-нибудь другое? Как с XSLT? А как насчет веб-служб... будешь утверждать, что справишься с веб-службами?

JSON: Не утомляй меня суть, мистер Угловые Скобки. Меня эти вещи попросту не интересуют. Я занимаюсь только одним: передачей информации с веб-страницы серверу и обратно без лишней работы... например, без получения доступа по дереву DOM. Знаешь кого-нибудь, кому бы нравилось это занятие?

XML: Да, как ни. Я знаю массу экспертов в области DOM, которые создают отличные пользовательские интерфейсы. Помнишь приложение Top 5? Это было весьма впечатляюще, и заняло всего 100 строк кода. Каждый, кто знаком с DOM, готов к использованию XML — прямо сегодня!

JSON: Послушай, все, что на самом деле нужно разработчикам — это облегченный формат данных, с которым удобно работать из JavaScript. А это я, а не ты, толстяк.

XML: И что на это скажут все серверы? PHP, Perl, Java... Не думаю, что они со всех ног побегут поддерживать тебя и твой «облегченный формат данных».

JSON: Да, это верно... Но есть библиотеки, при помощи которых они смогут работать со мной.

XML: Библиотеки? Если они захотят использовать библиотеки, то почему не выбрать общепринятый стандарт — такой как Document Object Model?

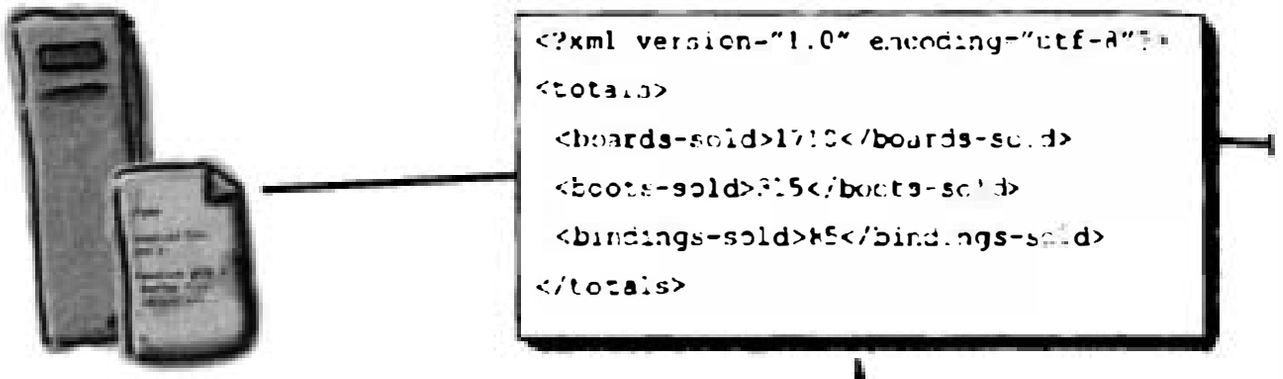
JSON: Мои библиотеки тоже когда-нибудь станут стандартными...

XML: А со мной работают сейчас, потому что я уже стал стандартом. И в конце концов, ты — еще один закрытый формат данных. Возможно, у тебя чуть больше преимуществ, чем у списков значений, разделенных запятыми, но я полагаю этому конец.

JSON: Неужели? Посмотрим, XML.

Чемпион в тяжелом весе: XML

Вы уже видели, как сервер возвращает XML в ответе на ваш запрос.



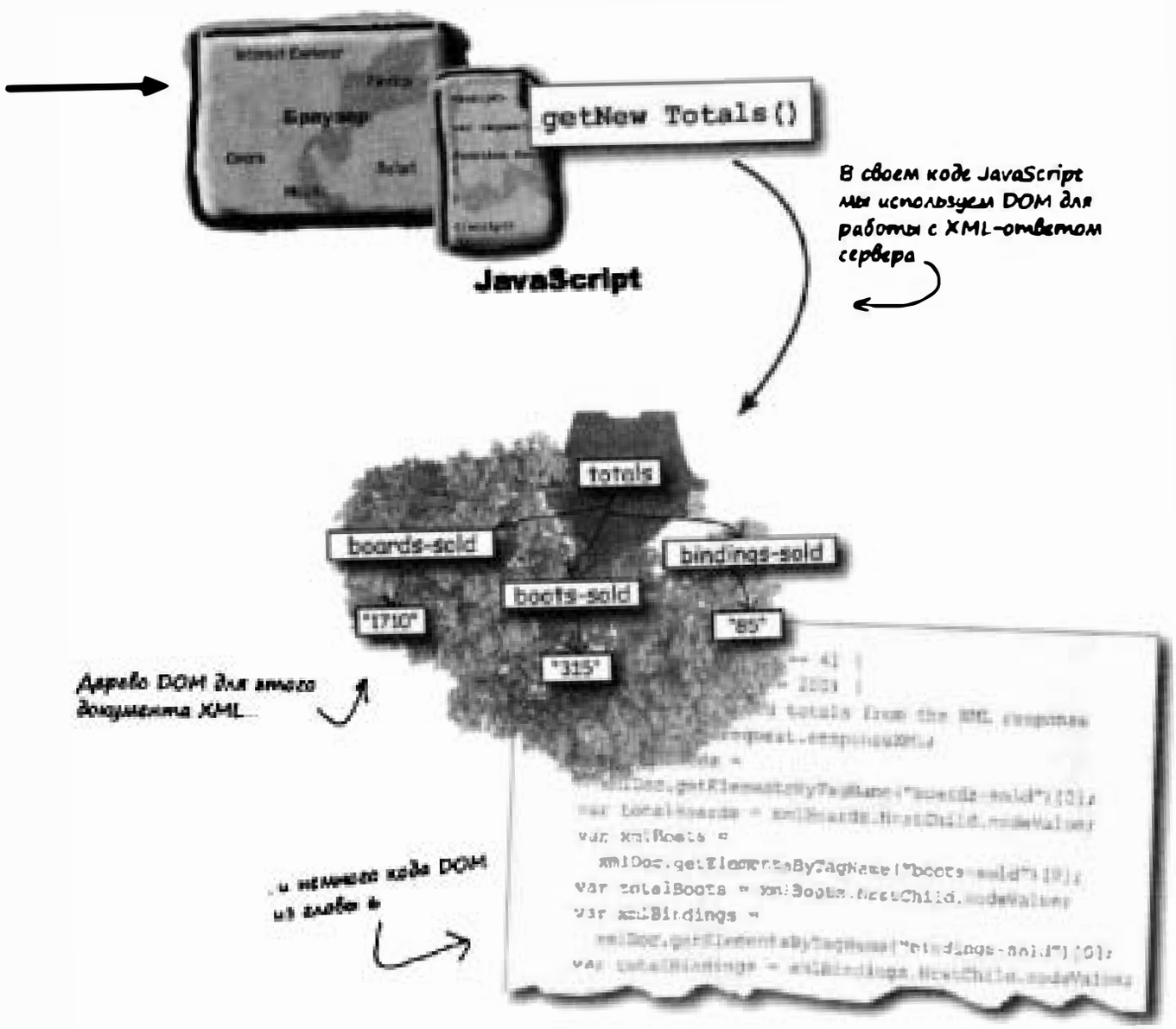
Сценарий PHP

Код XML, который возвращался сервером Boards в главе 6.

XML уже получил распространение и вы уже умеете использовать модель DOM для работы с данными XML.

