КРИСТИАН ДАРИ
БОГДАН БРИНЗАРЕ
ФИЛИП ЧЕРЧЕЗ-ТОЗА
МИХАЙ БУСИКА

AJAX и PHP

РАЗРАБОТКА ДИНАМИЧЕСКИХ ВЕБ-ПРИЛОЖЕНИЙ

HIGH TECH

СИМВОЛ®

AJAX and PHP

Building Responsive Web Applications

Cristian Darie, Bogdan Brinzarea,
Filip Chereches-Tosa, Mihai Bucica

HIGH TECH

AJAX и PHP

Разработка динамических веб-приложений

Кристиан Дари, Богдан Бринзаре,
Филип Черчез-Тоза, Михай Бусика

СИМВОЛ®

Санкт-Петербург — Москва
2007

Серия «High tech»

Кристиан Дари, Богдан Бринзаре,
Филип Черчез-Тоза, Михай Бусика

AJAX и PHP: разработка динамических веб-приложений

Перевод А. Киселева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*О. Цилюрик*
Редактор	*В. Овчинников*
Художник	*В. Гренда*
Корректор	*И. Губченко*
Верстка	*Д. Орлова*

Дари К., Бринзаре Б., Черчез-Тоза Ф., Бусика М.
AJAX и PHP: разработка динамических веб-приложений. – СПб.: Символ-Плюс, 2007. – 336 с., ил.
ISBN 5-93286-077-4

Книга «AJAX и PHP: разработка динамических веб-приложений» – самый удобный и полезный ресурс, который поможет вам войти в захватывающий мир AJAX. Вы научитесь писать более эффективные веб-приложения на PHP за счет использования всего спектра возможностей технологий AJAX. Применение AJAX в связке с PHP и MySQL описывается на многочисленных примерах, которые читатель сможет использовать в собственных проектах. Рассмотрены следующие темы: верификация заполнения форм на стороне сервера; чат-приложение, основанное на технологии AJAX; реализация подсказок и функции автодополнения; построение диаграмм в реальном времени средствами SVG; настраиваемые и редактируемые таблицы на основе баз данных; реализация RSS-агрегатора; построение сортируемых списков с поддержкой механизма drag-and-drop. Издание предназначено тем, кто владеет базовыми знаниями PHP, XML, JavaScript и MySQL и хочет узнать все тонкости функционирования AJAX и взаимодействия составляющих этой технологии.

ISBN–13: 978–5–93286–077–9
ISBN–10: 5–93286–077–4
ISBN 1–904811–82–5 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 30.03.2007. Формат 70x100 1/16 . Печать офсетная.
Объем 21 печ. л. Доп. тираж 2000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Об авторах 7

Предисловие 9

1. AJAX и будущее веб-приложений 15

Предоставление функциональности через Интернет 17
Разработка веб-сайтов до 1990 года 19
Что такое AJAX 24
Создание простого приложения на основе AJAX и PHP 29
Подведение итогов 42

2. Клиентские технологии на основе JavaScript 43

JavaScript и объектная модель документа (DOM) 44
События в JavaScript и DOM 48
И еще о DOM 52
JavaScript, DOM и CSS 55
Использование объекта XMLHttpRequest 59
Работа со структурой XML 75
Подведение итогов 85

3. Технологии, применяемые на стороне сервера: PHP и MySQL 86

PHP и DOM 86
Передача параметров и обработка ошибок в PHP 93
Соединение с удаленным сервером и безопасность сценариев JavaScript 104
Доверенный сценарий на стороне сервера 110
Основные принципы выполнения повторяющихся асинхронных запросов 118
Работа с MySQL 129
Технология обертывания и разделения функциональности 140
Подведение итогов 153

4. Верификация заполнения форм в AJAX 154

Реализация проверки правильности в AJAX 155
Подведение итогов 182

5. Чат AJAX 183

Введение в технологию прямого общения по сети 183
Реализация чата на основе технологии AJAX 185
Подведение итогов 206

6. Подсказки и функция автодополнения в AJAX 207

Введение в подсказки и функцию автодополнения на базе AJAX 208
Реализация подсказок и функции автодополнения средствами AJAX 209
Подведение итогов 235

7. Построение диаграмм в реальном времени средствами SVG и AJAX 236

Реализация построения диаграмм в реальном времени 237
Подведение итогов 251

8. Таблицы в AJAX 252

Реализация таблиц данных на стороне клиента средствами AJAX и XSLT 253
Подведение итогов 275

9. Чтение лент новостей в AJAX 276

Работаем с RSS 276
Структура документа RSS 277
Реализация чтения лент RSS с помощью технологии AJAX 279
Подведение итогов 292

10. Технология drag-and-drop в AJAX 293

Применение механизма перетаскивания во Всемирной паутине 293
Создание приложения с поддержкой механизма перетаскивания 295
Подведение итогов 313

A. Подготовка рабочего окружения 314

Алфавитный указатель 326

Об авторах

Кристиан Дари (Cristian Darie) – программист, имеющий опыт работы с широким кругом современных технологий, автор многочисленных книг, включая популярную серию «Beginning E-Commerce». Компьютерами начал заниматься, как только научился нажимать на клавиши. Впервые вкус успеха в области программирования почувствовал, когда в возрасте 12 лет выиграл главный приз в конкурсе программистов. С тех пор Кристиан достиг многого, а сейчас он занимается изучением архитектур распределенных приложений в рамках своей кандидатской диссертации. Ему всегда нравилось получать отзывы о своих книгах, поэтому, если у вас выдастся свободная минутка, не стесняйтесь передать ему привет от себя. Связаться с Кристианом можно через его персональный веб-сайт по адресу: *www.cristiandarie.ro*.

Кристиан хотел бы выразить глубокую благодарность своим соавторам Богдану, Филиппу и Михаю, а также техническому редактору книги Джимми за их упорный труд, который они вложили в создание этой замечательной книги.

Богдан Бринзаре (Bogdan Brinzarea) имеет очень серьезную подготовку в области информатики. Он получил степень магистра на факультете автоматизации управления и вычислительной техники Бухарестского политехнического университета в Румынии и диплом аудитора факультета информатики политехнической школы Ecole Polytechnique в Париже.

В круг интересов Богдана входят вопросы разработки программного обеспечения для встраиваемых систем, распределенные и мобильные вычисления, а также новейшие веб-технологии. В настоящее время он работает специалистом по альтернативным каналам предоставления услуг в «Banca Romaneasca» (член банка «National Bank of Greece»), где отвечает за реализацию проекта предоставления банковских услуг через Интернет и координирует ряд других проектов, связанных с защитой приложений и применением новых технологий в области банковского дела.

Филип Черчез-Тоза (Filip Cherecheş-Toşa) – веб-программист, свято верящий в большое будущее веб-технологий. Карьеру начал в возрасте 9 лет, когда получил в подарок свой первый Commodore 64, оснащенный устройством внешней памяти на магнитной ленте.

У себя на родине, в Румынии, Филип руководит компанией eXigo (*www.exigo.org*), которая активно занимается разработкой веб-прило-

жений и веб-дизайном. В настоящее время учится в университете города Оради на факультете информатики и активно участвует в работе Румынского сообщества PHP-программистов (*www.phpromania.org*).

Михай Бусика (Mihai Bucica) начал заниматься программированием (а также участвовать и побеждать во многих конкурсах программистов) в возрасте 12 лет. Имея степень бакалавра информатики, выпускник факультета автоматизации управления и вычислительной техники Бухарестского политехнического университета Бусика занимается разработкой коммуникационного программного обеспечения для различных интернет-магазинов.

Несмотря на богатый опыт работы с многочисленными языками программирования и технологиями, Бусика предпочитает C++ и влюблен в слово LGPL. Михай также участвовал в создании книги «PHP 5 and MySQL E-Commerce». Связаться с ним можно через его персональный веб-сайт по адресу: *www.valentinbucica.ro*.

О рецензентах

Эмилиан Баланеску (Emilian Balanescu) – программист, имеющий опыт работы с множеством технологий, включая PHP, Java, .NET, PostgreSQL, MS SQL Server, MySQL и др. В настоящее время работает администратором беспроводной сети в компании accessNET International S.A. Romania, которая предоставляет услуги по обеспечению постоянного доступа к беспроводной сети общенационального масштаба, действующей в радиочастотном диапазоне. Его последняя разработка на данной должности – система управления сетью в реальном масштабе времени, созданная на базе идеологии AJAX (с использованием SNMP, Perl, PHP и PostgreSQL) и использующаяся для наладки, мониторинга и решения задач по диагностике и устранению неполадок. Связаться с ним можно по адресу *http://www.emilianbalanescu.ro*.

Пола Бадаску (Paula Badascu) обучается на третьем курсе Бухарестского политехнического университета, одного из самых известных технических университетов Румынии. Занимается изучением электроники, телекоммуникации и информационных технологий. В настоящее время Пола работает программистом-аналитиком в «NCH Advisors Romania», где занимается разработкой веб-приложений с использованием UML, OOP, PHP, SQL, JavaScript и CSS. Она внесла существенный вклад в анализ и разработку инфраструктуры, используемой для мониторинга состояния дел на фондовом рынке Румынии.

Предисловие

AJAX – это сложный феномен, который разные люди понимают по-разному. С точки зрения обычного пользователя AJAX – это дружественность и более высокая динамичность их любимых сайтов. А у веб-разработчиков AJAX ассоциируется с новыми знаниями и умениями, благодаря которым они могут создавать отличные веб-приложения с меньшими усилиями. В действительности все, что говорится об AJAX, звучит весьма неплохо.

В основе AJAX лежит сочетание различных технологий, которые позволяют уйти от необходимости повторно загружать страницы при переходе с одной страницы на другую. И это лишь первый шаг к тому, чтобы реализовать на веб-сайтах более развитые функциональные возможности, такие как проверка корректности данных в режиме реального времени, перетаскивание объектов на странице мышью и других, которые традиционно не были связаны с веб-приложениями. Компоненты, составляющие AJAX, сами по себе достаточно зрелые (например, объект XMLHttpRequest был разработан в Microsoft еще в 1999 г.), однако их роль в свете тенденций развития Всемирной паутины получает новое развитие, и этот сплав технологий еще очень изменится, прежде чем его можно будет применять для удовлетворения потребностей конечных пользователей. К моменту написания этой книги названию «AJAX» исполнился всего один год.

Разумеется, AJAX не надо считать панацеей от всех бед, ведь это лишь одна из созданных в последнее время технологий. Как и любая другая технология, AJAX может применяться неправильно или не там где надо. Кроме того, AJAX порождает и свои собственные проблемы – вам придется бороться с несовместимостью броузеров, а страницы, реализованные на основе AJAX, не работают без поддержки JavaScript, их нельзя поместить в закладки, а поисковые системы не всегда в состоянии разобраться с их содержимым. И вообще AJAX вызывает восторг далеко не у всех. Одни стремятся выстроить каркас приложения на основе JavaScript, другие вообще предпочитают обходиться без него. Однако большинство из вас согласится, что, как правило, истина лежит посередине.

В книге «AJAX и PHP: разработка динамических веб-приложений» мы избрали наиболее прагматичный и надежный подход к обучению, основанный на решении практических задач, с которыми, на наш взгляд, любой веб-разработчик рано или поздно столкнется. Мы пока-

жем, как избежать самых распространенных ошибок, как писать высокопроизводительный код AJAX и как создать функциональность, которую без труда можно интегрировать в работающие и будущие веб-приложения без пересборки проектов. Знания, полученные из этой книги, можно сразу применить на практике и внедрить их в свои веб-приложения, написанные на языке PHP.

Мы надеемся, что эта книга окажется для вас полезной и поможет в работе над проектами. Самые последние сведения, касающиеся ее, можно найти на сайте *http://ajaxphp.packtpub.com*.

Здесь выложены дополнительные главы и ресурсы, с которыми мы рекомендуем ознакомиться.

Краткое содержание книги

Глава 1 *«AJAX и будущее веб-приложений»* представляет собой начальный экскурс в мир AJAX и его возможности, которые открываются перед веб-разработчиками и компаниями. В этой главе вы создадите свою первую веб-страницу на основе AJAX.

В главе 2 *«Клиентские технологии на основе JavaScript»* рассказывается о технологиях, применяемых при создании клиентских частей веб-приложений AJAX, основанных на JavaScript, DOM, объекте XMLHttpRequest и XML. Данная глава не претендует на роль всеобъемлющего справочного руководства по всем этим технологиям, однако после ее прочтения вы сможете построить на их основе крепкий фундамент своих будущих веб-приложений.

Глава 3 *«Технологии, применяемые на стороне сервера: PHP и MySQL»* завершает теоретическую часть книги и рассказывает о том, как создаются сценарии на стороне сервера, предназначенные для взаимодействия с клиентской частью AJAX. Здесь рассмотрены различные методы решения самых распространенных задач, включая проблемы безопасности JavaScript и обработку ошибок.

Глава 4 *«Верификация заполнения форм в AJAX»* проведет вас через создание современной, динамической и безопасной системы проверки правильности заполнения форм, которая реализует как проверку в режиме реального времени, так и проверку на стороне сервера после отправки формы.

Глава 5 *«Чат AJAX»* представит пример простейшего работающего веб-приложения, предназначенного для организации прямого общения по сети, в котором задействуется только код AJAX и не применяются Java-апплеты, Flash или какие-либо другие специализированные библиотеки, широко применяемые сейчас при разработке подобных приложений.

Глава 6 *«Подсказки и функция автодополнения в AJAX»* продемонстрирует пример реализации функциональности, напоминающей под-

сказки Google, которая поможет вам быстро отыскать требуемую функцию языка PHP и перенаправит на официальную справочную страничку по этой функции.

Глава 7 *«Построение диаграмм в реальном времени средствами SVG и AJAX»* расскажет, как реализовать создание диаграмм в режиме реального времени на основе решений, предлагаемых AJAX и SVG. SVG (Scalable Vector Graphics, масштабируемая векторная графика) – это язык создания графических изображений, который может применяться для вычерчивания фигур и вывода текста.

Глава 8 *«Таблицы в AJAX»* научит, как использовать преимущества AJAX для представления данных в табличной форме. Здесь вы также научитесь производить разбор XML-документов с помощью XSLT для извлечения табличных данных.

В главе 9 *«Чтение лент новостей в AJAX»* показан пример простейшего веб-приложения, считывающего информацию из лент новостей, реализованного на базе XML, XSLT и SimpleXML (библиотеки языка PHP).

Глава 10 *«Технология drag-and-drop в AJAX»* продемонстрирует использование платформы script.aculo.us для построения простейших списков элементов с возможностью перетаскивания мышью.

Приложение А *«Подготовка рабочего окружения»* показывает, как установить и сконфигурировать необходимое программное обеспечение, в состав которого входят: Apache, PHP, MySQL, phpMyAdmin. Примеры этой книги работают, только если рабочее окружение и базы данных настроены так, как показано в этом приложении.

На сайте *http://ajaxphp.packtpub.com* можно найти все примеры, приведенные в тексте книги.

Что потребуется для работы с этой книгой

PHP 5, веб-сервер и сервер баз данных. Примеры программ тестировались в различных окружениях, но главным образом с Apache 2 в качестве веб-сервера и MySQL 4.1 и MySQL 5 в качестве серверов баз данных.

Можно выбрать другой веб-сервер или продукт для работы с базами данных, но в этом случае мы не ручаемся за работоспособность процедур, описываемых в книге. Очень важно, чтобы версия PHP была не ниже 5, поскольку не все объектно-ориентированные особенности, задействованные нами, поддерживаются в более ранних версиях.

Дополнительные сведения по настройке рабочего окружения вы найдете в приложении А. Если в системе уже установлено все необходимое программное обеспечение, то вам останется только прочитать заключительную часть приложения, в которой рассказывается, как создать базу данных, фигурирующую в примерах этой книги.

Соглашения

Вам встретятся различные способы оформления текста, призванные выделить различные типы информации. Вот несколько примеров такого выделения и описание их назначения.

Исходные тексты программ могут оформляться тремя способами. Ключевые слова в тексте выделяются шрифтом, например: «Мы можем подключать содержимое других файлов с помощью директивы `include`».

Блоки кода форматируются так:

```
// функция обращается к серверу с помощью объекта XMLHttpRequest
function process()
{
    // получить имя, введенное пользователем при заполнении формы
    name = document.getElementById("myName").value;
    // запустить сценарий quickstart.php на сервере
    xmlHttp.open("GET", "quickstart.php?name=" + name, false);
    // выполнить синхронный запрос сервера
    xmlHttp.send(null);
    // прочитать ответ
    handleServerResponse();
}
```

Для того чтобы привлечь ваше внимание к отдельным строкам исходного кода в блоке, они выделяются жирным шрифтом:

```
// функция обращается к серверу с помощью объекта XMLHttpRequest
function process()
{
    // получить имя, введенное пользователем при заполнении формы
    name = document.getElementById("myName").value;
    // запустить сценарий quickstart.php на сервере
    xmlHttp.open("GET", "quickstart.php?name=" + name, false);
    // выполнить синхронный запрос сервера
    xmlHttp.send(null);
    // прочитать ответ
    handleServerResponse();
}
```

Текст, который должен вводиться в командной строке, форматируется так:

```
./configure --prefix=/usr/local/apache2 --enable-so --enable-ssl --withssl
--enable-auth-digest
```

Новые термины и **важные понятия** выделяются жирным шрифтом. Текст, который вы увидите на экране, в меню или в диалогах, в тексте книги заключается в кавычки, например: «щелкните по кнопке «Далее», чтобы перейти к следующему экрану».

Примечание

Предупреждения или важные примечания будут выглядеть так.

Совет

А так будут оформляться советы и подсказки.

Обратная связь с читателями

Мы всегда рады получать отзывы от наших читателей. Дайте нам знать, что вы думаете об этой книге, что вам понравилось или, наоборот, что не понравилось. Отзывы читателей имеют для нас очень большое значение, т. к. они помогают издавать действительно необходимые книги.

Свои отзывы отправляйте по адресу *feedback@packtpub.com*, при этом в поле «Тема» не забудьте указать название книги.

Заполнив и отправив форму SUGGEST A TITLE на сайте *www.packtpub.com*, можно предложить тему для новой книги. Предложение можно отправить и по электронной почте на адрес *suggest@packtpub.com*.

Если вы обладаете необходимыми знаниями и хотите написать или передать для публикации свою книгу, обращайтесь по адресу *www.packtpub.com/authors*.

Поддержка покупателей

Покупателям книг, выпущенных издательством Packt, мы делаем дополнительное предложение, чтобы они могли извлечь максимум пользы из своей покупки.

Загрузка исходных текстов примеров из этой книги

Чтобы загрузить исходные тексты примеров или дополнительные ресурсы, имеющие отношение к этой книге, надо зайти на страницу *http://www.packtpub.com/support* и выбрать название книги из списка. После того как вы сделаете выбор, откроется список файлов, доступных для скачивания.

Примечание

Скачиваемые файлы содержат инструкции по их применению.

Список опечаток

Мы приложили максимум усилий, чтобы избежать опечаток и неточностей, но ошибки все равно могут встретиться. Если вы найдете ошибку в тексте или в исходном коде примера и сообщите нам об этом, мы будем вам весьма признательны. Тем самым вы убережете других читате-

лей от разочарований и поможете улучшить последующие издания этой книги. Сообщения об ошибках можно оставить по адресу *http://www.packtpub.com/support*, где надо щелкнуть по ссылке `Submit Errata` и ввести подробное описание найденной ошибки. Если ваше замечание подтвердится, оно будет принято и добавлено в список известных опечаток. Вы можете просмотреть список известных опечаток, щелкнув по названию книги на странице *http://www.packtpub.com/support*.

Вопросы

Вопросы, так или иначе связанные с книгой, можно задать, написав нам по адресу *questions@packtpub.com*, и мы приложим все усилия, чтобы ответить на них.

1

AJAX и будущее веб-приложений

«Компьютер, нарисуй робота!», – сказал мой маленький кузен компьютеру, когда впервые увидел его. (Поскольку я научил компьютер не реагировать на приказы посторонних, то он, само собой, проигнорировал эту команду.) Если вы похожи на меня, то первая мысль, которая могла прийти вам в голову, была бы: «как глупо» или «как забавно», и это было бы ошибкой. Наш обученный и сформировавшийся мозг до определенной степени знает, как работать с компьютером. Люди обучаются приспосабливать компьютеры под себя, чтобы компенсировать отсутствие у компьютеров возможности понимать человеческий язык. (С другой стороны, возможности людей приспосабливаться также весьма ограничены, но это уже другая история.)

Этот забавный случай наглядно показывает, что в работе с компьютером люди руководствуются инстинктом. В идеале той речевой команды должно было бы хватить, чтобы компьютер порадовал моего кузена. В последние годы был достигнут значительный успех в развитии технологий, повышающих дружественность компьютеров по отношению к пользователю, но впереди еще долгий путь, который нам предстоит пройти, прежде чем мы получим по-настоящему интеллектуальные компьютеры. А пока люди должны обучаться взаимодействию с компьютерами, причем некоторые даже влюбляются в черный экран с крошечным приглашением к вводу команды на нем.

Неслучайно многие навыки работы с компьютером вырабатываются программами с пользовательским интерфейсом, который обеспечивает интуитивно понятное (и удобное) взаимодействие с человеком. Этим, наверное, можно объяснить популярность правой кнопки мыши, восхищение такими функциональными возможностями, как перетаскивание объектов мышью, или тем, что, введя слово в простое текстовое поле, можно выполнить его поиск по всему Интернету за 0,1 секунды (или что-то около того). Индустрия программного обеспечения (по крайней мере, та ее часть, которая приносит прибыль) видит,

анализирует и изучает ситуацию. Сейчас рынок заполнен программами со сверкающими кнопками, значками, окошками и разнообразными мастерами, и люди *платят за них большие деньги*.

Главное, что поняли в индустрии ПО, – это что *его удобство в использовании и доступность* представляют собой эквивалент мощного двигателя в красном спортивном автомобиле. Это замечательно, когда точка зрения бизнеса совпадает с точкой зрения обычного пользователя, потому что прибыль более или менее пропорциональна тому, насколько удовлетворен пользователь.

Мы планируем быть очень практичными и краткими в этой книге, но прежде чем обратиться к вашему любимому занятию (написанию кода), отвлечемся немного, чтобы вспомнить, что и почему мы делаем. Мы влюблены в звук, извлекаемый при нажатии на клавиши, поэтому мы легко забываем, что самая главная причина существования технологий заключается в служении людям и в том, чтобы делать их жизнь интереснее, а работу – эффективнее.

Для создания конечных приложений очень важно понять образ мышления человека. Мы еще далеки от этой цели, но уже понимаем, что конечные пользователи нуждаются в интуитивно понятных интерфейсах. Их мало интересует, с какой операционной системой они работают, пока функциональные возможности, которые они имеют, совпадают с тем, что они *ожидают*. Это очень важный момент, о котором надо помнить, тогда как многие программисты, работая с конечными пользователями, продолжают мыслить чисто техническими категориями (хотя в типичной группе разработчиков программист не общается с пользователем напрямую). Если вы не согласны с этим утверждением, попробуйте вспомнить, сколько раз вы произнесли слова *база данных* в разговоре с неспециалистом.

В результате наблюдений за привычками и потребностями людей, работающих с компьютерными системами, родился термин **юзабилити** (удобство в использовании), обозначающий *искусство* удовлетворять ожидания пользователей от интерфейса, понимать характер их работы и соответственно создавать приложения.

Исторически сложилось, что методы улучшения юзабилити применялись в основном к **обычным приложениям** по той простой причине, что они требовали наличия инструментальных средств, не доступных для **веб-приложений**. Однако по мере развития Интернета рос и потенциал технологий, позволяющих сделать это.

Современные сетевые технологии не только дают возможность повышать качество работы в Интернете, но и позволяют совершенствовать приложения, разрабатываемые специально для интрасетей. Наличие сайтов, дружественных к пользователям, очень важно для сетевого бизнеса, поскольку Интернет никогда не спит, а клиенты зачастую уходят на другой сайт только потому, что он выглядит лучше или *кажется* бо-

лее динамичным. В то же самое время возможность создавать дружественные веб-интерфейсы позволяет внедрять в интрасети программные решения, которые ранее создавались как обычные приложения.

Создание дружественного ПО всегда было проще в случае обычных приложений, чем веб-приложений, просто потому, что Интернет проектировался как средство доставки текста, изображений и несложной функциональности. За последние несколько лет проблема дружественности обострилась в связи с ростом объемов услуг, поставляемых через Интернет.

Для решения этой проблемы были придуманы (и они продолжают появляться) новые технологии, которые позволяют создавать яркие, удобные и мощные веб-приложения. В качестве примеров можно привести получившие широкую известность **Java-апплеты** и **Macromedia Flash**, которые требуют установки в броузеры дополнительных библиотек.

Предоставление функциональности через Интернет

Веб-приложения – это приложения, функциональные возможности которых обеспечиваются сервером и доставляются конечным пользователям по сети, такой как Интернет или интранет. Конечные пользователи запускают веб-приложения с помощью **тонкого клиента** (веб-броузера), который знает, как обрабатывать и отображать данные, полученные от сервера. В противоположность им, обычные приложения являют собой пример **толстого клиента**, который выполняет большую часть работы.[1]

Прогрессирующее развитие веб-приложений позволяет надеяться, что в один прекрасный день они будут выглядеть и вести себя как их более зрелые (и более мощные) родственники – обычные приложения. Поведение любого ПО, предназначенного для взаимодействия с людьми,

[1] Здесь авторы сознательно несколько упрощают картину, сводя ее к двум градациям: «толстый» и «тонкий» клиенты. На самом деле таких крупных градаций три (таков и хронологический порядок их развития): а) «монолитные» приложения, выполняющие всю работу автономно (говорить здесь о клиентской части бессмысленно, если не считать сервером такого приложения операционную систему, но это уже масло масленое; самый известный пример этого класса – MS Office); б) «толстый» клиент, пользующийся сервисами локального сервера (самые очевидные примеры – это приложения работы с базами данных, например все бухгалтерские системы); в) «тонкий» клиент, взаимодействующий с удаленным сервером, реализующим всю функциональность системы, по специализированному сетевому протоколу (почти всегда под таким протоколом имеют в виду HTTP, но таким же протоколом является графический протокол X и другие). – *Примеч. науч. ред.*

стало еще важнее, чем было до сих пор, потому что сейчас уровень подготовки пользователей варьируется в гораздо более широких пределах, чем раньше, когда они были технически грамотнее. Теперь, например, вы должны предоставить Синди, начальнику отдела сбыта, хорошо оформленные отчеты и обеспечить Дейва, сотрудника того же отдела, удобными формами ввода данных.

Поскольку удовлетворение нужд конечного пользователя – самое важное требование, необходимо создавать такие приложения, которые будут удовлетворять всех пользователей, взаимодействующих с ними. Что касается веб-приложений, их процесс движения к совершенству будет считаться завершенным тогда, когда интерфейс и поведение приложений не будут свидетельствовать о том, где они исполняются – на локальном ли компьютере или на удаленном сервере, поставляя результаты своей работы через сеть. Предоставление удобных интерфейсов через сеть всегда было делом проблематичным просто потому, что некоторые функциональные возможности, которыми, как правило, обладают обычные приложения, например перетаскивание объектов мышью или одновременное выполнение множества задач в одном и том же окне, были невозможны.

Еще одна трудность, с которой постоянно сталкиваются разработчики веб-приложений, – это **стандартизация**. Сейчас необходимо, чтобы все, что доступно через Интернет, было проверено на совместимость, по крайней мере, с двумя или тремя броузерами с целью гарантировать, что все посетители смогут извлечь максимум пользы из вашего сайта.

Преимущества веб-приложений

Да, попытки предоставить функциональные возможности через Интернет сопряжены с массой трудностей. Но почему тогда не попробовать сделать то же самое посредством обычных приложений? Это, конечно, верно, но даже проблемы с дружественностью веб-приложений не помешали им приобрести небывалую популярность, потому что они предлагают массу важных преимуществ, которые отсутствуют в обычных приложениях.

- **Установка веб-приложений проще и обходится дешевле.** Благодаря использованию веб-приложений предприятия могут снизить затраты на содержание отделов ИТ, которые отвечают за установку программного обеспечения на компьютеры пользователей. В этом случае пользователю необходимы лишь компьютер с броузером и соединение с Интернетом или корпоративной сетью.
- **Обновление веб-приложений проще и обходится дешевле.** Стоимость обслуживания ПО всегда имела большое значение. Обновление ПО в чем-то схоже с его установкой, поэтому упомянутые выше преимущества имеются и в этой ситуации. Как только приложение будет обновлено на сервере, все сразу же смогут работать с новой версией.

- **Веб-приложения предъявляют более гибкие требования к конечному пользователю.** Вам достаточно будет установить веб-приложение на сервере, работающем под управлением любой современной ОС, и вы сможете пользоваться им через Интернет или интранет на любом компьютере, работающим под управлением Mac OS, Windows, Linux или какой-либо другой ОС. Если приложение не будет содержать ошибок, оно одинаково хорошо будет работать в любом современном веб-броузере, будь то Internet Explorer, Mozilla Firefox, Opera или Safari.
- **Веб-приложения облегчают организацию централизованного хранения данных.** Если необходимо обращаться к одним и тем же данным из разных мест, то намного проще организовать их хранение в одном месте, вместо того чтобы разбрасывать по разным базам данных. Благодаря этому отпадет необходимость выполнения операций синхронизации и повысится степень их защищенности.

Далее в этой книге мы будем рассматривать применение современных веб-технологий для создания качественных веб-приложений, максимально задействующих возможности, предлагаемые Всемирной паутиной. Но прежде чем углубиться в детали, мы предпримем *краткий* экскурс в историю развития.

Разработка веб-сайтов до 1990 года

Интернет появился много раньше, но все-таки именно в 1991 г. был изобретен **протокол передачи гипертекстовой информации (HyperText Transfer Protocol – HTTP)**, который до сих пор применяется для передачи данных через Интернет. В своей начальной версии он делал лишь чуть больше, чем просто открывал/закрывал соединение. Последние версии HTTP (версия 1.0 появилась в 1996 г., а версия 1.1 – в 1999) стали тем протоколом, который мы знаем и применяем.

HTTP и HTML

Протокол HTTP поддерживается всеми веб-броузерами и прекрасно справляется с возложенными на него обязанностями, которые заключаются в доставке данных через Интернет. Всякий раз, когда вы в своем любимом броузере запрашиваете веб-страницу, это предполагает использование протокола HTTP. Например, если в адресной строке броузера Firefox ввести *www.mozilla.org*, то по умолчанию он предположит, что вы имели в виду *http://www.mozilla.org*.

Стандартный тип документа в Интернете, который веб-броузеры могут понять, проанализировать и отобразить, – это документ, написанный на **языке разметки гипертекста (HyperText Markup Language – HTML).** HTML – это язык описания содержимого и форматирования документа, который в основном состоит из статического текста и изображений. HTML не был предназначен для создания сложных веб-приложений

с изменяющимся содержимым или дружественными интерфейсами. Для того чтобы через HTTP получить другую страницу HTML, ее надо полностью загрузить, а сама она должна существовать в указанном месте в виде статического документа еще до выполнения запроса. Очевидно, что эти ограничения не способствуют созданию чего-нибудь интересного.

Тем не менее HTTP и HTML по-прежнему остаются удачной парой, которая понятна как веб-серверам, так и веб-клиентам (броузерам). Как мы уже знаем, они составляют фундамент современного Интернета. На рис. 1.1 показан простейший пример взаимодействия клиента и сервера, когда пользователь запрашивает веб-страницу из Интернета с помощью протокола HTTP.

Рис. 1.1. *Простой запрос HTTP*

Примечание

Необходимо запомнить три вещи:

1. Взаимодействия по протоколу HTTP всегда осуществляются между веб-клиентом (ПО, выполняющим запрос, таким как веб-броузер) и веб-сервером (ПО, отвечающим на запрос, таким как Apache или IIS). С этого момента везде в книге мы будем писать просто «клиент», подразумевая веб-клиент, и просто «сервер», подразумевая веб-сервер.
2. Пользователь – человек использующий программу-клиент.
3. Несмотря на то, что протокол HTTP (и его безопасная версия **HTTPS**) едва ли не самый важный, тем не менее это не единственный протокол, применяемый в Интернете. Различные типы веб-серверов используют различные протоколы для решения разнообразных задач, как правило, не связанных с простым просмотром сети. Чаще всего в этой книге мы будем говорить о протоколе HTTP, и, говоря «веб-запрос», мы будем подразумевать запрос с использованием протокола HTTP, в противном случае мы будем явно указывать название протокола.

Безусловно, комбинация HTTP–HTML предоставляет весьма ограниченные возможности, поскольку пользователи могут получать из Интернета только статическую информацию (страницы HTML). Чтобы восполнить нехватку функциональных возможностей, были разработаны различные дополнительные технологии.

Несмотря на то, что все веб-запросы, о которых мы будем говорить далее, по-прежнему применяют для передачи информации протокол HTTP, сама эта информация может генерироваться веб-сервером динамически (скажем, извлекаться из базы данных). Кроме того, она может содержать не только текст на языке HTML, позволяя клиенту не только просматривать содержимое статических страниц, но и производить некоторые дополнительные действия.

Технологии, расширяющие возможности Интернета, образуют две основные категории:

- **Клиентские технологии**, позволяющие веб-клиенту не только отображать статические документы, но и делать более интересные вещи. Как правило, эти технологии дополняют, а не замещают язык HTML.
- **Серверные технологии**, предоставляющие возможность перенести логику приложения на сторону сервера с целью создавать веб-страницы на лету.

PHP и другие серверные технологии

Серверные веб-технологии позволяют веб-серверу не просто возвращать запрошенные HTML-файлы, но и выполнять дополнительные действия, например сложные вычисления, запускать объектно-ориентированные программы, работать с базами данных и многое другое.

Только представьте себе, какой объем данных должен обработать Amazon, чтобы сгенерировать индивидуальные рекомендации для каждого посетителя. Или Google, который выполняет поиск информации по гигантской базе данных, чтобы удовлетворить запрос. Да, технологии обработки данных на стороне сервера стали механизмом, запустившим революцию во Всемирной паутине, и причиной, по которой Интернет в нынешнем его виде имеет такой успех.

Очень важно понимать, что, независимо от происходящего на стороне сервера, клиент должен получить ответ на том языке, который ему понятен (что вполне очевидно), например, на языке HTML, который имеет массу ограничений, о чем мы уже говорили.

PHP – это одна из технологий, применяемых для реализации логики приложений на стороне сервера. Введение в PHP будет дано в главе 3. Мы будем использовать PHP в процессе создания и изучения примеров **AJAX**. Тем не менее вы должны знать, что у PHP много конкурентов, таких как **ASP.NET (Active Server Pages** – активные серверные страницы, веб-технология, разработанная в Microsoft), **Java Server Page (JSP** – серверные страницы на языке Java), **Perl**, **ColdFusion**, **Ruby on Rails** и других. Каждая из этих технологий предоставляет свои возможности по реализации функциональности на стороне сервера.

PHP – это не только серверная технология, но и язык, на котором программисты могут писать сценарии. На рис. 1.2 показан пример запро-

са страницы index.php. На этот раз вместо передачи клиенту содержимого файла index.php сервер исполняет сценарий index.php и возвращает полученные результаты. Эти результаты должны быть размечены по правилам HTML или любого другого языка, понятного клиенту.

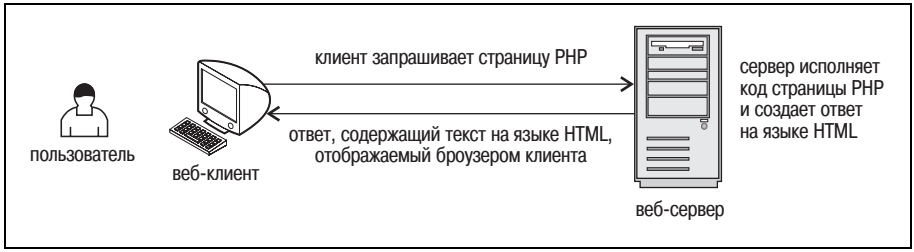

Рис. 1.2. *Клиент запрашивает страницу PHP*

Обычно на стороне сервера также должен размещаться **сервер баз данных**, который будет управлять данными. В примерах из этой книги будет фигурировать **MySQL**, но концепция в целом применима к любому другому серверу баз данных. Основы работы с базами данных и PHP вы изучите в главе 3.

Однако даже при использовании PHP, который может выполнять сложные запросы к базе данных, зависящие от ситуации, броузер пользователя по-прежнему будет статически отображать скучные, не вызывающие интереса веб-документы.

Потребность в более широких функциональных возможностях удовлетворяется отдельной категорией веб-технологий, которые называются клиентскими технологиями. Современные броузеры способны анализировать не только простой HTML. Давайте посмотрим как.

JavaScript и другие клиентские технологии

Различные клиентские технологии отличаются друг от друга в первую очередь способом, которым они загружаются и исполняются веб-клиентом. **JavaScript** – это язык сценариев, код которых в виде простого текста может быть внедрен прямо в страницы на языке HTML. Страница HTML, запрашиваемая клиентом, может содержать сценарии JavaScript. Все современные броузеры поддерживают JavaScript и не требуют от пользователя установки дополнительных компонентов в систему.