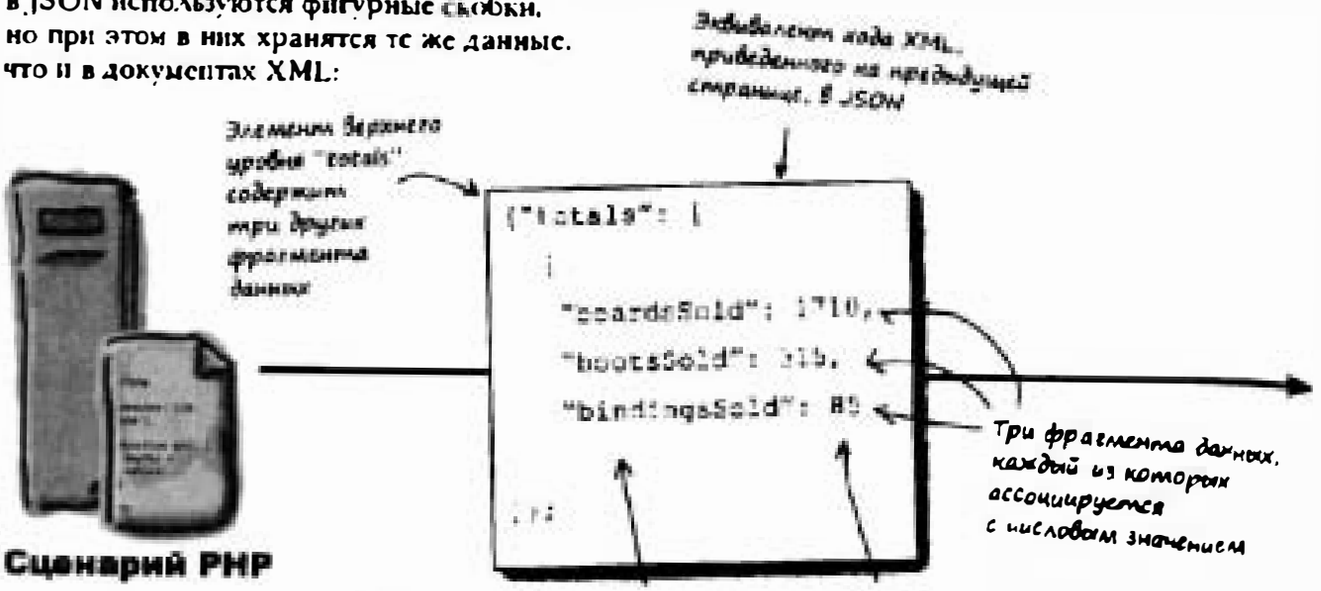
Мы работаем с XML при помощи DOM

Мы используем DOM для работы с XML



Претендент: JSON

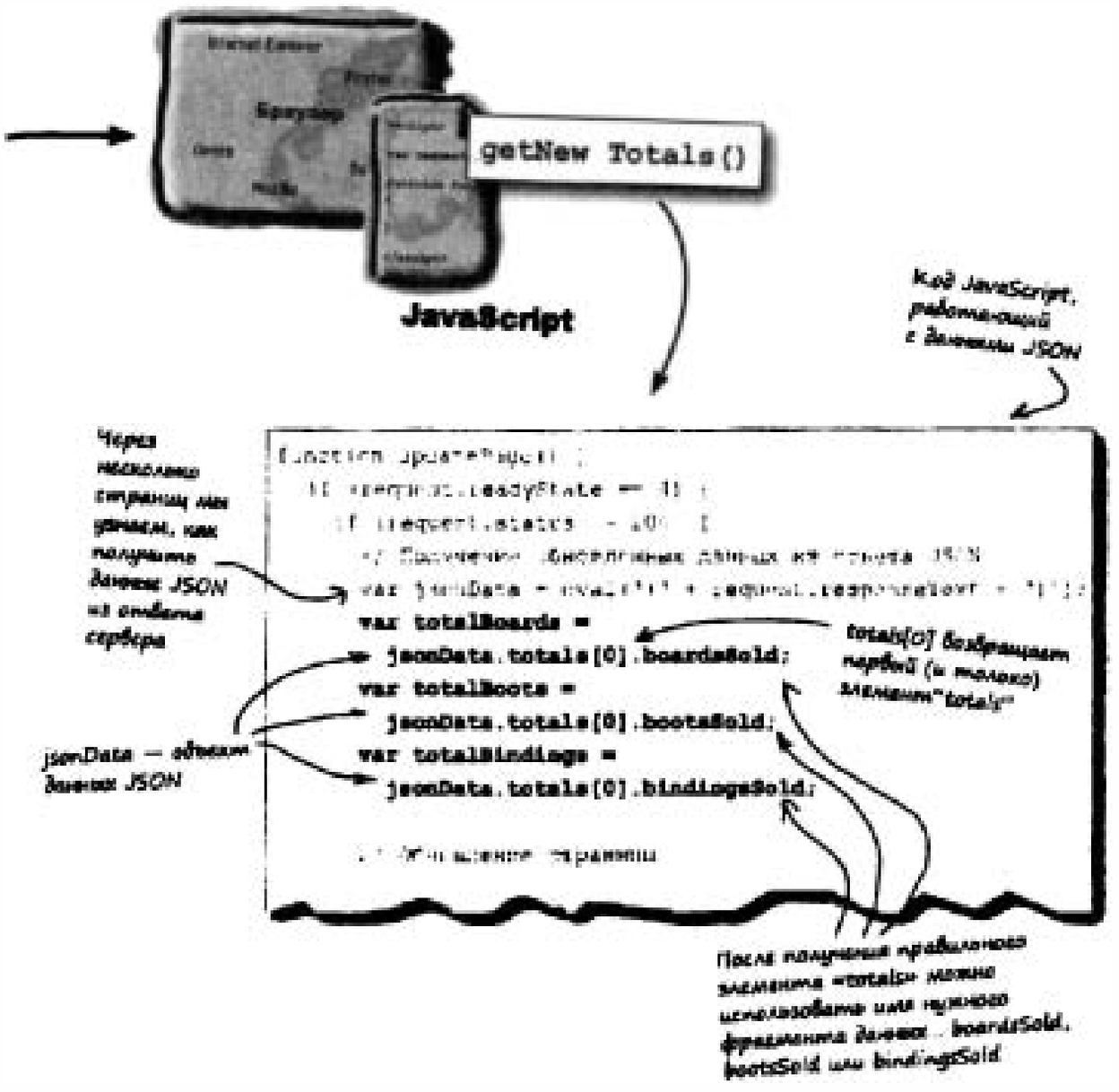
Вместо угловых скобок, характерных для XML, в JSON используются фигурные скобки, но при этом в них хранятся те же данные, что и в документах XML:



Технология JSON относительно недавно появилась на сцене Ajax, но уже имеет множество поклонников

Для работы с JSON используется «обычный» код JavaScript.

Для работы с данными JSON не требуется специальная объектная модель



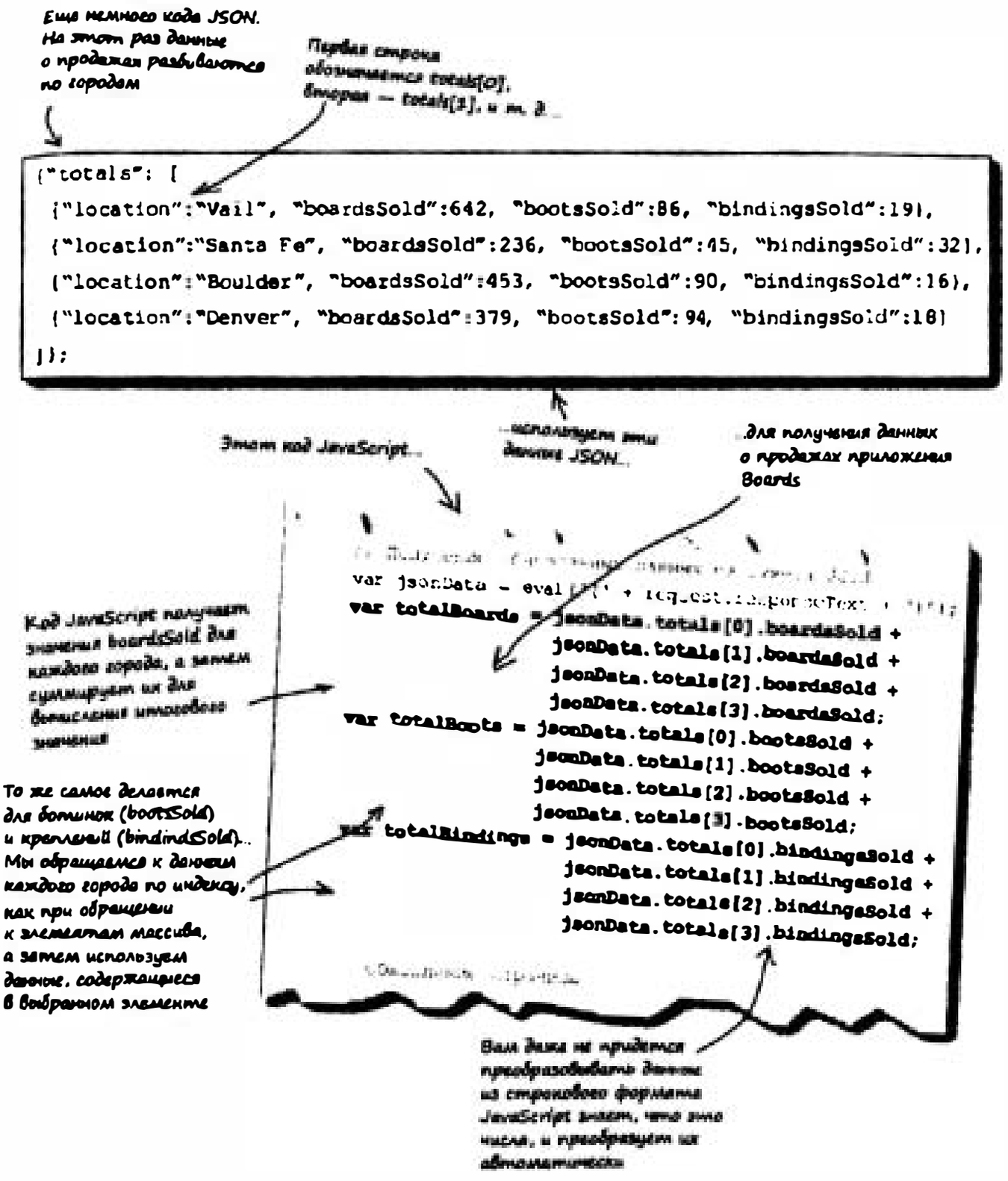
Значит, JSON — это обычный код JavaScript, верно? Мне не нужно беспокоиться ни о DOM, ни о специализированном инструментарии... Я просто использую JSON в своем коде, как использую, например, массивы.

JSON — это просто JavaScript

JSON — всего лишь способ представления объектов в JavaScript. Другими словами, JSON *является* JavaScript. Вам не придется работать с моделью DOM или другим инструментарием, или использовать JSON в своем коде JavaScript.

Более того, вы можете работать с данными, имеющими гораздо более сложную структуру — намного сложнее простых наборов значений, с которыми мы работали ранее. Давайте посмотрим, как в приложении Boards можно выполнить дальнейшую разбивку данных...

JSON — это JavaScript.


Еще немного кода JSON. На этот раз данные о продажах разбиваются по городам
Первая строка обозначается totals[0], вторая — totals[1], и т. д.
{"totals": [
{"location":"Vail", "boardsSold":642, "bootsSold":86, "bindingsSold":19},
{"location":"Santa Fe", "boardsSold":236, "bootsSold":45, "bindingsSold":32},
{"location":"Boulder", "boardsSold":453, "bootsSold":90, "bindingsSold":16},
{"location":"Denver", "boardsSold":379, "bootsSold":94, "bindingsSold":18}
]};
...используем эти данные JSON...
Этот код JavaScript...
...для получения данных о продажах приложения Boards
var jsonData = eval
var totalBoards = jsonData.totals[0].boardsSold +
jsonData.totals[1].boardsSold +
jsonData.totals[2].boardsSold +
jsonData.totals[3].boardsSold;
var totalBoots = jsonData.totals[0].bootsSold +
jsonData.totals[1].bootsSold +
jsonData.totals[2].bootsSold +
jsonData.totals[3].bootsSold;
var totalBindings = jsonData.totals[0].bindingsSold +
jsonData.totals[1].bindingsSold +
jsonData.totals[2].bindingsSold +
jsonData.totals[3].bindingsSold;
Код JavaScript получает значения boardsSold для каждого города, а затем суммирует их для вычисления итогового значения
То же самое делается для ботинок (bootsSold) и креплений (bindingSold). Мы обращаемся к данным каждого города по индексу, как при обращении к элементам массива, а затем используем данные, содержащиеся в выбранном элементе
Вам даже не придется преобразовывать данные из строкового формата JavaScript знает, что это числа, и преобразует их автоматически

В: Значит, JSON — еще один формат данных, как XML?

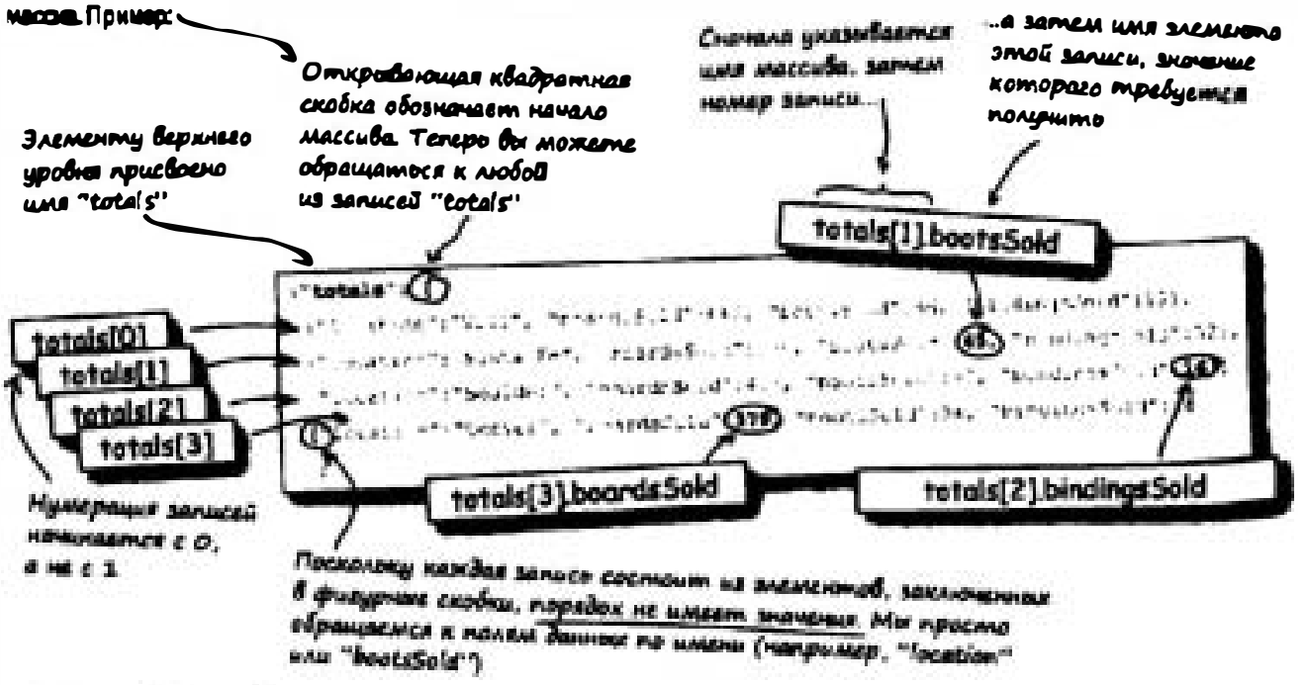
О: Да. Каждый раз, когда веб-страница обменивается информацией с сервером, эта информация должна быть каким-то образом отформатирована. До настоящего времени мы использовали обычный текст в запросах, и текст/XML в ответах. JSON — еще один способ пересылки данных.

В: Если у нас уже имеются такие варианты, как XML и текст, зачем нужен JSON?

О: Многие программисты JavaScript не привыкли читать или писать код XML. Несмотря на то что DOM скрывает многие технические детали работы с XML, формат JSON больше похож на массивы и списки, привычные для программистов JavaScript. Если вы знаете и XML, и JSON — тем лучше: значит, вы можете выбрать формат по своему усмотрению.

В: Меня слущают все эти фигурные скобки. Можно еще раз объяснить, как они работают?

О: Фигурные скобки { и } содержат неупорядоченные наборы данных, а символы [и] обозначают упорядоченный массив. Пример:



В: Должен ли я конвертировать мою версию приложения Boards, чтобы использовать JSON?

О: Да, должны. Мы создали версию приложения Boards, которая получает JSON с сервера. Вы можете ознакомиться с ней на странице http://www.headfirstlabs.com/books/hfajax/chapter07/boards/boards.html.

Но нет ничего, что говорило бы, что вы должны использовать JSON в вашей версии приложения Boards, или в ваших собственных приложениях. Это — только дополнительная возможность, наряду с XML и простым текстом. Выбор исключительно за вами.

В: Но что лучше: XML или JSON?

О: Все зависит от вашего приложения. В большинстве случаев JSON лучше смотрится при работе со строками и в таблицах или с массивами, но XML может облегчить представление сложных структурированных данных. Выбор — за вами!

Прежде чем обсуждать взаимодействие JSON с такими серверными языками, как PHP, давайте поподробнее разберемся с использованием данных JSON в JavaScript. Рассмотрим следующую структуру данных JSON:

```
{"books": [
 {"title":"Hyperion", "author":"Dan Simmons", "isbn":"0553283685"},
 {"title":"The Stars My Destination", "author":"Alfred Bester", "isbn":":"0679767800"},
 {"title":"Black House", "author": [ "Stephen King", "Peter Straub"],
                          "isbn":"0345441036"},
 {"title":"The Golden Compass", "author":"Philip Pullman", "isbn":"0679879242"},
]};
```

Ниже приведены некоторые значения из структуры JSON. Напишите конструкции JavaScript, которые бы возвращали указанные значения из структуры данных JSON. Предполагается, что данные JSON хранятся в объекте с именем **jsonData**.