JavaScript – это самостоятельный язык программирования (теоретически он не связан с разработкой веб-приложений). Он поддерживается большинством веб-клиентов на любой платформе и обладает некоторыми объектно-ориентированными возможностями. Язык JavaScript относится к интерпретирующим и потому не годится для разработки приложений с интенсивными вычислениями или драйверов устройств и должен целиком доставляться броузеру клиента для последующей интерпретации. Кроме того, он испытывает определенные

проблемы с безопасностью, но при использовании в составе веб-страниц прекрасно справляется с возложенными на него задачами.

Благодаря JavaScript разработчики наконец получили возможность создавать веб-страницы с визуальными эффектами и способностью проверять правильность заполнения форм, избавив тем самым пользователей от необходимости повторно загружать всю страницу (с потерей всех введенных ранее данных, кстати), если они забыли указать какую-либо информацию (например, пароль или номер кредитной карточки) или в случае ошибки. Однако несмотря на имеющийся потенциал, JavaScript никогда не применялся должным образом, чтобы сделать веб-интерфейс действительно дружественным к пользователю, как в обычных приложениях.

Среди других клиентских технологий, наделенных функциональными возможностями, можно назвать Java-апплеты и Macromedia Flash. Java-апплеты пишутся на весьма популярном и мощном языке программирования Java и исполняются **виртуальной Java-машиной (Virtual Java Machine – JVM)**, которую необходимо отдельно устанавливать в систему. Без сомнения, Java-апплеты позволяют создавать более сложные проекты, но применительно к веб-приложениям они уже потеряли свою былую популярность, поскольку потребляют значительное количество системных ресурсов. Иногда даже сам запуск их может занимать значительное время, и вообще они слишком тяжеловесны для простых и нетребовательных веб-приложений.

Технология Macromedia Flash обладает весьма широкими возможностями по созданию графических и анимационных эффектов и фактически стала стандартом для разработки подобных веб-приложений. Macromedia Flash также требует установки *дополнительных компонентов* броузера. Технологии Flash становятся все более мощными, и каждый год появляются все новые и новые ее разновидности.

Комбинируя HTML с серверными и клиентскими веб-технологиями, можно строить очень мощные веб-приложения.

Чего не хватает?

Итак, в нашем распоряжении есть практически все, зачем же нам нужны новые технологии? Чего еще не хватает?

Как говорилось в начале главы, любая технология существует для того, чтобы удовлетворять потребности рынка. А какая-то часть рынка всегда хочет предоставлять в распоряжение веб-клиентов более широкие функциональные возможности, без использования Flash, Java-апплетов или других технологий, которые либо выглядят слишком ярко и броско, либо слишком тяжеловесны для решения простых задач. В подобных случаях разработчики обычно создают веб-сайты и веб-приложения на основе комбинации HTML, JavaScript и PHP (или какой-нибудь другой серверной технологии). В данной ситуации типичный за-

прос выглядит так, как показано на рис. 1.3, где изображен запрос HTTP и ответ, состоящий из HTML и внедренного в него сценария на языке JavaScript, созданный программно с помощью PHP.

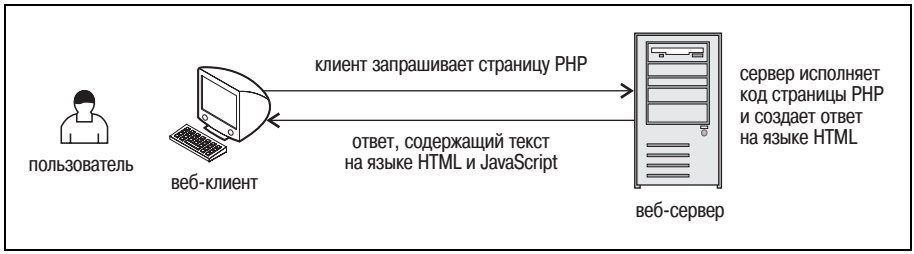

***Рис. 1.3.** HTTP, HTML, PHP и JavaScript в действии*

Подобное решение таит в себе неприятности, состоящие в том, что, когда клиенту необходимо получить новую порцию данных, он вынужден отправлять серверу новый запрос HTTP и повторно загружать страницу целиком, приостанавливая на время деятельность пользователя. **Повторная загрузка страницы** – это неизбежное зло в данной ситуации, а помочь нам победить это зло призван AJAX.

Что такое AJAX

Название AJAX – это акроним, раскрывающийся как **Asynchronous JavaScript and XML** и означающий **асинхронный JavaScript и XML**. Если это название, на ваш взгляд, мало о чем говорит, мы согласимся с вами. Проще говоря, можно считать, что AJAX – это «JavaScript с расширенными правами», поскольку по своей сути эта технология представляет собой сценарии на языке JavaScript, которые по мере необходимости в фоновом режиме выполняют запросы к серверу и получают дополнительные данные, обновляя отдельные части страницы и тем самым исключая необходимость повторной ее загрузки целиком. На рис. 1.4 наглядно показано, что происходит, когда посетитель запрашивает веб-страницу, созданную с применением технологии AJAX.

С точки зрения перспективы AJAX обладает лучшей сбалансированностью между функциональностью, реализуемой на стороне клиента, и функциональностью, реализуемой на стороне сервера, при выполнении действий, затребованных пользователем. До этого места функциональность клиента и функциональность сервера рассматривались как отдельные части, которые работают независимо друг от друга в ответ на действия, предпринимаемые пользователем. AJAX предлагает новое решение – распределить нагрузку между клиентом и сервером, разрешив им общаться между собой, *пока пользователь работает* со страницей.

Чтобы пояснить вышесказанное на простом примере, рассмотрим вебформу, куда пользователь должен внести некоторые сведения (напри-

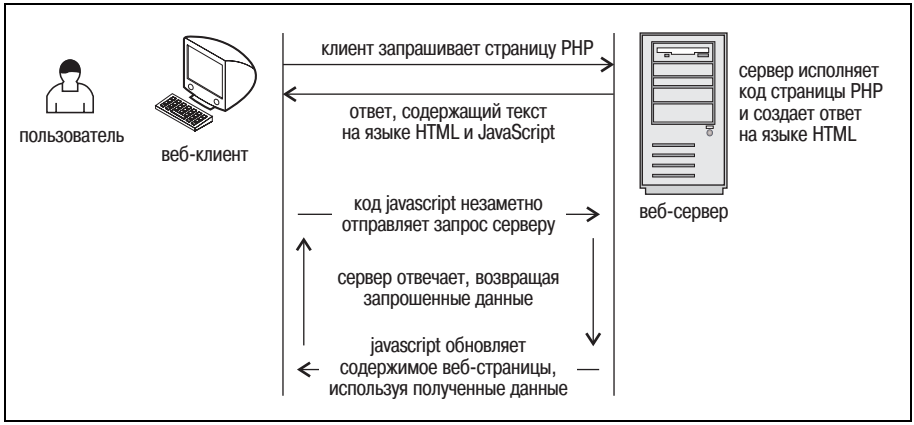

Рис. 1.4. *Типичный вызов AJAX*

мер, имя, адрес электронной почты, пароль, номер кредитной карточки и прочее). Прежде чем эта форма поступит в распоряжение вашего приложения, она должна быть проверена. В случае отказа от применения AJAX в нашем распоряжении имеются два способа проверки. Первый из них заключается в том, чтобы позволить ему или ей заполнить форму и *отправить* ее, после чего приложение проверит правильность заполнения на стороне сервера. В этом случае пользователь будет *терять время*, ожидая загрузки новой страницы. Другой вариант – выполнить проверку на стороне клиента, но это не всегда возможно, если для этого потребуется передать клиенту слишком большой объем данных (только представьте себе, что необходимо проверить правильность ввода названия города в соответствии с введенным названием страны).

Если же AJAX применяется, то веб-приложение может проверять данные, отправляя запросы серверу в фоновом режиме, по мере того как пользователь вводит их. Например, после того как пользователь выберет название страны, веб-броузер может запросить у сервера список городов этой страны, не отвлекая пользователя от его занятия. Пример проверки правильности заполнения формы с помощью технологии AJAX вы найдете в главе 4.

Рассказывать, где с успехом применяется AJAX, можно бесконечно. Более точное представление о возможностях, открываемых технологией AJAX, можно получить, взглянув на этот список реальных и достаточно популярных примеров:

- **Подсказки Google (Google Suggest)** помогут вам при работе с поисковой системой **Google**. Они представляют собой достаточно эффектное зрелище (просто зайдите по адресу *http://www.google.com/webhp?complete=1*). Аналогичная функциональность предлагается и **Yahoo! Instant Search**, которая доступна по адресу *http://instant.search.yahoo.com/*. (Как добиться этого, мы расскажем в главе 6.)

- **GMail (*http://www.gmail.com*).** Почтовая служба GMail пользуется заслуженной популярностью и не нуждается в представлении. Другие почтовые службы с веб-интерфейсом, такие как **Yahoo! Mail** и **Hotmail**, также наделены функциональностью технологии AJAX.
- **Google Maps (*http://maps.google.com*), Yahoo Maps (*http://maps.yahoo.com*)** и **Windows Live Local (*http://local.live.com*).**
- Прочие службы, такие как *http://www.writely.com* и *http://www.basecamphq.com*.

В этой книге вы встретите и другие примеры.

Примечание

AJAX, как и любая другая технология, может применяться *неправильно* или не *по назначению*. Если на вашем сайте применяется AJAX, то это еще не значит, что он станет лучше. Выигрыш, получаемый от технологии, зависит от того, насколько правильно и к месту вы ее используете.

Таким образом, технология AJAX служит для создания более гибких и интерактивных веб-приложений. Она позволяет выполнять асинхронные обращения к серверу, не прерывая работы пользователя и незаметно для него. AJAX – это инструмент, который может применяться разработчиками для создания веб-приложений, более интеллектуально взаимодействующих с человеком.

Технологии, из которых состоитAJAX, уже реализованы во всех современных веб-броузерах, таких как Mozilla Firefox, Internet Explorer или Opera. Таким образом, клиент не требует установки каких-либо дополнительных модулей, чтобы иметь возможность взаимодействия с веб-сайтами, построенными на основе AJAX. В состав AJAX входят следующие компоненты:

- JavaScript – основной ингредиент AJAX, позволяющий реализовать функциональность на стороне клиента. В ваших функциях JavaScript для манипулирования отдельными частями страницы HTML часто задействуется **объектная модель документа (Document Object Model – DOM)**.
- Объект **XMLHttpRequest** позволяет из JavaScript организовать асинхронный доступ к серверу, благодаря чему пользователь имеет возможность продолжать работу со страницей, в то время как она выполняет некоторые действия. Под доступом к серверу подразумеваются простые запросы HTTP на получение файлов или сценариев, размещенных на сервере. Запросы HTTP просты в исполнении и не вызывают каких-либо трудностей в случае применения брандмауэров.
- Серверные технологии, которые необходимы для обслуживания запросов, поступающих от JavaScript, со стороны клиента. В этой книге для выполнения действий на стороне сервера мы будем обращаться к PHP.

Для организации взаимодействия клиент-сервер необходимо иметь возможность передавать данные и понимать, что за данные были переданы. Передача данных – это самое простое. Сценарий на стороне клиента, обладающий доступом к серверу (посредством объекта `XMLHttpRequest`), может передавать серверу пары имя-значение с помощью методов **GET** или **POST**. Эти данные легко могут быть прочитаны с помощью любого сценария на стороне сервера.

Сценарий на стороне сервера просто отправляет свой ответ по протоколу HTTP, но, в отличие от обычного веб-сервера, ответ должен иметь такой формат, который легко может быть разобран кодом JavaScript на стороне клиента. Мы рекомендуем формат XML, который имеет свои преимущества, заключающиеся в том, что, во-первых, он получил широкое распространение и, во-вторых, существует большое количество библиотек, облегчающих работу с XML документами. Но при желании можно выбрать любой другой формат (данные могут передаваться даже в виде простого текста). Одна из известных альтернатив XML – **JavaScript Object Notation (JSON** – представление объектов в JavaScript).

Мы предполагаем, что вы уже знакомы со всеми составными частями AJAX, кроме, разве что, объекта `XMLHttpRequest`, который известен не так широко. Более подробно мы рассмотрим, как эти части работают и взаимодействуют между собой, в главах 2 и 3. А пока, до конца этой главы, мы сфокусируем свое внимание на рассмотрении общих принципов, а также, к большой радости самых нетерпеливых читателей, напишем программу с поддержкой AJAX.

Примечание

В AJAX нет ни одного нового или революционного компонента, как можно было бы подумать из-за шумихи, поднятой вокруг этой технологии, – все компоненты AJAX существуют, по крайней мере, с 1998 года. Название AJAX родилось в 2005 году в статье Джесси Джеймса Гаррета (Jesse James Garret, *http://www.adaptivepath.com/publications/essays/archives/000385.php*) и получило известность после того, как AJAX стал применяться компанией Google во многих собственных приложениях.

Зато с AJAX связано то новое обстоятельство, что впервые рынок аккумулировал достаточно энергии, чтобы поддержать стандартизацию и сфокусировать эту энергию на дальнейшем развитии. Как следствие, было разработано множество библиотек AJAX и появилось много веб-сайтов с поддержкой AJAX. Благодаря проекту Atlas, Microsoft также способствует развитию AJAX.

Применение технологии AJAX для создания новых веб-приложений дает нам следующие преимущества:

- Она позволяет создавать более динамичные и более качественные веб-сайты и веб-приложения.
- Высокая популярность способствует появлению шаблонных решений, которые помогут разработчикам не изобретать велосипед при решении наиболее распространенных задач.

- Она задействует уже существующие технологии.
- Позволяет разработчикам применять наработанные навыки.
- Функциональные возможности AJAX прекрасно интегрируются функциональностью, предоставляемой веб-броузерами (например, навигацией по странице, приведением размеров страницы к определенным значениям и т. д.).

Наиболее общие случаи применения AJAX:

- Проверка правильности заполнения формы с привлечением возможностей сервера, что очень удобно, когда невозможно заранее передать клиенту все данные, которые могут потребоваться в процессе проверки. Пример, в котором показано, как проверяется правильность заполнения формы, вы найдете в главе 4.
- Разработка простых чатов, которые не требуют наличия внешних библиотек, таких как виртуальная Java-машина или Flash. Подобную программу мы создадим в главе 5.
- Добавление функциональности, аналогичной подсказкам Google, пример мы разберем в главе 6.
- Более эффективное использование других технологий. В главе 7 мы реализуем приложение, которое динамически создает диаграммы с использованием **SVG (Scalable Vector Graphics** – масштабируемая векторная графика), а в главе 10 с помощью внешней библиотеки AJAX реализуем простейший список, поддерживающий возможность перетаскивания объектов мышью.
- Создание динамических **таблиц данных**, которые на лету обновляют базы данных на сервере. Подобное приложение мы создадим в главе 8.
- Разработка приложений, которые требуют обновления информации в режиме реального времени, считывая ее из различных внешних источников. В главе 9 мы создадим простейший **RSS**-агрегатор.

С AJAX связаны следующие потенциальные трудности:

- Поскольку адрес страницы в процессе ее работы не изменяется, добавить в закладки ссылку на страницу AJAX будет не так-то просто. В случае приложений AJAX установка закладки имеет иное значение, зависящее от конкретного приложения, т. е. обычно требуется сохранить текущее состояние (представьте себе, что работаете с обычной программой, ссылку на которую нельзя поместить в закладки).
- Поисковые системы могут оказаться не в состоянии проиндексировать все части вашего сайта, созданного на основе AJAX.
- Нажатие на кнопку «Назад» в броузерах не приводит к тому же результату, как в классических веб-приложениях, поскольку все действия пользователь выполняет в одной и той же странице.
- На стороне клиента JavaScript может быть отключен, что сделает приложения AJAX нефункциональными, поэтому неплохо бы пре-

дусмотреть на своем сайте альтернативные варианты страниц, чтобы не потерять потенциальных клиентов.

Наконец, прежде чем вы перейдете к написанию вашей первой программы с поддержкой AJAX, мы приведем ряд ссылок, которые помогут вам в вашем путешествии по увлекательному миру AJAX:

- *http://ajaxblog.com* – блог, посвященный AJAX.
- *http://www.fiftyfourleven.com/resources/programming/xmlhttprequest* – обширная коллекция статей об AJAX.
- *http://www.ajaxian.com* – сайт (с поддержкой AJAX) Бена Галбрайта (Ben Galbraith) и Диона Алмейра (Dion Almaer), авторов «Pragmatic AJAX».
- *http://www.ajaxmatters.com* – информационный сайт об AJAX, содержащий массу полезных ссылок.
- *http://ajaxpatterns.org* – рассказывает о шаблонах проектирования AJAX.
- *http://www.ajaxinfo.com* – ресурс содержит статьи об AJAX и полезные ссылки.
- *http://dev.fiaminga.com* – содержит массу ссылок на различные ресурсы и руководства, имеющие отношение к AJAX.
- *http://ajaxguru.blogspot.com* – популярный блог, посвященный AJAX.
- *http://www.sitepoint.com/article/remote-scripting-ajax* – замечательная статья Камерона Адамса (Cameron Adams) «Usable Interactivity with Remote Scripting».
- *http://developer.mozilla.org/en/docs/AJAX* – страница проекта Mozilla, посвященная AJAX.
- *http://en.wikipedia.org/wiki/AJAX* – страница об AJAX на Wikipedia.

По нашему мнению, этот список далеко не полный. Если вам потребуются сведения о других ресурсах, посвященных AJAX, в вашем распоряжении всегда имеется Google. В следующих главах мы дадим еще немало ссылок, но они будут касаться определенных технологий, о которых мы будем рассказывать.

Создание простого приложения на основе AJAX и PHP

А теперь приступим к написанию кода! На следующих страницах мы соберем простое приложение AJAX.

Примечание

Это упражнение для самых нетерпеливых читателей, которые стремятся приступить к программированию как можно скорее, однако вы уже должны быть знакомы с JavaScript, PHP и XML. Если это не так или в какой-то момент вы почувствуете, что это упражнение слишком сложно для вас, переходите к главе 2.

> В главах 2 и 3 мы поближе познакомимся с технологией AJAX и его методиками, и вам все станет понятно.

Здесь вы создадите простое веб-приложение под названием **quickstart**, применяя технологию AJAX. Оно запрашивает у пользователя имя и по мере ввода сервер возвращает клиенту полученные от него данные. На рис. 1.5 показана начальная страница `index.html`, загруженная пользователем. (Обратите внимание, страница `index.html` загружается по умолчанию, при обращении к веб-каталогу `quickstart`, даже если имя файла не указано явно.)

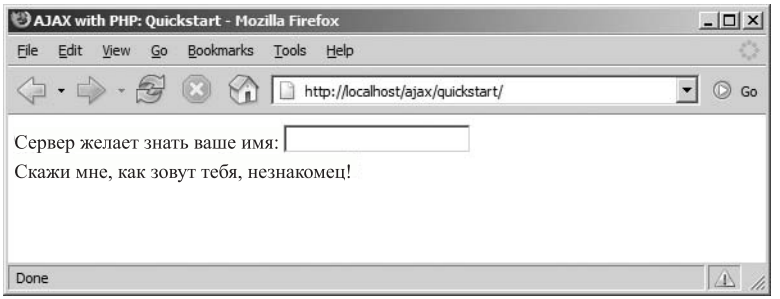

Рис. 1.5. *Начальная страница вашего приложения Quickstart*

Пока пользователь вводит свое имя, через регулярные интервалы времени производятся обращения к серверу, чтобы он мог опознать текущее имя. Сервер вызывается автоматически, примерно один раз в секунду, и поэтому на форме отсутствует кнопка «Передать» (Submit), с помощью которой производится передача данных серверу в классических веб-приложениях. (Такой метод плохо подходит для реализации механизмов авторизации пользователей, но он прекрасно демонстрирует некоторые возможности AJAX.)

В зависимости от того, какое имя было введено, сообщения, получаемые от сервера, могут отличаться (рис. 1.6).

Проверить работу этого приложения вы сможете, зайдя стандартным броузером по адресу *http://ajaxphp.packtpub.com/ajax/quickstart*.

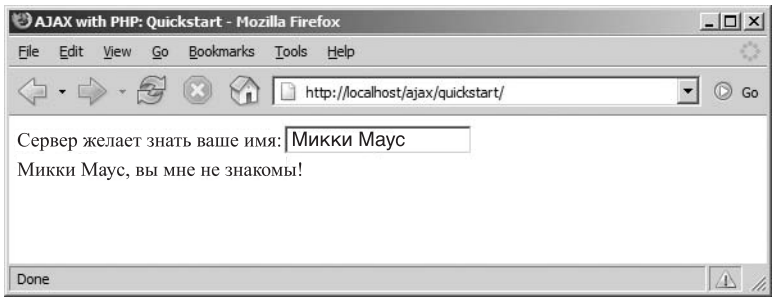

Рис. 1.6. *Пользователь получает ответ от сервера*

На первый взгляд в этом приложении нет ничего экстраординарного. Мы нарочно сохранили простоту первого примера, чтобы вам было проще разобраться в нем. Примечательно в этом приложении то, что отображаемое сообщение автоматически поступает от сервера, не прерывая действий пользователя. (Сообщение появляется сразу же, как только пользователь начинает вводить свое имя.) **Для того чтобы отобразить сообщение, приложение не производит повторную загрузку всей страницы, даже несмотря на то, что за получением текста сообщения необходимо обращаться к серверу.** Без помощи AJAX достигнуть такого эффекта очень сложно.

Приложение состоит из трех файлов:

1. `index.html` – файл HTML, который изначально запрашивает пользователь.
2. `quickstart.js` – файл, содержащий код JavaScript, который загружается клиентом вместе с `index.html`. Этот файл отвечает за выполнение асинхронных обращений к серверу, когда становится необходимой его функциональность.
3. `quickstart.php` – сценарий PHP, постоянно находящийся на сервере. Этот сценарий вызывается со стороны клиента из кода JavaScript, находящегося в файле `quickstart.js`.

На рис. 1.7 показана последовательность действий, выполняемых приложением.

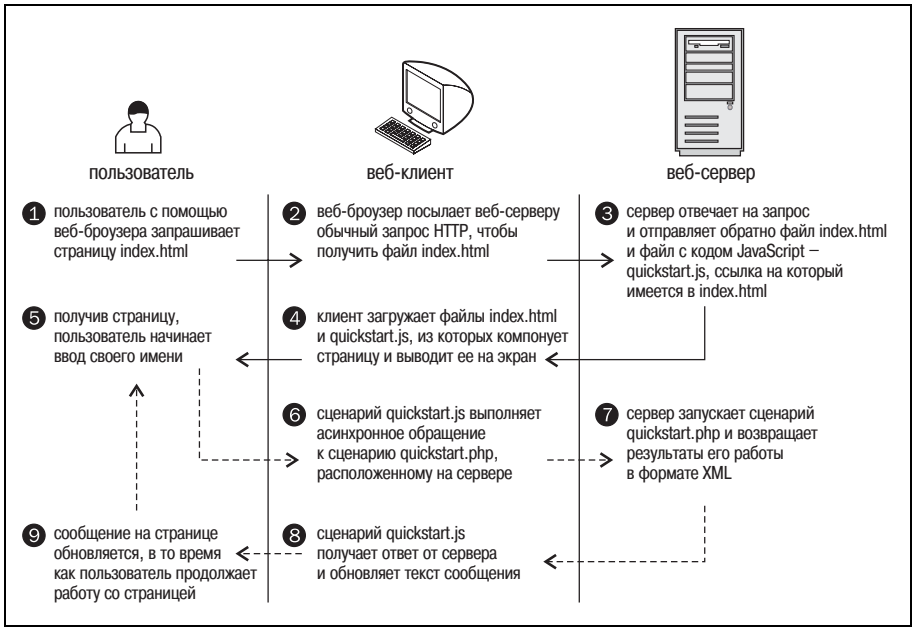

***Рис. 1.7.** Диаграмма, описывающая процессы, протекающие внутри приложения Quickstart*

Шаги с 1 по 5 – это типичный запрос HTTP. После того как запрос будет послан, пользователю придется подождать, пока загрузится страница. В случае типичных веб-приложений (не использующих технологию AJAX) такая загрузка страницы происходит всякий раз, когда клиент пытается получить новые данные от сервера.

Шаги с 5 по 9 демонстрируют запрос AJAX, а если точнее, то целую серию асинхронных запросов HTTP. Доступ к серверу производится в фоновом режиме, с помощью объекта `XMLHttpRequest`. В это время пользователь может продолжать нормальную работу со страницей, как если бы это было обычное приложение. Чтобы получить новые данные от сервера и обновить текст сообщения на странице, не надо загружать страницу повторно.

Теперь пора реализовать этот код на вашей машине. Прежде чем двинуться дальше, убедитесь, что ваше рабочее окружение настроено так, как описано в приложении A, где рассказано, как установить и настроить PHP и Apache и как настроить базу данных, фигурирующую в примерах из этой книги. (Для реализации примера quickstart база данных не нужна.)

Примечание

Все примеры в этой книге предполагают, что ваше рабочее окружение настроено так, как описано в приложении A. Если это не так, то вам скорее всего придется вносить некоторые изменения, например, изменить названия каталогов и тому подобное.

Время действовать – Quickstart AJAX

1. В приложении A вы найдете инструкции по установке и настройке веб-сервера и созданию каталога, доступного через Интернет, с именем `ajax`, в котором будут размещаться все исходные тексты примеров из этой книги. В каталоге `ajax` создайте каталог `quickstart`.
2. В каталоге `quckstart` создайте файл с именем `index.html` и добавьте в него следующий код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>AJAX и PHP: Quickstart</title>
        <script type="text/javascript" src="quickstart.js"></script>
    </head>
    <body onload='process()'>
        Сервер желает узнать ваше имя:
        <input type="text" id="myName" />
        <div id="divMessage" />
    </body>
</html>
```

3. **Создайте новый файл с именем** `quickstart.js` **и добавьте в него следующий код:**

```
// запомнить ссылку на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();

// создать объект XMLHttpRequest
function createXmlHttpRequestObject()
{
    // для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // если сценарий запущен под управлением Internet Explorer
    if(window.ActiveXObject)
    {
        try
        {
            xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        catch (e)
        {
            xmlHttp = false;
        }
    }
    // если сценарий запущен под управлением Mozilla или другого броузера
    else
    {
        try
        {
            xmlHttp = new XMLHttpRequest();
        }
        catch (e)
        {
            xmlHttp = false;
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// выполнить асинхронный запрос HTTP с помощью объекта XMLHttpRequest
function process()
{
    // работа возможна только если объект xmlHttp не занят
    if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
    {
        // получить имя, введенное пользователем в форму
        name = encodeURIComponent(document.getElementById("myName").value);
        // обратиться к сценарию quickstart.php на сервере
        xmlHttp.open("GET", "quickstart.php?name=" + name, true);
```

```
            // определить метод, который будет обрабатывать ответы сервера
            xmlHttp.onreadystatechange = handleServerResponse;
            // послать асинхронный запрос серверу
            xmlHttp.send(null);
        }
        else
            // если соединение занято, повторить попытку через одну секунду
            setTimeout('process()', 1000);
    }

    // вызывается автоматически по прибытии сообщения от сервера
    function handleServerResponse()
    {
        // продолжать можно только если транзакция с сервером завершена
        if (xmlHttp.readyState == 4)
        {
            // значение 200 говорит о том, что транзакция прошла успешно
            if (xmlHttp.status == 200)
            {
                // извлечь XML, полученный от сервера
                xmlResponse = xmlHttp.responseXML;
                // получить корневой элемент в структуре XML
                xmlDocumentElement = xmlResponse.documentElement;
                // извлечь текстовое сообщение, которое находится в первом
                // дочернем элементе корневого узла
                helloMessage = xmlDocumentElement.firstChild.data;
                // обновить текст сообщения на экране
                document.getElementById("divMessage").innerHTML =
                                                '<i>' + helloMessage + '</i>';
                // повторить последовательность действий
                setTimeout('process()', 1000);
            }
            // код статуса HTTP, отличный от 200, говорит о наличии ошибки
            else
            {
                alert("При обращении к серверу возникли проблемы: " +
                      xmlHttp.statusText);
            }
        }
    }
```

4. **Создайте файл с именем** `quickstart.php` **и добавьте в него следующий код:**

```
<?php
    // результаты будем отправлять в формате XML
    header('Content-Type: text/xml');
    // сгенерировать заголовок XML
    echo '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
    // создать элемент <response>
    echo '<response>';
    // получить имя пользователя
```

```
    $name = $_GET['name'];
    // сгенерировать текст сообщения в зависимости
    // от имени пользователя принятого от клиента
    $userNames = array('CRISTIAN', 'BOGDAN', 'FILIP', 'MIHAI', 'YODA');
    if (in_array(strtoupper($name), $userNames))
        echo 'Здравствуйте, мастер ' . htmlentities($name) . '!';
    else if (trim($name) == '')
        echo 'Скажи мне, как зовут тебя, незнакомец!';
    else
        echo htmlentities($name) . ', вы мне не знакомы!';
    // закрыть элемент <response>
    echo '</response>';
?>
```

5. Теперь вы можете выполнить вашу новую программу, зайдя своим любимым броузером по адресу *http://localhost/ajax/quickstart*. Загрузите страницу, и у вас должно получиться так, как изображено на рис. 1.5 и 1.6.

Примечание

Если при запуске приложения вы столкнетесь с какими-либо трудностями, проверьте еще раз правильность установки и настройки рабочего окружения в соответствии с описанием в приложении А. Как показывает опыт, большинство неприятностей связаны с банальными опечатками. В главах 2 и 3 вы научитесь обрабатывать ошибки в коде JavaScript и PHP.

Что происходит внутри?

Мы подошли к самому интересному – настало время разобраться с тем, что делает этот код. (Помните, что технические детали мы обсудим подробнее в следующих двух главах.)

Начнем с файла, который первым встречается пользователю, – `index.html`. Этот файл содержит ссылку на таинственный файл JavaScript с именем `quickstart.js` и создает очень простенький веб-интерфейс для клиента. В следующем отрывке кода, взятом из `index.html`, особенно примечательные элементы выделены жирным шрифтом:

```
<body onload='process()'>
    Сервер желает узнать ваше имя:
    <input type="text" id="myName" />
    <div id="divMessage" />
</body>
```

После загрузки страницы вызывается функция `process()` из файла `quickstart.js`. Она заполняет элемент `<div>` сообщением, полученным от сервера.

Прежде чем рассматривать, что происходит внутри функции `process()`, мы узнаем, что происходит на стороне сервера. На веб-сервере находится сценарий `quickstart.php`, который собирает сообщение в формате

XML и отправляет его клиенту. Это сообщение содержит элемент <response>, в который упакован текст сообщения:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
    ... текст сообщения, отправляемого клиенту ...
</response>
```

Если в качестве имени пользователя сервер получил пустую строку, то в качестве сообщения будет отправлен текст: «Скажи мне, как тебя зовут, незнакомец!». Если сервер получит имя, одно из пяти: Cristian, Bogdan, Filip, Mihai или Yoda, сервер ответит: «Здравствуйте, мастер <имя пользователя>!». Если будет получено какое-либо другое имя, сервер ответит сообщением: «<имя пользователя>, вы мне не знакомы!». Так, если Микки Маус введет свое имя, сервер отправит следующую структуру XML:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
    Микки Маус, вы мне не знакомы!
</response>
```

Сценарий quickstart.php начинает с генерации заголовка документа в формате XML и открытия элемента <response>:

```
<?php
    // результаты будем отправлять в формате XML
    header('Content-Type: text/xml');
    // сгенерировать заголовок XML
    echo '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
    // создать элемент <response>
    echo '<response>';
```

Выделенная строка сообщает о том, что выходные данные представляют собой документ XML. Это очень важно, поскольку клиент ожидает получить ответ от сервера в формате XML (прикладной программный интерфейс на стороне клиента сгенерирует ошибку, если в заголовке Content-Type не будет содержаться значение text/xml). После настройки заголовка код собирает ответ путем выполнения слияния строк. Фактический текст сообщения, который будет отправлен клиенту, заключается в элемент <response>, корневой элемент, а само сообщение генерируется на основе имени, полученного от клиента через параметр GET:

```
    // получить имя пользователя
    $name = $_GET['name'];
    // сгенерировать текст сообщения в зависимости от имени пользователя,
    // принятого от клиента
    $userNames = array('CRISTIAN', 'BOGDAN', 'FILIP', 'MIHAI', 'YODA');
    if (in_array(strtoupper($name), $userNames))
        echo 'Здравствуйте, мастер ' . htmlentities($name) . '!';
    else if (trim($name) == '')
        echo 'Скажи мне, как зовут тебя, незнакомец!';
    else
```

```
        echo htmlentities($name) . ', вы мне не знакомы!';
    // закрыть элемент <response>
    echo '</response>';
?>
```

Текст, введенный пользователем (предполагается, что это его имя), клиент отправляет серверу через параметр `GET`. Когда сервер передает имя обратно клиенту, мы вызываем функцию PHP `htmlentities()`, которая замещает служебные символы (такие как & или >) их HTML-аналогами, тем самым гарантируя безопасное отображение их веб-броузером и устраняя потенциальные неприятности.

Примечание

Форматирование текста, предназначенного для клиента, на стороне сервера (вместо того, чтобы делать это на стороне клиента) – это плохая практика. В идеале сервер должен возвращать клиентам данные в обобщенном формате, а уже клиент должен побеспокоиться о форматировании и безопасности. Такое положение вещей обретает еще больший смысл, если вы только представите, что когда-либо в будущем вам придется вставлять текст ответа, например, в базу данных, но база данных требует добавления иных форматирующих последовательностей (в этом случае точно так же о форматировании должен беспокоиться сценарий на стороне базы данных, но никак не сервер). В случае с приложением quickstart форматирование кода HTML в сценарии PHP вполне допустимо, поскольку это обеспечивает простоту нашего примера и облегчает его изучение.

Если вы испытываете желание протестировать сценарий `quickstart.php` и увидеть, что он возвращает, установите в броузере в адресном поле значение *http://localhost/ajax/quickstart/quickstart.php?name=Yoda*. Преимущество передачи параметров от клиента методом `GET` состоит в том, что такие запросы легко эмулируются с помощью броузера, поскольку в этом случае достаточно добавить параметры в виде пар имя/значение в конец строки запроса. В результате вы должны получить примерно то же, что показано на рис. 1.8.

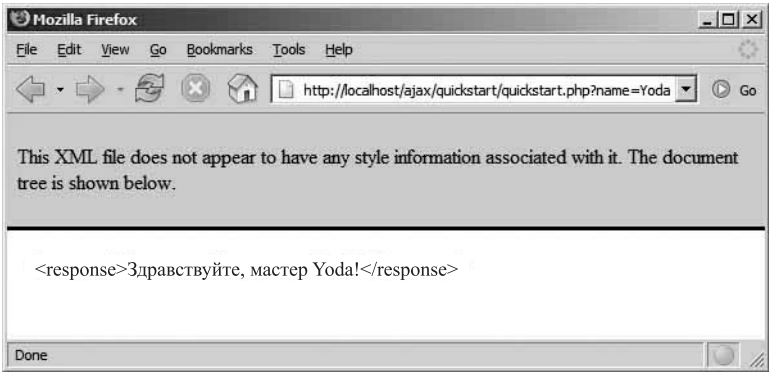

***Рис. 1.8.** Данные в формате XML, сгенерированные сценарием quickstart.php*

Это сообщение в формате XML клиент получает с помощью функции handleServerResponse(), которая находится в файле quickstart.js. Точнее говоря, текст сообщения «Привет, мастер Yoda!» извлекается следующей частью кода:

```
// извлечь XML, полученный от сервера
xmlResponse = xmlHttp.responseXML;
// получить корневой элемент в структуре XML
xmlDocumentElement = xmlResponse.documentElement;
// извлечь текстовое сообщение, которое находится в первом
// дочернем элементе корневого узла
helloMessage = xmlDocumentElement.firstChild.data;
```

Здесь переменная xmlHttp представляет собой ссылку на объект XMLHttpRequist, который в сценарии quickstart.js служит для передачи запросов серверу. Свойство responseXML этого объекта извлекает полученный XML-документ, имеющий иерархическую структуру, а корневой элемент документа XML называется *элементом документа*. В нашем случае корневым является элемент <response>, который имеет единственного потомка – текст интересующего нас сообщения. После того как текст будет извлечен, он отображается на странице, благодаря использованию DOM для доступа к элементу divMessage, находящемуся в файле index.html.