Чтобы вам было проще, мы выполнили первое задание

"Alfred Bester"	`var better = jsonData.books[1].author;`
"0679879242"	
"0345441036"	
"Dan Simmons"	
"Black House"	
"Peter Straub"	

Ответы на с...

Мы же сказали, что JSON — это JavaScript

Важным преимуществом JSON является то, что он представляет собой JavaScript, поэтому код JavaScript ваших веб-страниц очень легко работает с данными JSON. Если вы уже освоили работу с массивами в JavaScript, то сможете использовать массивы, полученные в JSON-ответах сервера. Если вы ранее уже работали с объектами JavaScript, то и JSON вам покажется совершенно естественным.

Впрочем, есть и оборотная сторона: такие языки, как PHP, Perl и Java, не поймут JSON, если им не помочь. Вероятно, вам потребуется библиотека, упрощающая создание и вывод данных в формате JSON в серверных сценариях и программах. Вам также надо будет разобраться в том, как эту библиотеку использовать, что тоже потребует времени и усилий.

О том, где достать одну из самых популярных библиотек JSON для PHP, рассказано в приложении 1

Обновленный сценарий PHP Фрэнка..
Теперь в ответах на запросы он выдает данные JSON вместо данных XML

Для кодирования ответа JSON в PHP потребуется объект Services_JSON

JSON.php — одна из нескольких библиотек, обеспечивающих работу с данными JSON в PHP

Ближайшим аналогом структур данных JSON являются массивы PHP

```
require_once('JSON.php');

$json = new Services_JSON();
$vail = array('location'     => 'Vail',
              'boardsSold'   => $vailBoards,
              'bootsSold'    => $vailBoots,
              'bindingsSold' => $vailBindings);
$santaFe = array('location'     => 'Santa Fe',
                 'boardsSold'   => $santaFeBoards,
                 'bootsSold'    => $santaFeBoots,
                 'bindingsSold' => $santaFeBindings);
$boulder = array('location'     => 'Boulder',
                 'boardsSold'   => $boulderBoards,
                 'bootsSold'    => $boulderBoots,
                 'bindingsSold' => $boulderBindings);
$denver = array('location'     => 'Denver',
                'boardsSold'   => $denverBoards,
                'bootsSold'    => $denverBoots,
                'bindingsSold' => $denverBindings);
$totals =
  array('totals' =>
    array($vail, $santaFe, $boulder, $denver));
$output = $json->encode($totals);
print($output);
```

В JSON широко используются массивы, и даже массивы массивов

Какие сложности. Кажется, вывести ответ XML было проще

Функция encode() преобразует массивы PHP в структуры JSON

Получив структуру данных JSON, мы отправляем ее в ответе

Хотите сравнить JSON с XML? Вернитесь к главе 6 и сравните сценарий PHP, возвращающий ответ в формате XML, со сценарием PHP, возвращающим JSON.

Даже если мне удастся убедить Фрэнка переделать его сценарии, чтобы они отправляли ответы JSON, что мне делать с этими ответами? Снова использовать свойство responseText? Или responseXML?

Можно подумать, мне хочется возиться с JavaScript в своих сценариях PHP...

JSON пересылается в виде текста

Данные JSON, возвращаемые сервером, пересылаются в виде текста. Следовательно, для получения доступа к ним следует использовать свойство **responseText** объекта запроса. Однако формат JSON ориентирован на работу с объектами, поэтому текст необходимо преобразовать в объектную форму. Для этой цели можно воспользоваться функцией JavaScript **eval()**:

Функция eval() получает строку и преобразует данные JSON, возвращенные сервером, в объект JavaScript

```
var jsonData = eval('(' + request.responseText + ')');
```

После выполнения этой команды объект jsonData будет содержать данные JSON, преобразованные в объектную форму

Свойство request.responseText содержит данные JSON в текстовой форме, полученной от сервера

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ

В: Разве я не могу вывести JSON в сценарии PHP в виде текста, и обойтись без использования библиотеки?

О: Если очень захочется — можете. Правда, вам придется достаточно хорошо знать формат JSON, чтобы вывести данные вручную, и не допустить ни одной ошибки. Но если вы действительно хорошо знаете JSON, такое решение существует, и вам не придется использовать библиотеки JSON в своих сценариях PHP.

Тем не менее применение библиотек для вывода JSON имеет одно серьезное преимущество: вашим разработчикам сценариев и программистам, работающим на стороне сервера, не придется изучать JSON. Они работают с обычными структурами данных PHP и Java, а преобразование этих структур в формат данных JSON поручается вспомогательному инструментарию.

В: Значит, для применения JSON в веб-приложениях потребуются библиотеки?

О: Код JavaScript может использовать JSON без каких-либо специальных библиотек или инструментариев. Но серверные программы, которые обычно пишутся на PHP, Java или Ruby, вероятно, не смогут работать с форматом данных JSON, если вы им не поможете. Следовательно, вам придется найти библиотеку, при помощи которой эти языки программирования будут понимать данные PHP и работать с ними.

В: Где достать библиотеки для моих серверных языков?

О: Информацию о многих популярных библиотеках JSON можно найти на сайте JSON по адресу http://www.json.org. Таких библиотек существует для всех основных языков программирования, поэтому вы без особого труда подберете для себя подходящий вариант.

В: А когда мой сервер возвращает данные JSON, я обращаюсь к ним через свойство responseText?

О: Точно. Сервер возвращает данные JSON в виде текста, который затем пересылается по сетевым соединениям. Браузер сохраняет текстовый ответ сервера в свойстве responseText объекта запроса, поэтому для получения ответа JSON следует обращаться именно к этому свойству.

В: Зачем нужна функция eval()?

О: Наименование сокращения JSON означает «JavaScript Object Notation», т. е. «объектная запись JavaScript». JSON — формат данных, но данные в нем представляются в виде объектов, а не в виде простого текста. Однако ответ, полученный от сервера, представляет собой простой текст, хранящийся в свойстве responseText объекта запроса JavaScript.

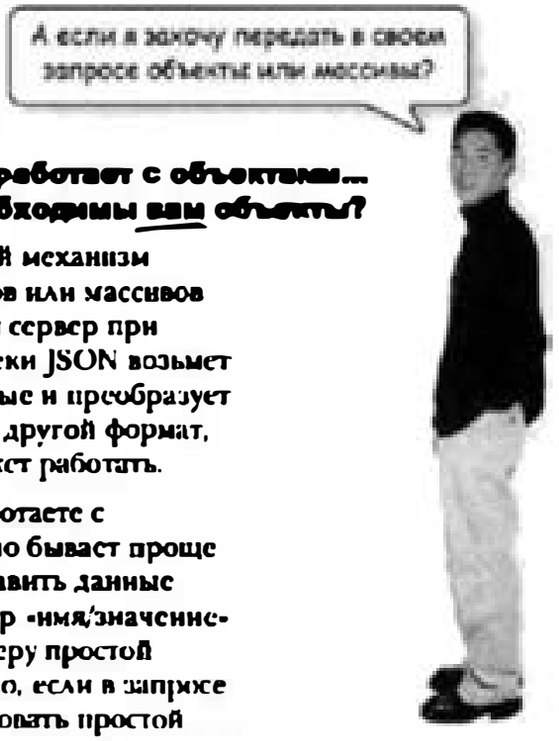
Чтобы преобразовать текст в объект, необходимо вызвать функцию JavaScript eval(). Функция eval() получает строковый аргумент и пытается либо выполнить указанную строку как команду, либо преобразовать ее в объект. В нашем случае функция eval() видит, что ей передана текстовая версия данных JSON, и возвращает объект JSON, который в дальнейшем может использоваться в коде JavaScript.

В: И дерево DOM при этом не используется?

О: Точно. Для работы с данными JSON модель DOM не понадобится.

В запросах к серверу можно использовать JSON, XML или текстовый формат

В запросах к серверу можно передавать данные в формате JSON, XML или в виде простого текста. Но как и в случае с XML, при передаче запросов серверу часто бывает проще использовать простые текстовые пары «имя/значение».



JSON отлично работает с объектами... ...но так ли необходимы вам объекты?

JSON — отличный механизм передачи объектов или массивов серверу. Но затем сервер при помощи библиотеки JSON возьмет полученные данные и преобразует их в массивы или другой формат, с которым он может работать.