```
// обновить текст сообщения на экране
document.getElementById("divMessage").innerHTML =
                                          '<i>' + helloMessage + '</i>';
```

где document – это объект по умолчанию в JavaScript, который позволяет манипулировать элементами в HTML коде вашей страницы.

Остальной код в quickstart.js занимается выполнением запроса к серверу, чтобы в ответ получить XML-сообщение. Функция createXmlHttpRequestObject() создает и возвращает экземпляр объекта XMLHttpRequest. Она несколько больше, чем могла бы быть, потому что нам необходимо сделать ее совместимой с различными типами броузеров (более подробно мы обсудим этот вопрос в главе 2), а пока основной интерес для нас представляет то, что она делает. Экземпляр XMLHttpRequest, ссылка на который хранится в переменной xmlHttp, используется функцией process() для передачи серверу асинхронных запросов:

```
// выполнить асинхронный запрос HTTP с помощью объекта XMLHttpRequest
function process()
{
    // работа возможна, только если объект xmlHttp не занят
    if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
    {
        // получить имя, введенное пользователем в форму
        name = encodeURIComponent(document.getElementById("myName").value);
        // обратиться к сценарию quickstart.php на сервере
        xmlHttp.open("GET", "quickstart.php?name=" + name, true);
```

```
            // определить метод, который будет обрабатывать ответы сервера
            xmlHttp.onreadystatechange = handleServerResponse;
            // послать асинхронный запрос серверу
            xmlHttp.send(null);
        }
        else
            // если соединение занято, повторить попытку через одну секунду
            setTimeout('process()', 1000);
    }
```

То, что вы сейчас видите перед собой, фактически является сердцем AJAX; это код, который отправляет асинхронные вызовы серверу.

Почему это так важно, чтобы обращения к серверу были асинхронными? Дело в том, что асинхронные запросы не приостанавливают работу пользователя (и, соответственно, ход его мыслей). Асинхронная обработка реализуется архитектурой, управляемой событиями, прекрасный пример – графический интерфейс с пользователем: если бы не было событий, вам наверняка пришлось бы постоянно проверять – не была ли нажата кнопка или не были ли изменены размеры окна. Благодаря наличию событий кнопка автоматически извещает приложение, когда по ней щелкают мышью, а вы можете предусмотреть выполнение необходимых действий в функции, обрабатывающей это событие. В AJAX такое положение вещей распространяется и на обращения к серверу – вы автоматически будете извещены, когда поступит ответ от сервера.

Если вам интересно узнать, как работают приложения, использующие синхронные запросы, измените третий аргумент функции `xmlHttp.open`, установив его в значение `false`, и после этого самостоятельно вызовите `handleServerResponse`, как это показано ниже. Запустив измененный таким образом пример, вы обнаружите, что поле ввода «замораживается» на время соединения с сервером (в данном случае длительность периода «замораживания» в значительной степени зависит от скорости соединения, поэтому задержка будет почти незаметной, если сервер запущен на той же самой машине).

```
// функция обращается к серверу с помощью объекта XMLHttpRequest
function process()
{
    // получить имя, введенное пользователем в форму
    name = encodeURIComponent(document.getElementById("myName").value);
    // обратиться к сценарию quickstart.php на сервере
    xmlHttp.open("GET", "quickstart.php?name=" + name, false);
    // послать синхронный запрос серверу
    // (приостанавливает работу приложения до получения ответа)
    xmlHttp.send(null);
    // прочитать ответ
    handleServerResponse();
}
```

Функция process() должна с помощью объекта XMLHttpRequest инициировать новый запрос. Однако это возможно только в том случае, если объект XMLHttpRequest не занят обработкой предыдущего запроса. Это может произойти, когда сервер занят обработкой нашего запроса более чем одну секунду. Это вполне возможно при достаточно медленном соединении с Интернетом. Таким образом, функция process() в первую очередь проверяет, может ли она инициировать новый запрос:

```
// выполнить асинхронный запрос HTTP с помощью объекта XMLHttpRequest
function process()
{
    // работа возможна, только если объект xmlHttp не занят
    if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
    {
```

Итак, если соединение занято обработкой предыдущего запроса, мы вызываем функцию setTimeout, чтобы повторить попытку через одну секунду (второй аргумент функции определяет время в миллисекундах, которое должно пройти, прежде чем будет вызвана функция, определяемая первым аргументом):

```
// если соединение занято, повторить попытку через одну секунду
setTimeout('process()', 1000);
```

Если линия свободна, можно послать новый запрос. Следующие строки подготавливают запрос к серверу, но не отправляют его:

```
// обратиться к сценарию quickstart.php на сервере
xmlHttp.open("GET", "quickstart.php?name=" + name, true);
```

Первый аргумент определяет метод передачи имени пользователя серверу. Здесь можно выбирать между методом GET и POST (подробнее об этих методах мы поговорим в главе 3). Второй аргумент – имя страницы на сервере, к которой мы хотим обратиться. При использовании метода GET передаваемые параметры помещаются в строку запроса в виде пар имя/значение. Третий параметр должен иметь значение true, если вы хотите, чтобы запрос был произведен асинхронно. Когда выполняется асинхронный вызов, не надо ждать, пока поступит ответ. Вместо этого вы определяете функцию, которая будет вызвана автоматически при изменении состояния запроса:

```
// определить метод, который будет обрабатывать ответы сервера
xmlHttp.onreadystatechange = handleServerResponse;
```

Установив это свойство объекта, можете быть спокойны – функция handleServerResponse будет вызвана системой, как только что-либо произойдет с вашим запросом. По окончании необходимых подготовительных операций запрос инициируется обращением к методу send объекта XMLHttpRequest:

```
        // послать асинхронный запрос серверу
        xmlHttp.send(null);
    }
```

А теперь рассмотрим функцию handleServerResponse:

```
// вызывается автоматически по прибытии сообщения от сервера
function handleServerResponse()
{
    // продолжать можно только если транзакция с сервером завершена
    if (xmlHttp.readyState == 4)
    {
        // значение 200 говорит о том, что транзакция прошла успешно
        if (xmlHttp.status == 200)
        {
```

Функция handleServerResponse вызывается всякий раз, когда изменяется статус запроса. Но только значение 4 в свойстве xmlHttp.readyState говорит о том, что запрос сервера был выполнен и вы можете прочитать полученный ответ. Можно дополнительно проверить код завершения транзакции HTTP, который в случае успеха должен быть равен 200. Если все эти условия соблюдены, можно прочитать сообщение, полученное от сервера, и вывести его пользователю.

После того как ответ сервера будет прочитан и обработан, вызывается функция setTimeout, которая в свою очередь вызовет функцию process() через одну секунду, и весь процесс повторится (обратите внимание: совершенно необязательно, и AJAX не требует этого, чтобы на стороне клиента в цикле выполнялись повторяющиеся задачи).

```
// повторить последовательность действий
setTimeout('process()', 1000);
```

А теперь еще раз пройдемся по тем событиям, которые происходят после того, как пользователь загрузит страницу (вы можете обращаться к рис. 1.7, чтобы зрительно представить себе происходящее):

1. Пользователь загружает index.html (это соответствует шагам 1–4 на рис. 1.7).
2. Пользователь начинает (или продолжает) ввод символов своего имени (что соответствует шагу 5 на рис. 1.7).
3. Когда в quickstart.js запускается метод process(), он асинхронно обращается к сценарию quickstart.php, расположенному на стороне сервера. Текст, введенный пользователем, передается серверу в виде параметров в строке запроса (методом GET). Для обработки изменений состояния запроса предназначена функция handleServerResponse.
4. Сервер запускает сценарий quickstart.php. Он конструирует документ в формате XML, который включает в себя сообщение, передаваемое клиенту.
5. При каждом изменении состояния запроса на стороне клиента отрабатывает метод handleServerResponse. В последний раз он вызывается, когда ответ будет благополучно принят от сервера. Далее производится чтение документа XML (текст сообщения извлекается из него и отображается в странице).

6. Изображение перед пользователем постоянно обновляется с получением каждого нового сообщения от сервера, но он может продолжать ввод, не приостанавливаясь. Через одну секунду все повторится, начиная с пункта 2.

Подведение итогов

Эта глава стала кратким введением в мир AJAX. Чтобы продолжать изучение и узнать, как создаются приложения AJAX, важно понимать, где и зачем они применяются. Как и любая другая технология, AJAX не является универсальным решением, но она предлагает средства, способные разрешить некоторые из проблем.

Технология AJAX объединяет функциональные возможности клиента и сервера, чтобы улучшить потребительские качества вашего сайта. Объект `XMLHttpRequest` – это ключевой элемент, который дает возможность коду JavaScript, расположенному на стороне клиента, асинхронно вызвать страницы, расположенные на сервере. Мы преднамеренно сделали эту главу такой краткой. Вероятно, она вызвала у вас массу вопросов. И это хорошо! Вы теперь готовы к тому, чтобы окунуться в книгу, специально посвященную ответам на вопросы и демонстрации массы интересных функциональных возможностей!

2

Клиентские технологии на основе JavaScript

Говорят, что одна картинка заменяет тысячу слов. То же самое можно сказать о хорошо написанных примерах исходного кода. В этой и последующих главах вы получите и то и другое, попутно закладывая фундамент знаний для ваших будущих приложений AJAX.

Хотелось бы надеяться, что первая глава пробудила в вас интерес к технологии AJAX настолько, что в этой вы сможете получить удовольствие от массы теоретических изысканий. С другой стороны, если вы нашли первый пример слишком обнадеживающим, то, уверяем вас, на этот раз мы будем двигаться вперед еще медленнее. Мы будем углубляться в теорию небольшими шагами, изучая много коротких примеров. В этой главе мы рассмотрим технологии AJAX, применяемые на стороне клиента, в состав которых входят:

- JavaScript
- JavaScript DOM
- Каскадные таблицы стилей (Cascading Style Sheets – CSS)
- Объект `XMLHttpRequest`
- Расширяемый язык разметки (eXtensible Markup Language – XML)

Здесь вы узнаете, как заставить все эти компоненты работать как единое целое, и заложите крепкий фундамент в строительство ваших будущих приложений AJAX. Вы увидите, как можно организовать обработку ошибочных ситуаций и как повысить эффективность кода. Глава 3 завершает теоретическую часть книги, и в ней мы рассмотрим технологии, применяемые на стороне сервера; в нашем случае это PHP, MySQL и другие.

Тот, кто хочет стать хорошим AJAX-разработчиком, должен во всех подробностях узнать, как работают все составные части этой технологии по отдельности, и затем овладеть методиками, способными заста-

вить их работать как единое целое. Материал этой книги предполагает, что вы уже хотя бы частично знакомы с этими технологиями.

В зависимости от уровня вашей подготовленности найдите время (до, во время или после прочтения главы 2 или 3) и загляните в приложение B, которое вы найдете по адресу *http://ajaxphp.packpub.com*. В нем вы ознакомитесь с некоторыми инструментами, которые способны облегчить жизнь программиста. Не упускайте такую возможность, поскольку очень важно обладать хорошим набором инструментальных средств и уметь эффективно применять его на практике.

Все примеры из этой книги вы найдете по адресу *http://ajaxphp.packtpub.com/*.

JavaScript и объектная модель документа (DOM)

Как уже упоминалось в главе 1, JavaScript представляет собой сердце AJAX. Синтаксис JavaScript напоминает старый добрый C. JavaScript относится к *языкам интерпретирующего типа* (не компилирующего) и обладает некоторыми возможностями **объектно-ориентированного программирования (ООП)**. JavaScript не предназначен для создания больших и мощных приложений, но вполне пригоден для разработки простых сценариев, реализующих (или дополняющих) функциональность веб-приложений на стороне клиента (однако в последнее время наблюдается тенденция развития JavaScript до уровня языков программирования корпоративного класса, и еще неизвестно, как далеко это зайдет).

JavaScript в полном объеме поддерживается большинством веб-броузеров. Несмотря на то, что существует возможность запускать сценарии JavaScript отдельно, они обычно загружаются клиентскими броузерами вместе с кодом HTML, требующим расширенной функциональности. Фактически весь исполняемый код JavaScript должен быть целиком доставлен клиенту, и в этом заключаются его сила и слабость одновременно, что необходимо учитывать, обдумывая архитектуру своего веб-приложения. Очень хороший вводный материал по JavaScript вы найдете по следующим адресам:

- *http://www.echoecho.com/javascript.htm*
- *http://www.devlearn.com/javascript/jsvars.html*
- *http://www.w2schools.com/js/default.asp*

Одним из слагаемых успеха JavaScript, работающего на стороне клиента, является его способность манипулировать родительским документом HTML, а достигается это за счет использования интерфейса DOM (Document Object Model – объектная модель документа). Интерфейс DOM присутствует во многих технологиях и языках программирования, включая JavaScript, Java, PHP, C#, C++ и т. д. Из этой главы вы узнаете, как использовать DOM из JavaScript и PHP. Интерфейс

DOM предоставляет возможность манипулировать (создавать, изменять, анализировать, производить поиск и т. д.) XML-подобными документами, включая HTML.

На стороне клиента DOM и JavaScript применяются для:

- Манипулирования страницами HTML во время работы.
- Чтения и синтаксического анализа документов XML, принимаемых от сервера.
- Создания новых документов XML.

На стороне сервера можно применять DOM и PHP для:

- Создания документов XML, которые обычно предназначены для последующей передачи клиенту.
- Чтения документов XML, поступающих из различных источников.

Прекрасные введения в интерфейс DOM вы найдете по адресам *http://www.quirksmode.org/dom/intro.html* и *http://www.javascriptkit/com/javatutors/dom.shtml*. Поиграть в замечательную DOM-игру можно здесь: *http://www.topxml.com/learning/games/b/default.asp*. Всеобъемлющий справочник по JavaScript DOM вы найдете по адресу: *http://krook.org/jsdom/*. Справочник по JavaScript DOM от Mozilla – по адресу: *http://www.mozilla.org/docs/dom/reference/javascript.html*.

В первом примере из этой главы интерфейс DOM задействуется из кода JavaScript для манипулирования документом HTML. Чтобы добавить код JavaScript в файл HTML, в него, как один из возможных вариантов, в пределах тега `<body>` необходимо добавить тег `<script>`. Рассмотрим следующий пример файла HTML, который после загрузки исполняет некоторый простой код JavaScript. Обратите внимание на объект `document`, который в JavaScript является объектом по умолчанию, через этот объект производятся манипуляции со страницей HTML. В данном случае мы добавляем строки в страницу посредством метода `write`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: JavaScript и DOM</title>
        <script type="text/javascript">
            // объявления новых переменных
            var date = new Date();
            var hour = date.getHours();
            // демонстрация условного оператора if
            if (hour >= 22 || hour <= 5)
                document.write("Вам пора ложиться спать.");
            else
                document.write("Привет, МИР!");
        </script>
    </head>
<body>
```

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

Команда `document.write` генерирует строку, которая добавляется в элемент `<body>` при исполнении сценария. Вновь созданное содержимое становится частью кода HTML страницы, этот способ позволяет при необходимости добавлять теги HTML.

Мы советуем вам всегда создавать *корректный* (*well-formed*) и *действительный* (*valid*) код HTML. Если код соответствует формату HTML, то страницы будут с большей вероятностью корректно отображаться большинством современных и будущих версий веб-броузеров. По адресу *http://www.w3.org/QA/2002/04/Web-Quality* вы найдете полезную статью, описывающую стандарты, которым необходимо следовать. Описание элемента `DOCTYPE` приводится по адресу *http://www.alistapart.com/stories/doctype/*. Дебаты о стандартах, похоже, никогда не закончатся. Кто-то призывает строго следовать стандартам, других интересует лишь то, чтобы их страницы хорошо смотрелись в определенном наборе броузеров. Примеры в этой книге содержат действительный код HTML, за исключением нескольких случаев, когда мы отступили от правил только для того, чтобы упростить код и сделать его более понятным. В Интернете не так много сайтов, которые строго следуют стандартам.

Как правило, код JavaScript сохраняют в отдельном файле с расширением `.js`, а ссылку на него вставляют в файл `.html`. Это позволяет сделать код HTML простым и понятным и организовать хранение всего кода JavaScript в одном месте. Ссылка на файл с кодом JavaScript оформляется в виде тега `<script>`, добавляемого в элемент `<head>` следующим образом:

```
<html>
    <head>
        <script type="text/javascript" src="file.js"></script>
    </head>
</html>
```

Примечание

Даже если вы не добавляете какой-либо код между тегами `<script>` и `</script>`, старайтесь избегать краткой формы записи `<script type="text/javascript" src="file.js" />`. Это нарушает работу IE 6, который в этом случае отказывается загружать код JavaScript.

Рассмотрим короткий пример.

Время действовать – поиграем с JavaScript и DOM

1. В каталоге `ajax` создайте подкаталог `foundations`. В нем мы будем размещать все примеры из этой и следующей глав.
2. В каталоге `foundations` создайте подкаталог `jsdom`.