Даже если вы работаете с объектами, обычно бывает проще и удобнее представить данные объекта в виде пар «имя/значение» и отправить серверу простой текст. Как правило, если в запросе *возможно* использовать простой текст, это *следует* сделать.

По возможности используйте в своих запросах текстовые данные.

Какой формат данных лучше?

Джо: Я и брюссельскую капусту умею готовить. Но это не значит, что я люблю ее есть!

Фрэнк: Не вижу, что мы реально выигрываем от использования JSON. Может, тебе с ним немного удобнее работать, но у меня возникает масса проблем.

Джо: Не знаю. Мне показалось, что работать с библиотекой JSON.php не так уж сложно.

Фрэнк: Конечно, но мне придется учить всех моих программистов работать с форматом JSON. Они уже знают XML, и для него нам не нужны специальные библиотеки.

Джо: По-моему, все понятно. JSON — это круто! Я недавно читал статью, в которой говорилось, насколько он превосходит XML при пересылке посетителей.

Фрэнк: А я как раз прочитал другую статью, в которой говорилось прямо противоположное! Думаю, все зависит от данных... Невозможно доказать, что JSON всегда быстрее XML в любой ситуации...

Джо: Или что XML быстрее JSON!

Фрэнк: Верно. Полагаю, мы так и не придем к единомыслию?

Джо: Но мы все-таки должны выбрать тот или иной формат?

Какой формат данных

предпочтете вы?

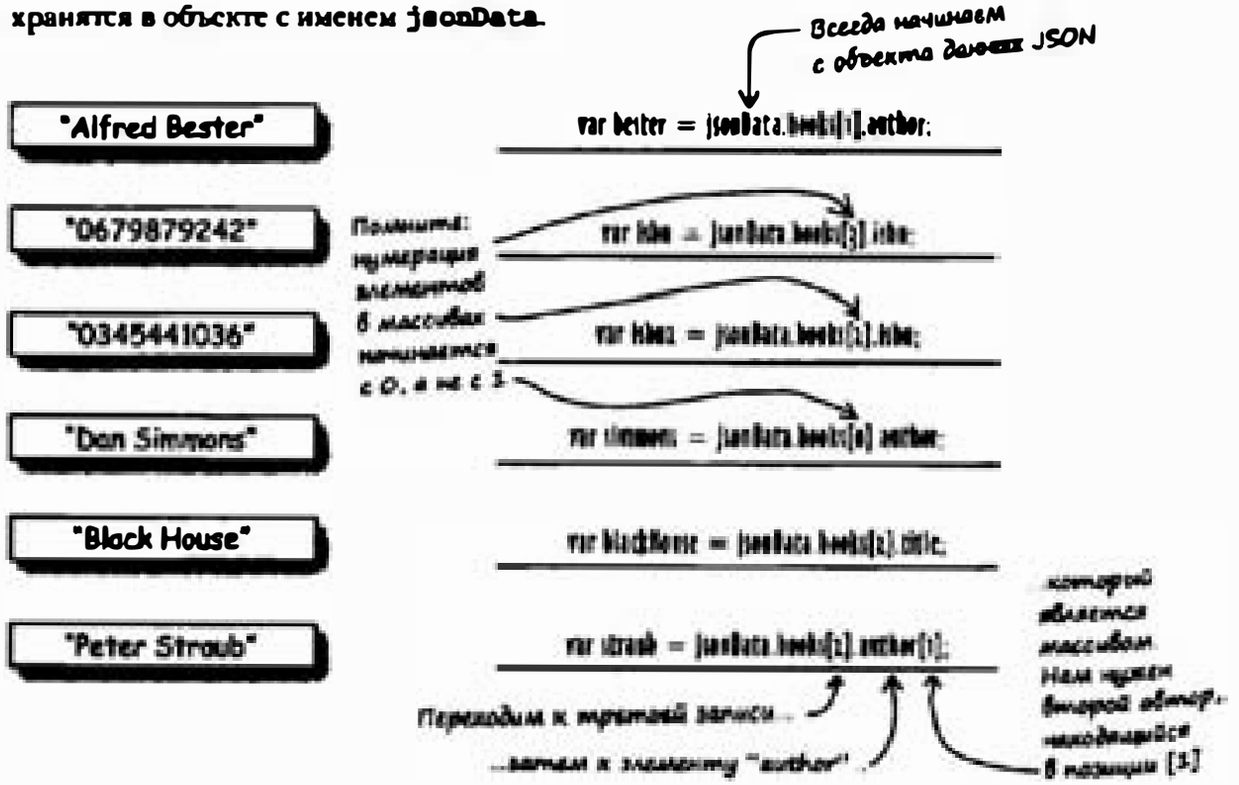
...за вами!

Просто сделайте это — решение

Прежде чем обсуждать взаимодействие JSON с такими серверными языками, как PHP, давайте поподробнее разберемся с использованием данных JSON в JavaScript. Рассмотрим следующую структуру данных JSON:

```
{"books": [
 {"title":"Hyperion", "author":"Dan Simmons", "isbn":"0553283685"},
 {"title":"The Stars My Destination", "author":"Alfred Bester", "isbn:":"0679767800"},
 {"title":"Black House", "author": [ "Stephen King", "Peter Straub"],
                          "isbn":"0345441036"},
 {"title":"The Golden Compass", "author":"Philip Pullman", "isbn":"0679879242"},
]};
```

Ниже приведены некоторые значения из структуры JSON. Напишите конструкции JavaScript, которые бы возвращали указанные значения из структуры данных JSON. Предполагается, что данные JSON хранятся в объекте с именем **jsonData**.



Приложение 1

Дополнительные материалы

Только для вас: прощальный подарок от Head First Labs. Вернее, в этом приложении наши читатели найдут пять специальных [illegible]. Мы бы охотноостались с вами и рассказали еще много интересного, но сейчас вам пора взять все, что вы узнали, и самостоятельно отправиться в холодный, жестокий мир веб-программирования. И все же бросать вас без [illegible] подготовки не хочется, поэтому в приложении вы найдете пять основных тем, не вошедших в книгу.

А потом — все. Будет еще одно [illegible], и алфавитный указатель, и еще несколько страниц, которые нас заставили вставить по соображениям маркетинга... но на самом деле вы уже почти добрались до самого конца книги.

№ 1: Инструментарии Ajax

Также часто встречается термин «инфраструктура» (framework). Не беспокойтесь... это то же самое

Мы несколько раз упоминали «инструментарии» (пакеты разработчика) Ajax, упрощающие выполнение таких стандартных задач, как создание объекта запроса или отправка запроса. Когда вы начнете понимать, как написать такой код самостоятельно, просмотрите код некоторых пакетов такого рода и выясните, предоставляют ли они какую-либо функциональность, которая бы вам могла пригодиться. Далее приводятся примеры популярных инструментальных пакетов Ajax.

Prototype

Где взять: http://prototype.conio.net/

Как использовать:

Просто вставьте файл prototype.js, загруженный вами с сайта Prototype, в код HTML

```
<head>
<title>The New and Improved Break Neck Pizza</title>
<link rel="stylesheet" type="text/css" href="breakneck.css" />
<script type="text/javascript" src="prototype.js"></script>
<script type="text/javascript" src="pizza.js"></script>
</head>
```

Отправка запроса:

Ajax.Request — класс Prototype для работы с запросами Ajax

```
var request = new Ajax.Request(
  url,
  {
    method: 'get',
    parameters: 'phone=2142908762&name=Mary',
    onSuccess: updatePage,
    onFailure: reportError
  }
);
```

url — URL для отправки запроса

Допустимые значения "get" или "post"

Пара «имя/значение», передаваемая серверной программе

Одна из приятных особенностей Prototype: возможность назначения функций для разных типов ответов

Prototype передаст объект запроса в параметре функции обратного вызова, поскольку объект запроса не является глобальной переменной

Обработка ответа:

```
function updatePage(request) {
  var response = request.responseText;
}
```

Проверять состояние готовности и код статуса не нужно... Prototype сделает это за вас, и при возникновении проблем вызовет функцию onFailure

Dojo

Где взять: http://dojotoolkit.org/

Сначала включаем файл dojo.js, загруженный с сайта Dojo

Как использовать:

```
<script type="text/javascript" src="dojo.js"> </script>
<script type="text/javascript" src="pizza.js"> </script>
<script language="JavaScript" type="text/javascript">
  dojo.require("dojo.io.bind");
</script>
```

В статическом коде JavaScript вызовите dojo.require() для всех пакетов Dojo, которые вы собираетесь использовать

dojo.io.bind — пакет Dojo, содержащий Ajax-код и служебные функции

Отправка запроса:

Обычно бывает проще разместить все аргументы, используемые с Dojo, в структуре данных JavaScript

```
var arguments = {
  url:      'lookupCustomer.php',
  method:   'GET',
  content:  'phone=2142908762&name=Mary',
  error:    reportError,
  load:     updatePage
};

dojo.io.bind(arguments);
```

Эти аргументы аналогичны тем, которые использовались с Prototype

После подготовки аргументов вызовите функцию dojo.io.bind() и передайте ей созданную структуру

Обработка ответа:

Dojo предполагает, что функции обратного вызова получают несколько параметров. В большинстве случаев для вас представляет интерес только параметр с именем "value". В нем содержится ответ сервера

```
function updatePage(type, value, evt) {
  var response = value;
}
```

№ 2: script.aculo.us и другие интерфейсные библиотеки

Наряду с инструментариями разработчика Ajax также существует немало замечательных библиотек JavaScript, позволяющих создавать весьма эффектные пользовательские интерфейсы. Эти библиотеки представляют собой внешние файлы JavaScript, которые вы можете включать и использовать в своих приложениях (как асинхронных, так и обычных).

script.aculo.us

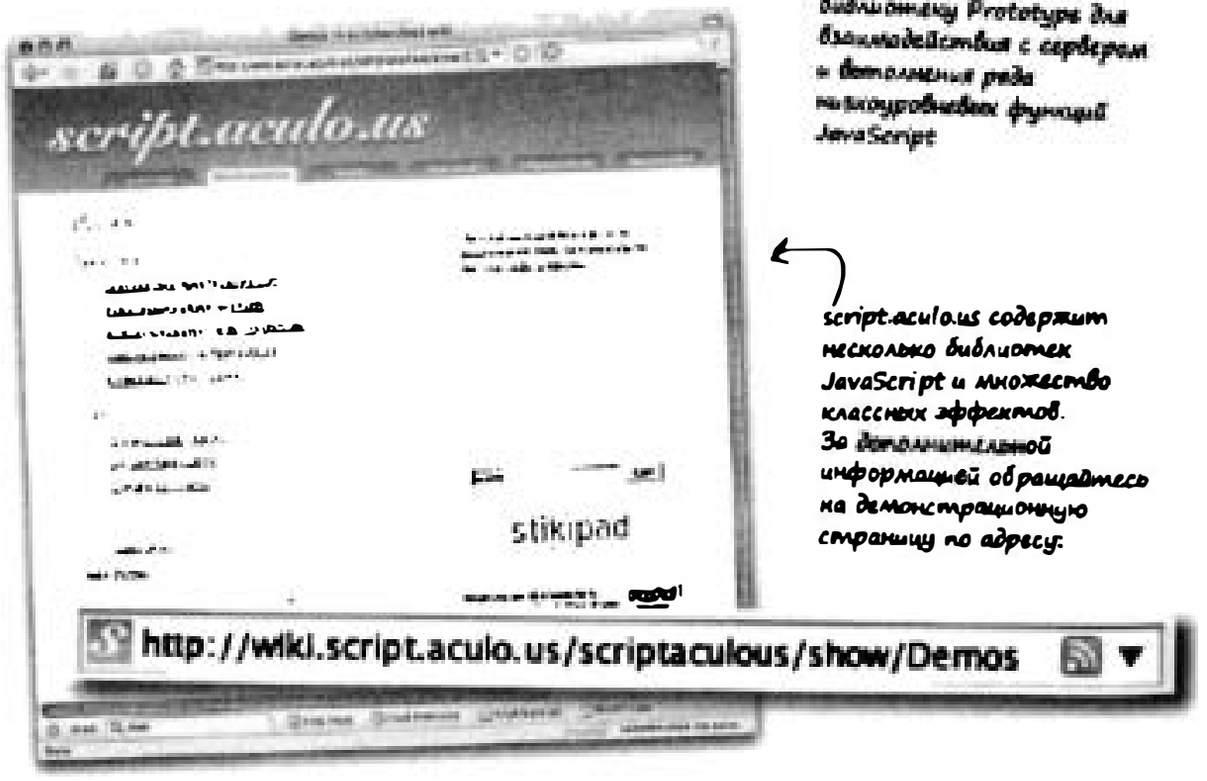
Где взять: http://script.aculo.us/

Как использовать:

Ничего нового. Дополнительные библиотеки JavaScript включаются в HTML.

```
<head>
<script type="text/javascript" src="prototype.js"> </script>
<script type="text/javascript" src="scriptaculous.js"> </script>
</head>
```

Script.aculo.us использует библиотеку Prototype для взаимодействия с сервером и дополнения ряда низкоуровневых функций JavaScript



script.aculo.us содержит несколько библиотек JavaScript и множество классных эффектов. За дополнительной информацией обращайтесь на демонстрационную страницу по адресу:

Rico

Где взять: http://openrico.org/

Как и script.aculo.js, Rico использует Prototype, а также вспомогательную библиотеку util

Как использовать:

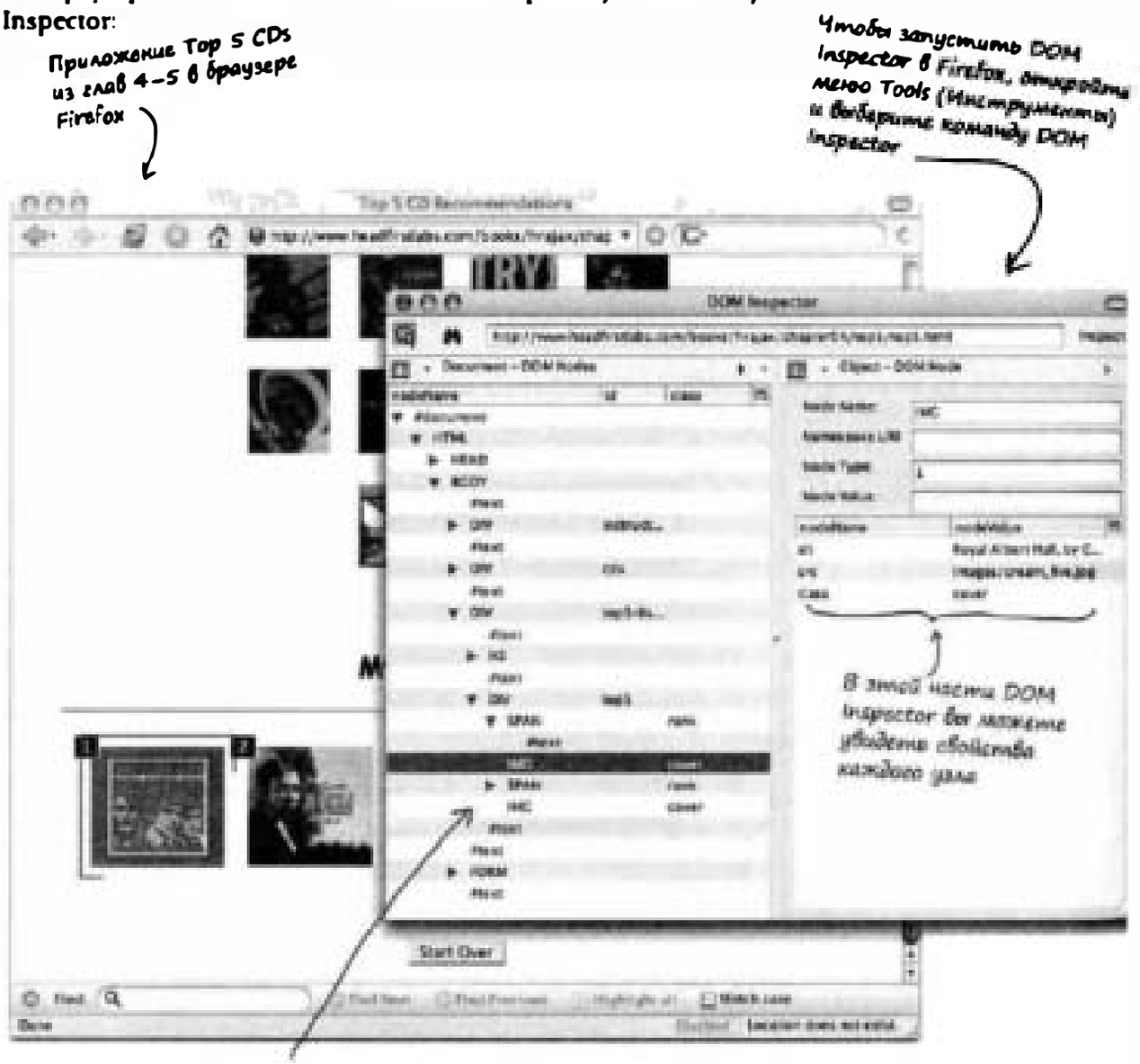
```
<head>
 <script type="text/javascript" src="prototype.js"> </script>
 <script type="text/javascript" src="rico.js"> </script>
 <script type="text/javascript" src="util.js"> </script>
</head>
```

Как и script.aculo.js, Rico обладает множеством замечательных возможностей из области пользовательского интерфейса — например, поддержкой функции перетаскивания

Я перетаскиваю текст из списка имен в графическое поле...