3. В каталоге `jsdom` создайте файл `jsdom.html` и добавьте в него код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: JavaScript и DOM</title>
        <script type="text/javascript" src="jsdom.js"></script>
    </head>
    <body>
        Я люблю тебя!
    </body>
</html>
```

4. В том же каталоге создайте файл `jsdom.js` и добавьте в него код:

```
// объявление новых переменных
var date = new Date();
var hour = date.getHours();
// демонстрация условного оператора if
if (hour >= 22 || hour <= 5)
    document.write("Спокойной ночи, МИР!");
else
    document.write("Привет, МИР!");
```

5. Откройте страницу *http://localhost/ajax/foundations/jsdom/jsdom.html* в вашем веб-броузере, и, если на часах еще не слишком много времени, вы увидите сообщение, как на рис. 2.1 (после 22:00 текст сообщения будет несколько иным, но не менее романтичным).

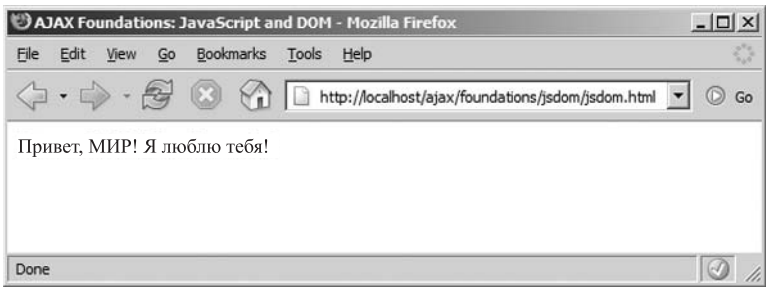

Рис. 2.1. *Пример работы с JavaScript и DOM*

Что происходит внутри?

Код приложения настолько прост, что едва ли нуждается в пространных комментариях. Вот основные моменты, которые нужно понять:

- Взаимодействие со сценарием на стороне сервера (например, со сценарием PHP) отсутствует, поэтому можно загрузить страницу в броузер прямо с диска, не обращаясь к веб-серверу. Если запустить файл прямо с диска, то веб-броузер откроет его автоматически по

локальному адресу, например такому: *file:///C:/Apache2/htdocs/ajax/foundations/jsdom/jsdom.html*.

- Загружая страницу HTML с кодом JavaScript с диска (*file:///*), а не с веб-сервера (*http://*), Internet Explorer может выдать предупреждение о том, что вы собираетесь исполнить код с высокими привилегиями (подробнее о безопасности говорится в главе 3).
- JavaScript не требует предварительного объявления переменных, поэтому, теоретически, можно убрать ключевые слова `var` из текста сценария. Но мы не рекомендуем это делать.
- Сценарий JavaScript запускается автоматически, сразу же *после* загрузки страницы HTML. Но можно сгруппировать код JavaScript в функции, которые запускаются только в случае явного обращения к ним.
- Код JavaScript запускается до начала синтаксического анализа кода HTML, поэтому все, что будет сгенерировано сценарием, будет размещаться перед всем остальным, находящимся на странице. Обратите внимание: текст «Привет, МИР!» был выведен перед текстом «Я люблю тебя!».

В этом сценарии есть одна неувязка – нельзя заранее определить место, где должен отображаться результат его работы. Строка, генерируемая сценарием, выводится первой, и лишь вслед за ней выводится содержимое тега `<body>`. Разумеется, такое положение вещей совершенно неприемлемо даже для простейших веб-приложений.

За исключением самых простых случаев, безусловного исполнения кода JavaScript во время загрузки страницы HTML будет недостаточно. Как правило, требуется управлять порядком исполнения отдельных участков JavaScript. Достигается это за счет оформления кода JavaScript в виде *функций*, которые запускаются по возникновению каких-либо *событий* (например, по щелчку кнопкой мыши) в странице HTML.

События в JavaScript и DOM

В следующем примере мы создадим код HTML из сценария JavaScript. При подготовке к созданию веб-страницы, которая имеет части, генерируемые динамически, в первую очередь требуется создать *шаблон* (который содержит статическую часть) и вставить в него заполнители, которые будут замещаться частями, создаваемыми динамически. Заполнители должны быть элементами HTML, имеющими уникальную идентификацию (элементы с атрибутом `ID`). Пока что в качестве заполнителя у нас выступал только элемент `<div>`, но в процессе чтения книги вы встретитесь с многочисленными примерами, в которых роль заполнителей отводится и другим элементам.

Рассмотрим следующий документ HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
```

```
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: Снова JavaScript и DOM</title>
    </head>
    <body>
        Привет, дружище! Ниже ты найдешь отличный список цветов:
        <br/>
        <ul>
            <li>Черный</li>
            <li>Оранжевый</li>
            <li>Розовый</li>
        </ul>
    </body>
</html>
```

Допустим, что вы хотите динамически генерировать содержимое элемента `<ul>`. В приложения AJAX это обычно достигается за счет размещения именованного пустого элемента `<div>` в нужном месте:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: Снова JavaScript и DOM</title>
    </head>
    <body>
        Привет, дружище! Ниже ты найдешь отличный список цветов:
        <br/>
        <div id="myDivElement"/>
    </body>
</html>
```

В этом примере элемент `<div>` нужен для размещения кода HTML, генерируемого сценарием JavaScript, но не забывайте, что можно взять элемент HTML любого типа, главное, чтобы он имел уникальный идентификатор. После того как код JavaScript добавит элемент `<ul>` в элемент `<div>`, получится следующая структура HTML:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: Снова JavaScript и DOM</title>
    </head>
    <body>
        Привет, дружище! Ниже ты найдешь отличный список цветов:
        <br/>
        <div id="myDivElement">
            <ul>
                <li>Черный</li>
                <li>Оранжевый</li>
```

```
                <li>Розовый</li>
            </ul>
        </div>
    </body>
</html>
```

Цели нашего следующего упражнения:

- Получить доступ к именованному элементу `<div>` программно, из функции JavaScript.
- Написать код JavaScript, который запускается *после* загрузки шаблона страницы, чтобы имелась возможность обращения к элементу `<div>` (ни один из элементов HTML не будет доступен из кода JavaScript, который запускается по ссылке из элемента `<head>`). Чтобы добиться этого, надо запускать код JavaScript из элемента `<body>` по событию `onload`.
- Сгруппировать код JavaScript в функции, чтобы с ним было проще работать.

Время действовать – использование событий JavaScript и DOM

1. В каталоге `foundations` (см. предыдущее упражнение) создайте подкаталог `morejsdom`.
2. В каталоге `morejsdom` создайте файл `morejsdom.html` и добавьте в него следующий код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: Снова JavaScript и DOM</title>
        <script type="text/javascript" src="morejsdom.js"></script>
    </head>
    <body onload="process()">
        Привет, дружище! Ниже ты найдешь отличный список цветов:
        <br />
        <div id="myDivElement" />
    </body>
</html>
```

3. Создайте новый файл `morejsdom.js` и добавьте в него следующий код:

```
function process()
{
    // Создать код HTML
    var string;
    string = "<ul>"
           + "<li>Черный</li>"
           + "<li>Оранжевый</li>"
           + "<li>Розовый</li>"
```

```
            + "</ul>";
        // Получить ссылку на элемент <div>
        myDiv = document.getElementById("myDivElement");
        // Добавить содержимое в элемент <div>
        myDiv.innerHTML = string;
    }
```

4. Загрузите страницу `morejsdom.html` в веб-броузер. Вы должны увидеть окно, подобное тому, что приводится на рис. 2.2.

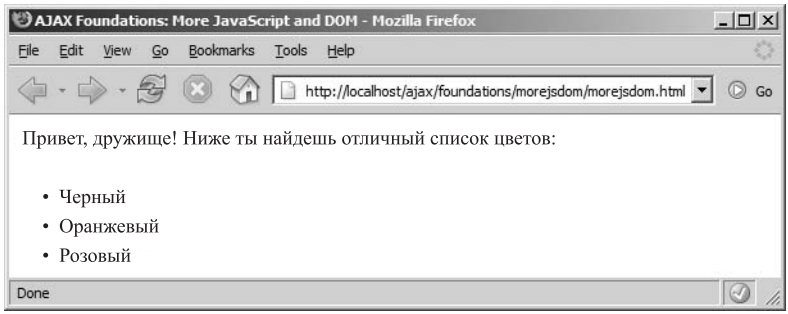

***Рис. 2.2.** Ваша маленькая страница HTML в действии*

Что происходит внутри?

Упражнение достаточно простое. Самые важные моменты в коде HTML выделены жирным шрифтом в следующем отрывке:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: Снова JavaScript и DOM</title>
        <script type="text/javascript" src="morejsdom.js"></script>
    </head>
    <body onload="process()">
        Привет, дружище! Ниже ты найдешь отличный список цветов:
        <br />
        <div id="myDivElement" />
    </body>
</html>
```

Все начинается с элемента `<script>`, содержащего ссылку на файл с исходным текстом сценария JavaScript. Файл JavaScript содержит функцию `process()`, обрабатывающую событие `onload` элемента `<body>`. Событие `onload` возникает после полной загрузки файла HTML. Таким образом, к моменту своего вызова функция `process()` может обратиться к любой части структуры HTML. Функция `process()` начинается с создания кода HTML, который должен быть добавлен в элемент `<div>`:

```
function process()
{
```

```
// Создать код HTML
var string;
string = "<ul>"
       + "<li>Черный</li>"
       + "<li>Оранжевый</li>"
       + "<li>Розовый</li>"
       + "</ul>";
```

Затем вызывается функция `getElementById` объекта `document`, возвращающая ссылку на элемент с именем `myDivElement`. Не забывайте, что `document` – это объект JavaScript по умолчанию, который ссылается на элемент `<body>` вашего документа HTML.

```
// Получить ссылку на элемент <div>
myDiv = document.getElementById("myDivElement");
```

Примечание

Обратите внимание, что JavaScript разрешает заключать описания строковых значений как в двойные, так и в одиночные кавычки. Предыдущая строка кода может быть с успехом изменена :

```
myDiv = document.getElementById('myDivElement');
```

В JavaScript оба варианта приемлемы, пока вы последовательно придерживаетесь какого-либо одного. Сочетая обе формы записи в одном и том же сценарии, вы рискуете столкнуться с ошибкой синтаксического анализа. В наших примерах программ JavaScript фигурируют двойные кавычки.

И в заключение производится заполнение элемента с именем `myDivElement` сгенерированным кодом HTML, который находится в переменной `string`:

```
        // Добавить содержимое в элемент <div>
        myDiv.innerHTML = string;
    }
```

В этом примере вы использовали свойство интерфейса DOM – `innerHTML`, в которое записали сконструированный вами документ HTML.

И еще о DOM

В предыдущем упражнении вы создали список элементов, объединяя текстовые строки для того, чтобы собрать простую структуру HTML. Та же самая структура может быть создана программно с помощью интерфейса DOM. В следующем упражнении аналогичная структура генерируется программно:

```
<div id="myDivElement">
    Привет, дружище! Ниже ты найдешь отличный список цветов:
    <br/>
    <ul>
```

```
            <li>Черный</li>
            <li>Оранжевый</li>
            <li>Розовый</li>
        </ul>
    </div>
```

Объектная модель документа представляет собой иерархическую структуру элементов, в которой каждый элемент может иметь один или более атрибутов. В данном фрагменте HTML дополнительным атрибутом обладает единственный элемент – `<div>`, это атрибут `id` со значением `myDivElement`. Корневым узлом структуры, к которому вы можете получить доступ, является элемент `<body>`. При реализации показанного выше документа HTML получится такая структура (рис. 2.3).

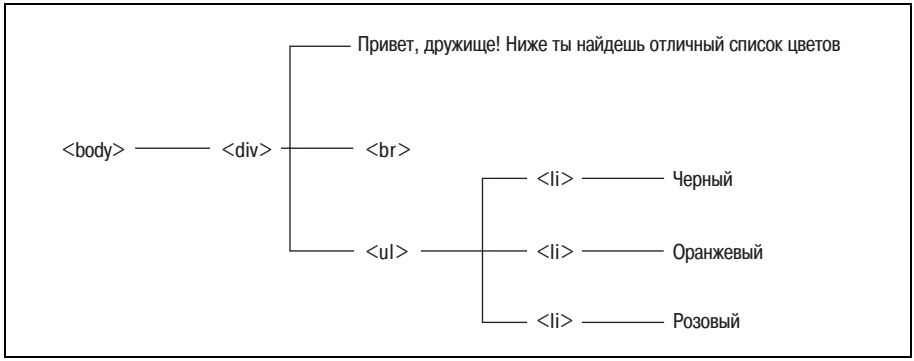

***Рис. 2.3.** Иерархия элементов HTML*

Из рисунка видно, что структура HTML сформирована элементами `<body>`, `<div>`, `<br>`, `<ul>` и `<li>` и четырьмя текстовыми узлами («Привет...», «Черный», «Оранжевый», «Розовый»). В следующем упражнении вы создадите эту структуру с помощью функций DOM `createElement`, `createTextNode` и `appendChild`.

Время действовать – и еще о DOM

1. В каталоге `foundations` создайте подкаталог `evenmorejsdom`.
2. В каталоге `evenmorejsdom` создайте файл `evenmorejsdom.html` и добавьте в него следующий код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: Еще больше о JavaScript и DOM</title>
        <script type="text/javascript" src="evenmorejsdom.js"></script>
    </head>
    <body onload="process()">
        <div id="myDivElement" />
```

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

3. Создайте новый файл `evenmorejsdom.js` и добавьте в него следующий код:

```
function process()
{
    // создать первый текстовый узел
    oHello = document.createTextNode
         ("Привет, дружище! Ниже ты найдешь отличный список цветов:");
    // создать элемент <ul>
    oUl = document.createElement("ul")
    // создать первый элемент <li> и добавить в него текстовый узел
    oLiBlack = document.createElement("li");
    oBlack = document.createTextNode("Черный");
    oLiBlack.appendChild(oBlack);
    // создать второй элемент <li> и добавить в него текстовый узел
    oLiOrange = document.createElement("li");
    oOrange = document.createTextNode("Оранжевый");
    oLiOrange.appendChild(oOrange);
    // создать третий элемент <li> и добавить в него текстовый узел
    oLiPink = document.createElement("li");
    oPink = document.createTextNode("Розовый");
    oLiPink.appendChild(oPink);
    // добавить элементы <li> как дочерние в элемент <ul>
    oUl.appendChild(oLiBlack);
    oUl.appendChild(oLiOrange);
    oUl.appendChild(oLiPink);
    // получить ссылку на элемент <div>
    myDiv = document.getElementById("myDivElement");
    // добавить содержимое в элемент <div>
    myDiv.appendChild(oHello);
    myDiv.appendChild(oUl);
}
```

4. Откройте страницу `evenmoredom.html` в веб-броузере. Результат должен выглядеть как на рис. 2.4.

Что происходит внутри?

Итак, в конечном результате мы получили то же самое, что и в предыдущем упражнении, но написали гораздо больше кода, в чем можно убедиться, взглянув на функцию `process()`. Она очень проста, хотя строк в ней стало намного больше. Это лишний раз доказывает, что применение DOM для создания структур HTML – это не всегда оптимальный выбор. Однако такой подход способен облегчить разработку страниц в следующих случаях:

- Если необходимо программно создавать динамические структуры HTML, например добавлять новые элементы в цикле `for`, когда важно не форматирование текста, а сами элементы структуры.

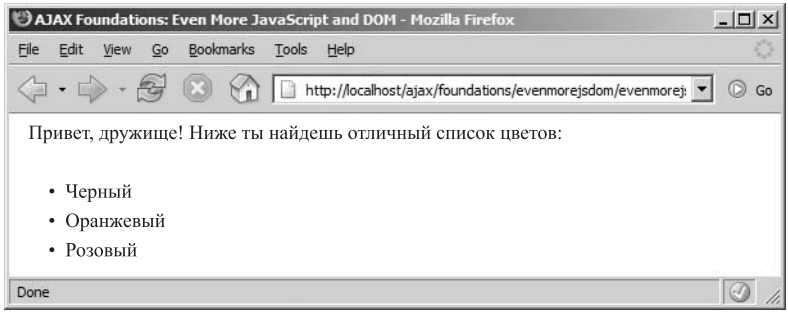

***Рис. 2.4.** Упражнение «Еще больше о JavaScript и DOM»*

- Когда необходимо сначала обработать все узлы независимо друг от друга, а потом принять решение о том, как строить иерархию. И снова DOM побеспокоится о тонкостях реализации – вам достаточно лишь будет сообщить ему, что вы хотите получить.
- И, как следствие, нет необходимости вручную добавлять закрывающие теги. Если, к примеру, добавить элемент `li`, то DOM сам сгенерирует пару тегов `<li>` и `</li>`.

JavaScript, DOM и CSS

Термин *каскадные таблицы стилей* (Cascading Style Sheets – CSS) уже должен быть вам знаком. CSS позволяют поместить описание форматирования страницы в отдельный документ, ссылка на который вставляется в файл HTML. Если все делать правильно, то CSS позволят вносить изменения в оформление всего сайта целиком (или его части) с наименьшими усилиями, путем изменения файла CSS. Каскадным таблицам стилей посвящено множество книг, включая свободно распространяемые, которые вы можете найти по адресу *http://www.w3.org/Style/css* и *http://www.w3schools.com/css/default.asp*. Статья, в которой впервые появилось название AJAX (*http://www.adaptivepath.com/publications/essays/archives/000385.php*), упоминает CSS как одну из составных частей AJAX, но технической необходимости в применении CSS для создания успешных динамических веб-приложений нет. Однако это настоятельно рекомендуется делать из-за преимуществ, которые они обеспечивают.

Рассмотрим простое упражнение, демонстрирующее использование CSS и принципы манипулирования стилями отображения элементов HTML посредством объектной модели документа. Подобные задачи часто встречаются при разработке приложений AJAX. В следующем упражнении мы нарисуем таблицу и добавим в страницу две кнопки, «Стиль 1» и «Стиль 2». Нажатие этих кнопок будет приводить к изменению цвета и внешнего вида таблицы за счет изменения стиля ее отображения.

Время действовать – работаем с CSS и JavaScript

1. В каталоге `foundations` создайте новый подкаталог с именем `csstest`.
2. В каталоге `csstest` создайте новый файл `csstest.html` и добавьте в него следующий код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: CSS</title>
        <script type="text/javascript" src="csstest.js"></script>
        <link href="styles.css" type="text/css" rel="stylesheet"/>
    </head>
    <body>
        <table id="table">
            <tr>
                <th id="tableHead">
                    Название продукта
                </th>
            </tr>
            <tr>
                <td id="tableFirstLine">
                    Самолет
                </td>
            </tr>
            <tr>
                <td id="tableSecondLine">
                    Автомобиль
                </td>
            </tr>
        </table>
        <br />
        <input type="button" value="Стиль 1" onclick="setStyle1();" />
        <input type="button" value="Стиль 2" onclick="setStyle2();" />
    </body>
</html>
```

3. Создайте новый файл `csstest.js` и добавьте в него следующий код:

```
// Отобразить таблицу с применением стиля 1
function setStyle1()
{
    // получить ссылки на элементы HTML
    oTable = document.getElementById("table");
    oTableHead = document.getElementById("tableHead");
    oTableFirstLine = document.getElementById("tableFirstLine");
    oTableSecondLine = document.getElementById("tableSecondLine");
    // применить стили
    oTable.className = "Table1";
    oTableHead.className = "TableHead1";
    oTableFirstLine.className = "TableContent1";
    oTableSecondLine.className = "TableContent1";
```

```
}

// Отобразить таблицу с применением стиля 2
function setStyle2()
{
    // получить ссылки на элементы HTML
    oTable = document.getElementById("table");
    oTableHead = document.getElementById("tableHead");
    oTableFirstLine = document.getElementById("tableFirstLine");
    oTableSecondLine = document.getElementById("tableSecondLine");
    // применить стили
    oTable.className = "Table2";
    oTableHead.className = "TableHead2";
    oTableFirstLine.className = "TableContent2";
    oTableSecondLine.className = "TableContent2";
}
```

4. **В заключение создайте новый файл CSS** – `styles.css`**:**

```
.Table1
{
    border: DarkGreen 1px solid;
    background-color: LightGreen;
}
.TableHead1
{
    font-family: Verdana, Arial;
    font-weight: bold;
    font-size: 10pt;
}
.TableContent1
{
    font-family: Verdana, Arial;
    font-size: 10pt;
}

.Table2
{
    border: DarkBlue 1px solid;
    background-color: LightBlue;
}
.TableHead2
{
    font-family: Verdana, Arial;
    font-weight: bold;
    font-size: 10pt;
}
.TableContent2
{
    font-family: Verdana, Arial;
    font-size: 10pt;
}
```

5. Откройте в броузере страницу *http://localhost/ajax/foundations/csstest/csstest.html* и убедитесь, что нажатие кнопок изменяет стиль отображения таблицы (рис. 2.5).

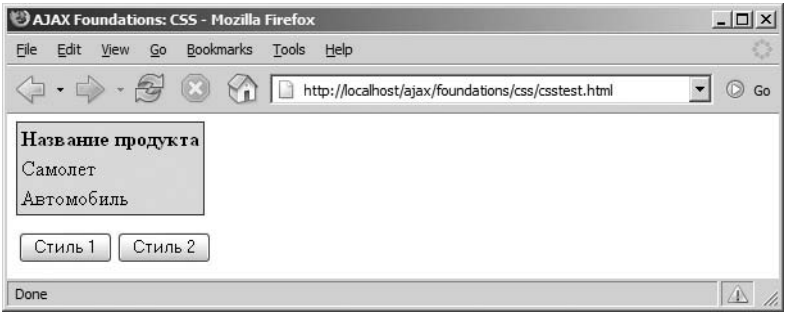

Рис. 2.5. *Рисование таблицы с применением CSS и JavaScript*

Что происходит внутри?

Файл `styles.css` содержит два набора стилей, которые применяются к таблице из файла `csstest.html`. Когда пользователь нажимает кнопки «Стиль...» (SetStyle), сценарий JavaScript применяет выбранный стиль к элементам таблицы, используя функции DOM.

В первой части методов `SetStyle` мы вызываем функцию `getElementByID`. С ее помощью мы получаем ссылки на элементы HTML, к которым следует применить стили отображения:

```
// получить ссылки на элементы HTML
oTable = document.getElementById("table");
oTableHead = document.getElementById("tableHead");
oTableFirstLine = document.getElementById("tableFirstLine");
oTableSecondLine = document.getElementById("tableSecondLine");
```

Примечание

Как и во многих других случаях, связанных с разработкой веб-приложений, манипулируя таблицами CSS, можно столкнуться с тем, что не все веб-броузеры поддерживают их одинаково хорошо. Попробуйте в предыдущем фрагменте кода изменить названия объектов так, чтобы они совпадали с именами соответствующих им элементов HTML (например, измените имя `oTable` на `Table`), и IE не будет работать. Он очень не любит, когда имена объектов совпадают с названиями элементов HTML. Трудно понять, в чем тут дело, поскольку области видимости объектов и элементов не пересекаются, но если вы хотите, чтобы ваш код работал и в Internet Explorer, вам придется учитывать это обстоятельство.

После инициализации объектов следующий отрывок кода, который применяет стили отображения, одинаково хорошо работает во всех броузерах:

```
// применить стили
oTable.className = "Table1";
oTableHead.className = "TableHead1";
oTableFirstLine.className = "TableContent1";
oTableSecondLine.className = "TableContent1";
```

Использование объекта XMLHttpRequest

Объект XMLHttpRequest дает возможность отправлять асинхронные запросы HTTP из кода JavaScript. Он позволяет отправлять запросы HTTP, принимать ответы и обновлять отдельные части страницы полностью в фоновом режиме, не прерывая работу пользователя. Это очень важно, поскольку позволяет сохранить отклик пользовательского интерфейса во время обращений к серверу за новыми порциями данных.

Изначально объект XMLHttpRequest был разработан в Microsoft в 1999 году как объект ActiveX для Internet Explorer и в конечном счете стал стандартом *де факто* для всех остальных броузеров. Сейчас он поддерживается всеми броузерами как встроенный объект, за исключением Internet Explorer 6.

Примечание

Обратите внимание, что, хотя XMLHttpRequest фактически стал стандартом в броузерах, он не является стандартом W3C. Аналогичная функциональность, предусматриваемая стандартом W3C «DOM Level 3 Load and Save», до сих пор не реализована в броузерах.

При работе с объектом XMLHttpRequest обычно выполняется такая последовательность действий:

1. Создается экземпляр объекта XMLHttpRequest.
2. С помощью объекта XMLHttpRequest создается асинхронный запрос к серверу, определяется функция обратного вызова, которая будет автоматически вызвана в случае приема ответа от сервера.
3. Функция обратного вызова обрабатывает ответ сервера.
4. Переход к пункту 2.

А теперь рассмотрим эти действия на реальном примере.

Создание объекта XMLHttpRequest

В разных броузерах объект XMLHttpRequest реализован по-разному. В Internet Explorer 6 и в более старых версиях XMLHttpRequest реализован в виде элемента управления ActiveX, который следует создавать так:

```
xmlhttp = new ActiveXObject("Microsoft.XMLHttp");
```

Во всех остальных броузерах он реализован в виде встроенного объекта, поэтому его экземпляр создается иначе:

```
xmlhttp = new XMLHttpRequest();
```

Примечание

Библиотека ActiveX XMLHttp имеет столько версий и разновидностей, что и представить невозможно. Каждая часть программного обеспечения от Microsoft, включая Internet Explorer и MDAC, распространяется со своей версией этого элемента управления. Объект `Microsoft.XMLHTTP` является самым старым и может использоваться повсюду для выполнения основных операций, однако более новые версии обладают более высокой производительностью и предлагают расширенные функциональные возможности. Позднее вы узнаете, как автоматически создавать объект наиболее свежей версии.

Во всех упражнениях из этой книги фигурирует упрощенная версия кода, создающего объект `XMLHttpRequest` в любом типе броузеров:

```
// Создать экземпляр XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        try
        {
            xmlHttp = new ActiveXObject("Microsoft.XMLHttp");
        }
        catch(e) { }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}
```

Предполагается, что эта функция возвращает экземпляр объекта `XMLHttpRequest`. Работа этой функции построена на основе конструкции `try/catch`, имеющейся в арсенале JavaScript.

Конструкция `try/catch` первоначально была реализована в объектно-ориентированных языках программирования и предоставляет чрезвычайно мощную возможность обработки исключительных ситуаций в JavaScript. Когда в ходе исполнения кода JavaScript возникает ошибка,

возбуждается исключение. Исключение представляет собой объект, содержащий описание возникшей ошибки (исключительной ситуации). Конструкция try/catch позволяет *перехватить* (*catch*) исключение и обработать его так, чтобы сообщение об ошибке не выводилось в окне броузера перед пользователем.

Типичный пример применения конструкции try/catch:

```
try
{
    // код, который может породить исключение
}
catch (e)
{
    // код, который исполняется только в случае возбуждения
    // исключительной ситуации в блоке try
    // (описание исключения доступно через параметр e)
}
```

Любой код, который может породить исключение, помещается в блок try. Если возникает какая-либо ошибка, управление немедленно передается в блок catch. Если в блоке try ошибок не возникло, управление никогда не попадет в блок catch.

Во время исполнения исключения передаются от точки их появления вверх по стеку вызовов, пока не будут перехвачены первым встретившимся блоком catch. Если исключительная ситуация не будет обработана приложением, она будет перехвачена самим броузером, который может вывести перед вашим посетителем не очень дружественное сообщение об ошибке.

Способ обработки исключения зависит от сложившейся ситуации. Можно проигнорировать ошибку, можно установить какой-либо флаг, а можно вывести сообщение об ошибке для посетителя. В этой книге вы встретите все эти способы обработки исключений.

Создавая объект XMLHttpRequest, мы в данном случае сначала пробуем создать его экземпляр, как если бы он был встроенным объектом броузера:

```
// эта часть кода должна работать во всех броузерах, за исключением
// IE6 и более старых его версий
try
{
    // попытаться создать объект XMLHttpRequest
    xmlHttp = new XMLHttpRequest();
}
```

Internet Explorer 7, Mozilla, Opera и другие броузеры прекрасно справляются с этой частью кода и не порождают ошибок, поскольку XMLHttpRequest поддерживается ими как встроенный объект. Однако Internet Explorer 6 и более старые его версии не могут распознать объект XMLHttpRequest и возбуждают исключение; в результате управление пере-

дается в блок `catch`. Для броузера Internet Explorer 6 и более старых его версий объект `XMLHttpRequest` должен создаваться как элемент управления ActiveX:

```
catch(e)
{
    // предполагается, что в качестве броузера используется
    // IE6 или более старая его версия
    try
    {
        xmlHttp = new ActiveXObject("Microsoft.XMLHttp");
    }
    catch(e) { }
}
```

Сколько программистов, работающих с JavaScript, столько можно встретить методов создания объекта `XMLHttpRequest`, и, что самое удивительное, все они прекрасно работают. В этой книге мы предпочитаем метод, основанный на конструкции `try/catch`, поскольку, на наш взгляд, он должен будет справляться с возложенной на него задачей и в будущих версиях броузеров, выполняя проверку на наличие ошибок небольшим объемом кода.

Возможен такой метод, основанный на проверке типа броузера перед попыткой создания объекта `XMLHttprequest`, с помощью функции `typeof`:

```
if (typeof XMLHttpRequest != "undefined")
    xmlHttp = new XMLHttpRequest();
```

Функция `typeof` нередко бывает весьма полезной. Но в нашем случае функция `typeof` не устраняет необходимость предпринимать меры по обработке ошибок с помощью конструкции `try/catch`, таким образом, применение данного метода привело бы лишь к увеличению объема кода.

Альтернативный способ достичь той же самой функциональности основывается на особенности JavaScript, которая называется **определением объекта (object detection)**. Она позволяет проверить, поддерживается ли тот или иной объект данным броузером. Проверка может быть выполнена так:

```
if (window.XMLHttpRequest)
    xmlHttp = new XMLHttpRequest();
```

Например, проверив поддержку объекта `window.ActiveX`, можно определить, что броузер принадлежит к семейству Internet Explorer. Но и эта методика может увеличить объем кода, не давая взамен никаких преимуществ. Сама идея, однако, достойна, чтобы ее запомнить.

Если вы остановитесь на методике определения объектов, тогда в первую очередь проверяйте поддержку объекта `XMLHttpRequest` и лишь потом поддержку ActiveX. Причина таких рекомендаций – Internet Explorer 7, который поддерживает как ActiveX, так и `XMLHttpRequest`, причем последний вариант предпочтительнее, поскольку предоставляет

объект более свежей версии. Как вы увидите позже, в случае применения ActiveX потребуется не так много кода, чтобы обеспечить использование одной из последних версий объекта, хотя нет никаких гарантий, что полученная вами версия будет самой свежей.

В конце нашей функции `createXmlHttpRequestObject`, завершив все действия, мы проверяем, удалось ли создать экземпляр объекта `XMLHttpRequest`:

```
// вернуть созданный объект или вывести сообщение об ошибке
if (!xmlHttp)
    alert("Ошибка создания объекта XMLHttpRequest.");
else
    return xmlHttp;
```

Примечание

Побочный эффект от определения объекта даже еще лучше, чем сама эта возможность. Функция определения объекта гарантирует, что JavaScript вычислит правильность экземпляра объекта и запишет в него (`xmlHttp`) значение `true`. Это особенно важно, поскольку выражение `(!xmlHttp)` возвращает значение `true`, не только когда в `xmlHttp` содержится значение `false`, но также если в нем находится значение `nil` или `undefined`.

Создание объекта более поздней версии в Internet Explorer

В функции `createXmlHttpRequestObject` возможно единственное улучшение – ее можно заставить распознавать последнюю версию элемента управления ActiveX в случае исполнения в броузере Internet Explorer 6. Как правило, вполне достаточно базовой функциональности, заложенной в объект `ActiveX("Microsoft.XMLHttp")`, но если вы захотите попытаться заполучить более свежую версию объекта, то можете сделать это.

Типичное решение этой задачи заключается в том, чтобы попытаться создать объект самой последней из известных версий и, если эта попытка не увенчается успехом, проигнорировать ошибку и повторить попытку, но уже для более старой версии, и так до тех пор, пока объект не будет создан. Последний *prog ID* объекта `XMLHTTP ActiveX Object` – `MSXML2.XMLHTTP.6.0`. Более подробно об идентификаторах `progID` и о том хаосе, который стоит за ними, можно узнать по адресу *http://puna.net.nz/etc/xml/msxml.htm*.

Ниже приводится обновленная версия функции `createXmlHttpRequestObject`. Новые строки выделены жирным шрифтом.

```
// создает экземпляр XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
```

```
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array('MSXML2.XMLHTTP.6.0',
                                        'MSXML2.XMLHTTP.5.0',
                                        'MSXML2.XMLHTTP.4.0',
                                        'MSXML2.XMLHTTP.3.0',
                                        'MSXML2.XMLHTTP',
                                        'Microsoft.XMLHTTP');
        // пытаться создавать объект наиболее свежей версии
        // пока одна из попыток не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {} // игнорировать возможные ошибки
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}
```

Несмотря на то, что этот код выглядит несколько замысловато, на самом деле он ничего сложного не делает. В первую очередь он пытается создать объект `MSXML2.XMLHttp.6.0 ActiveX`. Если попытка терпит неудачу, ошибка игнорируется (обратите внимание на пустой блок `catch`) и предпринимается попытка создать объект `MSXML2.XMLHttp.5.0` и т. д. Эти операции продолжаются до тех пор, пока одна из попыток создания объекта не завершится успехом.

Вероятно, наиболее интересный момент в новом коде – это способ использования механизма определения объекта (`!xmlHttp`), при котором мы останавливаем поиск новых prog ID после попытки создания объекта и прерываем исполнение цикла `for`.

Инициализация запросов к серверу с помощью объекта XMLHttpRequest

Создав объект XMLHttpRequest, можно производить весьма интересные манипуляции. Экземпляр этого объекта может быть создан разными способами, в зависимости от версии броузера, однако все экземпляры XMLHttpRequest, как предполагается, имеют идентичный прикладной программный интерфейс (Application Programming Interface – API) и обладают идентичной функциональностью. (На самом деле это не гарантируется, поскольку каждый броузер имеет свою собственную реализацию.)

Наиболее интересные сведения об объекте XMLHttpRequest вы узнаете на практике, а пока мы приведем краткую справку по свойствам и методам этого объекта:

Метод/свойство	Описание
abort()	Останавливает исполнение текущего запроса.
getAllResponseHeaders()	Возвращает заголовки ответов в виде строки.
getResponseHeader("headerLabel")	Возвращает заголовок одного ответа в виде строки.
open("method", "URL"[, asyncFlag[,"userName"[, "password"]]])	Инициализирует параметры запроса.
send(content)	Выполняет запрос HTTP.
setRequestHeader("label", "value")	Назначает пару label/value в заголовке запроса.
onreadystatechange	Используется для установки функции обратного вызова, которая будет обслуживать изменение состояния запроса.
readyState	Возвращает состояние запроса: 0 = не инициализирован 1 = идет отправка запроса 2 = запрос отправлен 3 = идет обмен 4 = завершен
responseText	Возвращает ответ сервера в виде строки.
responseXML	Возвращает ответ сервера в виде документа XML.
status	Возвращает код состояния запроса.
statusText	Возвращает сообщение о состоянии запроса.

При каждом обращении к серверу задействуются методы open и send. Метод open задает конфигурацию запроса, определяя его параметры, а метод send отправляет запрос (обеспечивает доступ к серверу). Если

запрос выполняется в асинхронном режиме, то перед вызовом метода send надо также определить свойство onreadystatechange, указав функцию обратного вызова, которая будет исполнена при изменении состояния запроса, обеспечив, таким образом, поддержку механизма AJAX.

Метод open предназначен для инициализации запроса. Ему передаются два обязательных и несколько необязательных аргументов. Метод open не создает соединение с сервером, он лишь определяет параметры соединения. Первый аргумент указывает метод, который будет применяться для передачи данных серверу, и может принимать значения GET, POST или PUT. Второй аргумент URL, он определяет адрес, по которому должен быть отправлен запрос. URL может быть как полным, так и относительным. Если URL говорит о том, что доступ к ресурсу будет осуществляться не через HTTP, то значение первого аргумента игнорируется.

Третий аргумент, async, определяет режим исполнения запроса – синхронный или асинхронный. Если он равен true, то после вызова метода send() управление немедленно будет возвращено сценарию, не дожидаясь получения ответа от сервера, а если false, то метод send() будет ожидать ответа от сервера, прежде чем вернуть управление, приостанавливая работу веб-страницы. Чтобы установить режим асинхронной работы, необходимо передать в аргументе async значение true и установить обработчик события onreadystatechange, который будет заниматься обработкой ответа сервера.

В случае применения метода GET параметры передаются серверу в строке URL запроса примерно так: *http://localhost/ajax/test.php?param1=x&param2=y*. В данном примере серверу передаются два параметра: первый с именем param1 и значением x и второй с именем param2 и значением y.

```
// обратиться к серверу, чтобы выполнить операцию на стороне сервера
xmlHttp.open("GET", "http://localhost/ajax/test.php?param1=x&param2=y", true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
```

Если параметры передаются методом POST, то они не присоединяются к строке URL запроса, а передаются методу send() в виде строки:

```
// обратиться к серверу, чтобы выполнить операцию на стороне сервера
xmlHttp.open("POST", "http://localhost/ajax/test.php", true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send("param1=x&param2=y");
```

Эти два отрывка кода выполняют идентичные действия. На практике метод GET чаще всего применяется для отладки, поскольку его нетрудно эмулировать с помощью веб-броузера и увидеть своими глазами, что именно возвращает сценарий, исполняющийся на стороне сервера. Метод POST необходим, когда объем передаваемых данных превышает 512 байт, поскольку этот объем нельзя отправить методом GET.

В наших упражнениях мы поместили код, выполняющий запросы HTTP, внутрь функции process(). В минимальном объеме ее реализация, не предусматривающая обработки ошибочных ситуаций, выглядит так:

```
function process()
{
   // обратиться к серверу, чтобы выполнить операцию на стороне сервера
    xmlHttp.open("GET", "server_script.php", true);
    xmlHttp.onreadystatechange = handleRequestStateChange;
    xmlHttp.send(null);
}
```

С этим методом могут быть связаны следующие трудности:

- Функция process() может быть исполнена, даже если переменная xmlHttp не содержит ссылку на корректный экземпляр объекта XMLHttpRequest. Это может произойти, например, если броузер пользователя не поддерживает объект XMLHttpRequest. В результате может возникнуть исключительная ситуация, в которой даже наши усилия по обработке ошибок окажутся бесполезными, если мы не будем последовательны и не предусмотрим специальную обработку внутри функции process().
- Функция process() не защищена от появления других типов ошибок. Например, в конце этой главы вы увидите, что некоторые броузеры возбуждают исключительные ситуации из соображений безопасности, если им «не нравится» сервер, обращение к которому производится с помощью объекта XMLHttpRequest (подробнее о вопросах безопасности мы поговорим в главе 3).

Ниже приводится более безопасная версия функции process():

```
// вызывается для чтения файла с сервера
function process()
{
    // продолжать только если xmlHttp не содержит пустую ссылку
    if (xmlHttp)
    {
        // попытаться соединиться с сервером
        try
        {
            // инициировать чтение файла с сервера
            xmlHttp.open("GET", "server_script.php", true);
            xmlHttp.onreadystatechange = handleRequestStateChange;
            xmlHttp.send(null);
        }
        // вывести сообщение об ошибке в случае неудачи
        catch (e)
        {
            alert("Невозможно соединиться с сервером:\n" + e.toString());
        }
    }
```

```
}
```

Если в переменной `xmlHttp` содержится значение `null` (или `false`), мы не выводим никакого сообщения, т. к. предполагается, что оно уже было выведено функцией `createXmlHttpRequestObject`. Однако сообщение будет выведено в случае появления каких-либо других проблем при попытке установить соединение.