№ 3: Анализ DOM

Вы уже стали настоящим профессионалом по работе с моделью DOM для оперативного обновления веб-страниц. Но как узнать, что именно видит веб-браузер после внесения изменений в страницу? Воспользуйтесь DOM Inspector:



ВНИМАНИЕ: DOM Inspector не устанавливается в Windows по умолчанию. Во время установки Firefox выберите режим полнофункциональной установки (Custom Install) и включите установку инструментов веб-разработчика (Web Developer Tools). В этом случае программа DOM Inspector будет установлена в вашей версии Firefox.

Анализ DOM в Internet Explorer

Для анализа дерева DOM в браузере Internet Explorer для Windows необходимо загрузить и установить отдельную программу.

Программа IE DOM Inspector является условно-бесплатной. Если она вам понравится, и вы решите использовать ее более 15 дней, заплатите $29.99

Где взять: http://www.ieinspector.com/dominspector/

Как использовать:

Загрузите EXE-файл с сайта IE DOM Inspector и установите программу. IE DOM Inspector работает в течение 15 дней; по истечении этого срока придется либо купить программу, либо удалить ее из системы.

IE DOM Inspector в действии

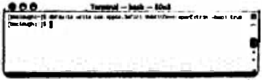
Анализ DOM в Safari

Чтобы проанализировать дерево DOM в Safari, вам понадобится WebKit — системная библиотека с открытыми исходными текстами, используемая в приложениях Mac OS X (таких, как Safari, Dashboard и Mail).

Где взять: http://webkit.opendarwin.org/

Как использовать:

Загрузите WebKit с сайта проекта. Введите в приглашении терминала Mac OS X следующую команду:



Щелкните правой кнопкой мыши и выберите команду «Inspect Element», чтобы вызвать DOM Inspector в Safari WebKit

№ 4: Использование библиотек JSON в сценариях PHP

Вы уже видели, как JSON помогает отправлять и принимать сложные объекты в Ajax-приложениях. Но если вы не хотите вводить код JSON вручную в сценариях PHP, потребуется вспомогательная библиотека. В этом разделе показано, как эффективно использовать JSON в сценариях PHP.

JSON-PHP

Первоначально, JSON-PHP — самая удобная библиотека JSON для сценариев PHP, простая в установке и в работе

Где взять: http://mike.teczno.com/json.html

Как использовать:

Вызов require_once() гарантирует, что файл JSON.php будет загружен только один раз, даже если ссылки на него присутствуют в нескольких сценариях PHP

Сначала вызовите require_once() и включите в сценарий файл JSON.php, загруженный с сайта JSON-PHP

```
require_once('JSON.php');

$json = new Services_JSON();
```

Создайте новую переменную и присвойте ей новый экземпляр класса Services_JSON. Этот класс позволяет выводить данные JSON по структурам данных PHP

Создайте массивы и переменные, как это обычно делается в коде PHP

```
$order1 = array('name'      => 'Jim',
                'size'      => 'large',
                'beverage'  => 'mocha',
                'coffeemaker' => 1);
$order2 = array('name'      => 'Bob',
                'size'      => 'medium',
                'beverage'  => 'latte',
                'coffeemaker' => 2);
```

Когда все будет готово к преобразованию структуры данных в формат JSON, используйте объект Services_JSON и вызовите encode() для переменной PHP

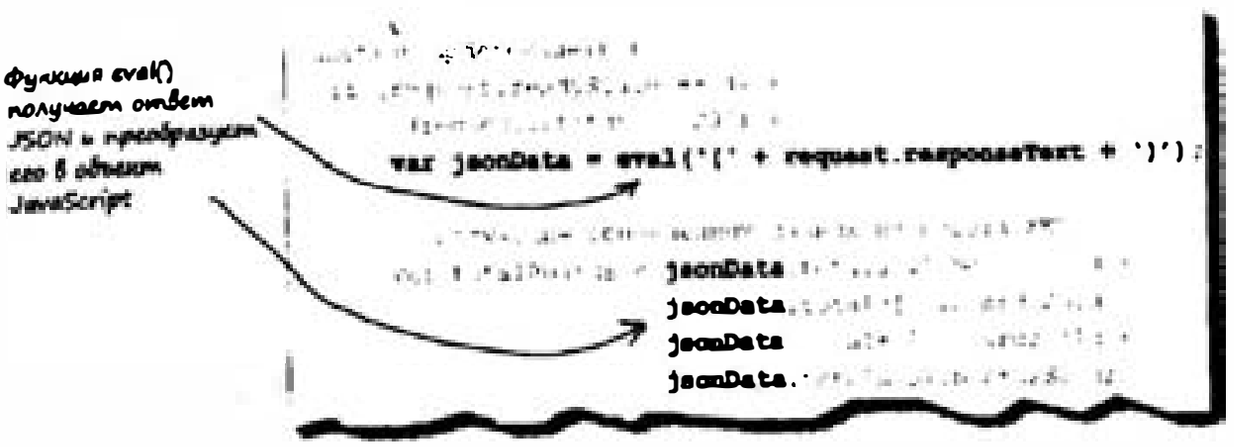
```
$orders =
  array('coffeeOrders' =>
        array($order1, $order2));
$output = $json->encode($orders);
```

Наконец, вызовите функцию print() для возвращения закодированных данных JSON браузеру, от которого поступила заявка

```
print($output);
```

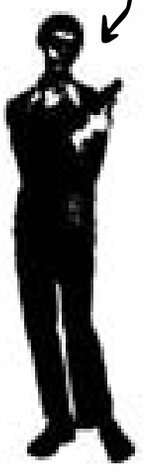
№ 5: Использование eval() с JSON

В главе 7 вы научились использовать функцию **eval()** для обработки данных JSON, возвращенных серверным сценарием:



Недостаток функции eval() заключается в том, что она обрабатывает ответ JSON, полученный от сервера, без каких-либо проверок безопасности... Если какая-нибудь злонамеренная организация внесет изменения в ответ сервера, это может кончиться выполнением вредоносного кода из JavaScript

Такая, как проект «Хаос»



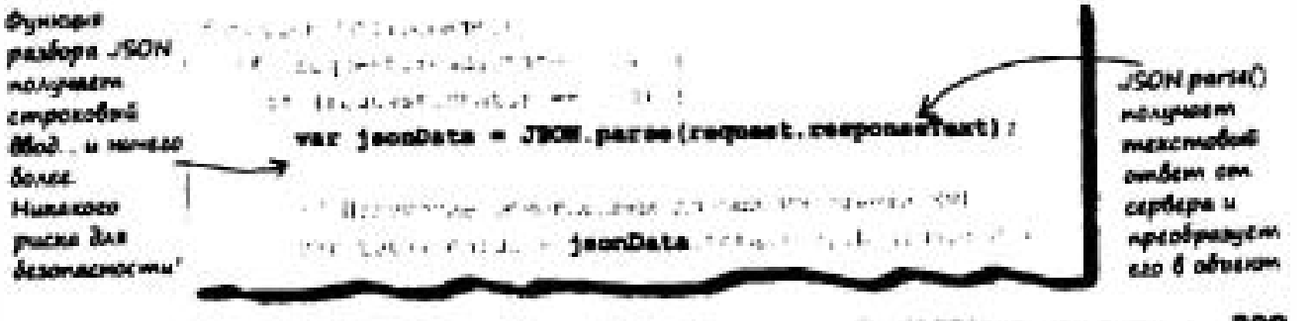
Использование программы разбора JSON

Если вы заботитесь о безопасности при работе с JSON, возможно, вам стоит использовать подсистему разбора JSON (парсер) и обойтись без использования **eval()** в функциях JavaScript

Где взять: **http://www.json.org/js.html**

Ссылка на файл JSON.js, загруженный с сайта json.org, включается в тег <script> в коде HTML

Как использовать:



Приложение 2: Инструментарий Ajax и DOM

«Все, что мне нужно — это код»

И еще немного подарков. На этих последних страницах приводится служебный код, слишком сложный для того, чтобы рассматривать его в предыдущих главах. Но сейчас вы уже готовы к знакомству с этими служебными функциями Ajax и DOM.

ajax.js

Мы разместили весь этот код в файле с именем ajax.js. Впрочем, ему можно присвоить произвольное имя — главное не забудьте изменить тег HTML <script> и включить в него правильное имя файла

Вы уже много раз видели этот код JavaScript. В нем мы создаем объект запроса и убеждаемся в том, что он будет работать во всех современных браузерах, от Internet Explorer до Firefox и Opera. Привыкайте к этому коду, если он вам кажется непонятным... он заложен в основу каждого Ajax-приложения.

```
var request = null;

try {
  request = new XMLHttpRequest();
} catch (trymicrosoft) {
  try {
    request = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (othermicrosoft) {
    try {
      request = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (failed) {
      request = null;
    }
  }
}

if (request == null)
  alert("Error creating request object!");
```

Вы встретили много исходных текстов, в которых используется false вместо null, но они не будут работать на некоторых версиях IE. Замените false на null, и ваш код будет работать на любых платформах

Напомним: мы должны сначала попробовать XMLHttpRequest для наших браузеров, как Opera и Mozilla, а затем ActiveXObject для Internet Explorer

Follow the code... things end up here if all the different attempts at creating a request object fail.

Ладно, признаем... Это не самый изящный способ обработки ошибок. Но сейчас вы уже готовы к тому, чтобы самостоятельно реализовать более мощный механизм обработки ошибок

Использование ajax.js

Чтобы использовать файл ajax.js, создайте тег **<script>** в секции **<head>** кода HTML вашей страницы, и включите в него ссылку на имя файла:

Секция <head> страницы HTML

```
<head>
 <title>The New and Improved Break Neck Pizza</title>
 <link rel="stylesheet" type="text/css" href="breakneck.css" />
 <script type="text/javascript" src="ajax.js"> </script>
 <script type="text/javascript" src="pizza.js"> </script>
</head>
```

В данном примере код HTML содержит ссылки на файл ajax.js, а также на файл JavaScript pizza.js, содержащий специфический код конкретного приложения

Этот пробел действительно важен. Без него некоторые браузеры не загрузят файл JavaScript, указанный в атрибуте src

А теперь объясните, почему я не должна использовать инструментарии Ajax, часто встречающиеся в Web. Разве они не обладают всевозможными дополнительными функциями?

Не используйте то, что не понимаете

В использовании инструментариев Ajax нет ничего плохого, особенно если они берут на себя выполнение многих вторичных, утомительных операций. Собственно, именно это делает файл ajax.js: мы взяли код, необходимый в каждом Ajax-приложении, и поместили его в файл, который снова и снова используется в наших программах.

Но это вовсе не значит, что вы должны схватить самый классный инструментарий, который вам попадется, запихнуть его в тег **<script>** и надеяться на лучшее. Откройте код JavaScript инструментария и разберитесь, что в нем происходит. После 400 страниц асинхронного программирования это будет не так уж трудно!

text-utils.js

Разобравшись с асинхронным кодом JavaScript, можно переходить к коду DOM. Модель Document Object Model была описана в главе 4, но файл text-utils.js использовался несколькими главами ранее... еще до того, как вы стали экспертом по DOM. Далее приводится код text-utils.js с некоторыми замечаниями по поводу его работы.

```
function replaceText(el, text) {
  if (el != null) {
    clearText(el);
    var newNode = document.createTextNode(text);
    el.appendChild(newNode);
  }
}

function clearText(el) {
  if (el != null) {
    if (el.childNodes) {
      for (var i = 0; i < el.childNodes.length; i++) {
        var childNode = el.childNodes[i];
        el.removeChild(childNode);
      }
    }
  }
}

function getText(el) {
  var text = "";
  if (el != null) {
    if (el.childNodes) {
      for (var i = 0; i < el.childNodes.length; i++) {
        var childNode = el.childNodes[i];
        if (childNode.nodeValue != null) {
          text = text + childNode.nodeValue;
        }
      }
    }
  }
  return text;
}
```

Функция replaceText() получает элемент и заменяет все его содержимое тем текстом, который вы укажете

Сначала мы используем функцию clearText(), чтобы избавиться от всех существующих потомков

Затем мы создаем новый текстовый узел при помощи объекта document, и присоединяем его к потомкам элемента

Функция clearText() удаляет всех потомков элемента, передаваемого функции

В цикле перебираем потомков и удаляем каждого из них

Осторожно! Цикл удаляет всех потомков, даже если они не являются текстовыми узлами

Функция getText() возвращает текст элемента, переданного при вызове функции

В цикле перебираем всех потомков указанного элемента

У узлов элементов и других нетекстовых узлов значение nodeValue равно null

Команда берет свойство nodeValue каждого текстового узла и прибавляет его к текущему значению text

После завершения функции эта переменная содержит весь текст элемента

Использование text-utils.js

Чтобы использовать файл text-utils.js, включите тег <script> в секцию <head> кода HTML своей веб-страницы, как это было сделано для файла ajax.js:

```
<head>
 <title>Ajax-powered Coffee Maker</title>
 <link rel="stylesheet" type="text/css" href="coffee.css" />
 <script type="text/javascript" src="ajax.js"> </script>
 <script type="text/javascript" src="text-utils.js"> </script>
 <script type="text/javascript" src="coffee.js"> </script>
</head>
```

Не забывайте вставлять пробел между открывающим и закрывающим тегом <script>, чтобы внешний код JavaScript загружался во всех браузерах

Бретт Маклафлин — гитарист, который [illegible] не желает признать [illegible] заработать себе на жизнь [illegible]

[illegible] Head First [illegible] Java 5.0 [illegible] Home Theater Hacks [illegible] Data Binding [illegible] Enterprise Applications [illegible] Head First [illegible]

Если вы действительно хотите чему-то научиться, и притом научиться быстрее и глубже, подумайте над тем, как вы думаете. Осознайте суть осознания. Научитесь учиться.

Большинство читателей не посещали курсы метапознания или теории обучения. От нас хотели, чтобы мы учились, но редко учили нас учиться.

Но раз уж вы держите в руках эту книгу, предполагается, что вы хотите изучить A[illegible] И скорее всего, вам не захочется тратить много времени. А поскольку речь пой[illegible] о разработке приложений, прочитанное придется запоминать — но для этого материал нужно сначала понять. Чтобы извлечь максимум пользы из этой книги, научитесь управлять работой своего мозга. Фокус в том, чтобы ваш мозг воспр[illegible] изучаемый материал как нечто Очень Важное. Нечто критичное для вашего су[illegible]ствования. Есть два способа обучения: один медленный и скучный, другой быст[illegible] и эффективный.

Медленный способ основан на обычном повторении. Быстрый — на выполнении любых операций, стимулирующих мозговую деятельность.

- Мы использовали рисунки, потому что мозг настроен на прием визуальной информации, а не текста.
- Мы использовали повторение, передавая одну и ту же мысль разными способами и по разным каналам.
- Мы использовали необычные концепции и рисунки, потому что мозг хорошо улавливает все новое.
- Мы использовали личный разговорный стиль, потому что наш мозг действует более активно, когда считает, что вы участвуете в беседе, а не являетесь пассивным слушателем.
- Мы включили в книгу свыше сорока упражнений, потому что наш мозг лучше изучает и запоминает, когда вы что-то делаете, а не просто читаете.
- Мы использовали разные стили обучения, потому что вы можете предпочита[illegible] пошаговые описания, кто-то другой захочет сначала представить общую картину, а третьему достаточно увидеть пример кода.
- Мы включили информацию для обоих полушарий мозга, потому что чем акти[illegible] задействован ваш мозг, тем больше вероятность того, что вы что-то узнаете и запомните, и тем дольше сможете концентрироваться на изучаемой теме.
- Мы приводим истории и упражнения, представляющие различные точки зрен[illegible] потому что ваш мозг лучше усваивает материал, когда ему приходится оценивать и делать выводы.
- Мы задаем читателю вопросы, на которые не всегда существует простой, однозначный ответ, потому что ваш мозг лучше учится и запоминает за работ[illegible]
- Мы использовали людей в своих историях, примерах, рисунках и т. д., потому что [illegible] вы [illegible] человек.

Заказ книг:

197198, Санкт-Петербург, а/я 619
тел.: (812) 703-73-74, postbook@piter.com
61093, Харьков-93, а/я 9130
тел.: (057) 712-27-05, piter@kharkov.piter.com

www.piter.com
вся информация о книгах и веб-магазин

ПИТЕР

ISBN 978-5-91180-322-3

9785911803223

O'REILLY