Обработка ответа сервера

При выполнении асинхронных запросов (как в отрывках кода, представленных ранее) метод `xmlHttp.send()` не приостанавливает работу приложения в ожидании прибытия ответа от сервера. Обработкой события изменения состояния запроса занимается функция обратного вызова `handleRequestStateChange`. Обычно она вызывается четыре раза, т. е. каждый раз, когда изменяется состояние запроса. Запомните: свойство `readyState` может иметь одно из следующих значений:

```
0 = не инициализирован
1 = идет отправка запроса
2 = запрос отправлен
3 = идет обмен
4 = завершен
```

Смысл всех состояний, за исключением 3, очевиден из этого описания. Состояние «*идет обмен*» имеет место, когда ответ сервера частично уже принят. В наших приложениях AJAX встречается только состояние «*завершен*», которое сообщает о том, что ответ сервера был получен полностью.

Типичная реализация функции `handleRequestStateChange` показана в следующем отрывке кода, где жирным шрифтом выделен участок, непосредственно выполняющий чтение ответа сервера:

```
// функция исполняется, когда изменяется состояние запроса
function handleRequestStateChange()
{
    // продолжить, если процесс завершен
    if (xmlHttp.readyState == 4)
    {
        // продолжить только если статус HTTP равен "OK"
        if (xmlHttp.status == 200)
        {
            // получить ответ
            response = xmlHttp.responseText;
            // (для чтения ответа в формате XML, как объект DOM,
            // используйте xmlHttp.responseXML)
            // обработать полученный ответ
            // ...
            // ...
        }
```

```
    }
}
```

Здесь мы также с успехом можем применить конструкцию `try/catch` для обработки ошибок, которые могут возникнуть при чтении ответа сервера или инициализации нового запроса.

Вот более безопасная реализация функции `handleRequestStateChange`:

```
// функция исполняется, когда изменяется состояние запроса
function handleRequestStateChange()
{
    // продолжить, если процесс завершен
    if (xmlHttp.readyState == 4)
    {
        // продолжить только если статус HTTP равен "OK"
        if (xmlHttp.status == 200)
        {
            try
            {
                // получить ответ
                response = xmlHttp.responseText;
                // обработать полученный ответ
                // ...
                // ...
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                alert("Ошибка чтения ответа сервера: " + e.toString());
            }
        }
        else
        {
            // вывести сообщение о состоянии
            alert("Возникли проблемы при получении данных:\n" +
                  xmlHttp.statusText);
        }
    }
}
```

Отлично, теперь посмотрим на эти функции в действии.

Время действовать – асинхронные обращения с использованием объекта XMLHttpRequest

1. В каталоге `foundations` создайте подкаталог `async`.
2. В каталоге `async` создайте файл с именем `async.txt` и добавьте в него следующий текст: «Привет, клиент!»
3. В том же самом каталоге создайте файл с именем `async.html` и добавьте в него следующий код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: использование объекта XMLHttpRequest</title>
        <script type="text/javascript" src="async.js"></script>
    </head>
    <body onload="process()">
        Привет, сервер!
        <br/>
        <div id="myDivElement" />
    </body>
</html>
```

4. Создайте файл с именем `async.js` **и добавьте в него следующий код:**

```
// в этой переменной хранится ссылка на экземпляр XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// создает экземпляр XMLHttpRequest
function createXmlHttpRequestObject()
{
    // для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // этот участок кода работает во всех броузерах,
    // за исключением IE6 и более старых версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предположительно IE6 или более старой версии
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
```

```
            alert("Ошибка создания объекта XMLHttpRequest.");
        else
            return xmlHttp;
    }

    // вызывается для чтения файла с сервера
    function process()
    {
        // продолжать только если в xmlHttp не пустая ссылка
        if (xmlHttp)
        {
            // попытаться установить соединение с сервером
            try
            {
                // инициировать чтение файла async.txt с сервера
                xmlHttp.open("GET", "async.txt", true);
                xmlHttp.onreadystatechange = handleRequestStateChange;
                xmlHttp.send(null);
            }
            // вывести сообщение об ошибке в случае неудачи
            catch (e)
            {
                alert("Невозможно соединиться с сервером:\n" + e.toString());
            }
        }
    }

    // функция обработки ответа HTTP
    function handleRequestStateChange()
    {
        // получить ссылку на элемент <div> на странице
        myDiv = document.getElementById("myDivElement");
        // вывести состояние запроса
        if (xmlHttp.readyState == 1)
        {
            myDiv.innerHTML += "Состояние запроса: 1 (отправляется) <br/>";
        }
        else if (xmlHttp.readyState == 2)
        {
            myDiv.innerHTML += " Состояние запроса: 2 (отправлен) <br/>";
        }
        else if (xmlHttp.readyState == 3)
        {
            myDiv.innerHTML += " Состояние запроса: 3 (идет обмен) <br/>";
        }
        // когда readyState = 4, мы можем прочитать ответ сервера
        else if (xmlHttp.readyState == 4)
        {
            // продолжать, только если статус HTTP равен "OK"
            if (xmlHttp.status == 200)
            {
                try
```

```
                {
                    // прочитать ответ сервера
                    response = xmlHttp.responseText;
                    // вывести сообщение
                    myDiv.innerHTML +=
                    " Состояние запроса: 4 (завершен). Сервер ответил: <br/>";
                    myDiv.innerHTML += response;
                }
                catch(e)
                {
                    // вывести сообщение об ошибке
                    alert("Ошибка чтения ответа: " + e.toString());
                }
            }
            else
            {
                // вывести сообщение о состоянии
                alert("Возникли проблемы во время получения данных:\n" +
                      xmlHttp.statusText);
            }
        }
    }
```

5. Откройте в броузере страницу *http://localhost/ajax/foundations/async/async.html* (эту страницу надо открыть через HTTP, потому что на этот раз прием с доступом как к локальному файлу неэффективен). Вы должны получить результат, показанный на рис. 2.6.

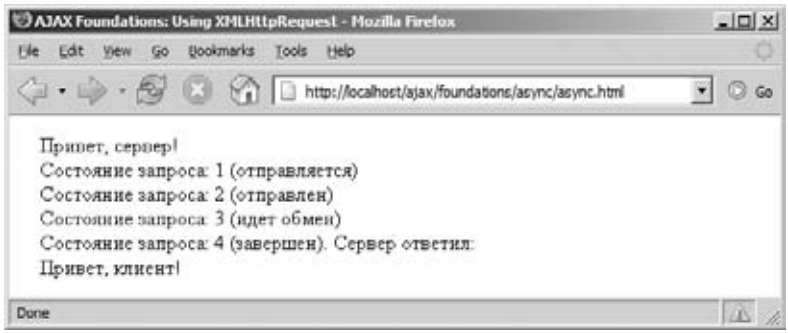

***Рис. 2.6.** Четыре кода состояния запроса*

Примечание

Не волнуйтесь, если броузер отобразит не все коды состояния. Некоторые реализации `XMLHttpRequest` просто игнорируют отдельные коды состояния. Например, Opera генерирует событие, только если код состояния равен 3 или 4. Internet Explorer сообщает о кодах состояния 2, 3 и 4 при использовании наиболее свежих версий `XMLHttp`.

Что происходит внутри?

Чтобы разобраться с тем, что происходит, начнем с самого начала – с файла `async.html`:

```
<html>
    <head>
        <title>AJAX Foundations: Using XMLHttpRequest</title>
        <script type="text/javascript" src="async.js"></script>
    </head>
    <body onload="process()">
```

Этот небольшой фрагмент кода таит в себе весьма интересную функциональность. Прежде всего он содержит ссылку на файл `async.js`, в этот момент производится синтаксический анализ файла сценария. Обратите внимание: код JavaScript, находящийся внутри функций, не исполняется автоматически, зато исполняется та часть кода, которая находится вне функций. Весь код в нашем файле JavaScript помещен в функции, за исключением единственной строки:

```
// в этой переменной хранится ссылка на экземпляр XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
```

Таким способом мы гарантируем, что с самого начала переменная `xmlHttp` будет содержать ссылку на экземпляр объекта `XMLHttpRequest`. Экземпляр объекта создается обращением к функции `createXmlHttpRequestObject`, с которой мы уже встречались ранее.

По событию `onload` вызывается метод `process()`. Этот метод уже может работать с инициализированным объектом `xmlHttp`, поэтому он концентрируется только на инициализации запроса к серверу. Для защиты от возможных проблем предусматривается код, занимающийся обработкой ошибок. Код, инициирующий запрос к серверу:

```
// инициировать чтение файла async.txt с сервера
xmlHttp.open("GET", "async.txt", true);
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
```

Примечание

Обратите внимание, что сценарий не удастся загрузить как локальный файл с диска, используя ресурс `file://`. Вам потребуется выполнить загрузку через HTTP. Чтобы загрузить сценарий как локальный файл, надо указать полный путь доступа к файлу `.txt`, а это может повлечь за собой проблемы, связанные с настройками безопасности в броузере, о которых мы еще поговорим.

Предположим, что запрос HTTP был успешно инициализирован и исполнен в асинхронном режиме, в этом случае метод `handleRequestStateChange` будет вызываться всякий раз, когда будет изменяться состояние запроса. В реальных приложениях все состояния, за исключением `4` (которое сигнализирует о завершении запроса), игнорируются, но в дан-

ном упражнении мы выводим сообщения при каждом изменении состояния, чтобы показать, что функция обратного вызова работает именно так, как мы и предполагали.

Код в `handleRequestStateChange` не представляет собой ничего особенного, но сам факт, что он вызывается, интересен. Можно не ждать ответа сервера, выполняя синхронные запросы HTTP, а решать другие задачи (в ожидании ответа на выполняемый асинхронный запрос).

Работа функции `handleRequestStateChange` начинается с получения ссылки на элемент HTML с именем `myDivElement`, предназначенный для отображения различных состояний запроса HTTP в процессе его исполнения:

```
// функция обработки ответа HTTP
function handleRequestStateChange()
{
    // получить ссылку на элемент <div> на странице
    myDiv = document.getElementById("myDivElement");
    // вывести состояние запроса
    if (xmlHttp.readyState == 1)
    {
        myDiv.innerHTML += "Состояние запроса: 1 (отправляется) <br/>";
    }
    else if (xmlHttp.readyState == 2)
    ...
    ...
```

Для обработки состояния запроса со значением `4` мы предусматриваем типичный код, который читает ответ сервера, спрятанный внутри `xmlHttp.ResponseText`:

```
// если readyState = 4, мы можем прочитать ответ сервера
else if (xmlHttp.readyState == 4)
{
    // продолжать, только если статус HTTP равен "OK"
    if (xmlHttp.status == 200)
    {
        try
        {
            // прочитать ответ сервера
            response = xmlHttp.responseText;
            // вывести сообщение
            myDiv.innerHTML +=
            " Состояние запроса: 4 (завершен). Сервер ответил: <br/>";
            myDiv.innerHTML += response;
        }
        catch(e)
        {
            // вывести сообщение об ошибке
            alert("Ошибка чтения ответа: " + e.toString());
        }
    }
```

```
        else
        {
            // вывести сообщение о состоянии
            alert("Возникли проблемы во время получения данных:\n" +
            xmlHttp.statusText);
        }
    }
```

Кроме кода, предназначенного для обработки ошибок, неплохо было бы обратить внимание на вызов метода `xmlHttp.responseText`, отвечающий за чтение ответа сервера. Этот метод имеет родственный ему `xmlHttp.responseXml`, применяемый в случаях, когда ответ от сервера поступает в формате XML.

Примечание

За исключением метода `responseXml` объекта `XMLHttpRequest`, язык разметки XML больше нигде не участвует, разве только в имени объекта (только что рассмотренное упражнение – яркий пример этому). Более подобающее название для объекта было бы `HttpRequest`. Префикс XML, вероятно, был добавлен Microsoft просто потому, что тогда слово XML только входило в обиход и оно казалось очень умным и модным, как, например, слово AJAX сейчас. Не удивляйтесь, если вскоре вам встретятся объекты с именем `AjaxRequest` (или что-то вроде этого).

Работа со структурой XML

Документы XML напоминают документы HTML в том смысле, что оба этих формата основаны на текстовом представлении и содержат иерархии элементов. За последние несколько лет XML достиг небывалой популярности при решении задач инкапсуляции и передачи данных.

Кстати, в названии AJAX от XML присутствует буква X, и префикс в имени объекта `XMLHttpRequest`. Однако надо понимать, что использование XML в этой технологии совершенно необязательно. В предыдущем упражнении мы создали простое приложение, выполняющее асинхронные обращения к серверу с целью получения простого текстового документа, и не привлекали XML.

Примечание

XML – это чрезвычайно широкий предмет для обсуждения, включающий в себя множество дополнительных технологий. Вам наверняка встречались специалисты, которые толкуют о DTD, схемах и пространствах имен, XSLT и XPath, XLink и XPointer и о многих других вещах, имеющих отношение к XML. В этой книге мы будем использовать XML в основном для передачи простых структур данных. В качестве быстрого введения в язык XML рекомендуем прочитать *http://www.xmlnews.org/docs/xml-basics.html*. И если вы не возражаете против узкоспециального материала, рекомендуем отличный ресурс *http://www.w3schools.com/xml/default.asp*. В приложении C, расположенном по адресу *http://ajax-php.packtpub.com*, вы найдете введение в XSLT и XPath.

Для работы с файлами XML может применяться интерфейс DOM, точно так же, как и для работы с файлами HTML. Следующее упражнение похоже на предыдущее в том смысле, что оно читает статический файл с сервера. Новое в нем то, что файл имеет формат XML, а чтение его производится с привлечением интерфейса DOM.

Время действовать – выполнение асинхронных обращений с помощью объекта XMLHttpRequest и XML

1. В каталоге `foundations` создайте подкаталог с именем `xml`.
2. В каталоге `xml` создайте новый файл с именем `books.xml`. Он будет содержать структуру XML, которая должна будет читаться с использованием средств DOM из JavaScript. Добавьте в этот файл следующий код:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
  <books>
    <book>
      <title>
        Building Reponsive Web Applications with AJAX and PHP
      </title>
      <isbn>
        1-904811-82-5
      </isbn>
    </book>
    <book>
      <title>
        Beginning PHP 5 and MySQL E-Commerce: From Novice to Professional
      </title>
      <isbn>
        1-59059-392-8
      </isbn>
    </book>
  </books>
</response>
```

3. В том же самом каталоге создайте файл с именем `books.html` и добавьте в него следующий код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Основы AJAX: JavaScript и XML</title>
        <script type="text/javascript" src="books.js"></script>
    </head>
    <body onload="process()">
        Сервер, сообщи мне названия своих любимых книг!
        <br/>
        <div id="myDivElement" />
```

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

4. **И наконец, создайте файл** `books.js`**:**

```
// переменная для хранения ссылки на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// создать экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузера, за исключением
    // IE6 и более старых его версий
    try
    {
        попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array('MSXML2.XMLHTTP.6.0',
                                        'MSXML2.XMLHTTP.5.0',
                                        'MSXML2.XMLHTTP.4.0',
                                        'MSXML2.XMLHTTP.3.0',
                                        'MSXML2.XMLHTTP',
                                        'Microsoft.XMLHTTP');
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// вызывается для чтения файла с сервера
function process()
{
    // продолжать, только если в xmlHttp не пустая ссылка
```

```
    if (xmlHttp)
    {
        // попытаться установить соединение с сервером
        try
        {
            // инициировать чтение файла async.txt с сервера
            xmlHttp.open("GET", "books.xml", true);
            xmlHttp.onreadystatechange = handleRequestStateChange;
            xmlHttp.send(null);
        }
        // вывести сообщение об ошибке в случае неудачи
        catch (e)
        {
            alert("Невозможно соединиться с сервером:\n" + e.toString());
        }
    }
}

// эта функция вызывается при изменении состояния запроса HTTP
function handleRequestStateChange()
{
    // если readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен "OK"
        if (xmlHttp.status == 200)
        {
            try
            {
                // обработать ответ, полученный от сервера
                handleServerResponse();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                alert("Ошибка чтения ответа: " + e.toString());
            }
        }
        else
        {
            // вывести сообщение о состоянии
            alert("Возникли проблемы во время получения данных:\n" +
                  xmlHttp.statusText);
        }
    }
}

// обработать ответ, полученный от сервера
function handleServerResponse()
{
    // прочитать сообщение, полученное от сервера
    var xmlResponse = xmlHttp.responseXML;
```

```
        // получить ссылку на корневой элемент XML
        xmlRoot = xmlResponse.documentElement;
        // получить ссылки на массивы с названиями книг и их ISBN
        titleArray = xmlRoot.getElementsByTagName("title");
        isbnArray = xmlRoot.getElementsByTagName("isbn");
        // сгенерировать HTML-код
        var html = "";
        // обойти в цикле массивы и создать структуру HTML
        for (var i=0; i<titleArray.length; i++)
            html += titleArray.item(i).firstChild.data +
                    ", " + isbnArray.item(i).firstChild.data + "<br/>";
        // получить ссылку на элемент <div>
        myDiv = document.getElementById("myDivElement");
        // вывести полученный код HTML
        myDiv.innerHTML = "Сервер говорит: <br />" + html;
    }
```

5. Откройте в броузере страницу *http://localhost/ajax/foundations/xml/books.html* (рис.2.7).

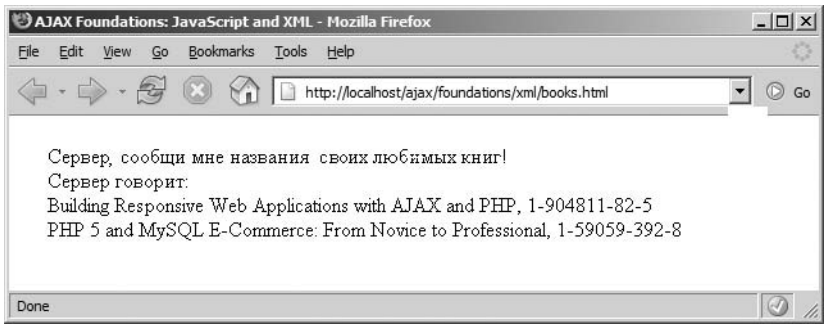

***Рис. 2.7.** Сервер знает, о чем говорит*

Что происходит внутри?

Большая часть кода уже знакома вам, поскольку в основном он повторяет то, что мы уже видели. Но в нем не было функции `handleServerResponse`, которая вызывается из `handleRequestStateChange` после получения ответа.

Начинается функция `handleServerResponse` с извлечения ответа сервера в формате XML:

```
// обработать ответ, полученный от сервера
function handleServerResponse()
{
    // прочитать сообщение, полученное от сервера
    var xmlResponse = xmlHttp.responseXML;
```

Метод `responseXML` объекта `XMLHttpRequest` возвращает ответ сервера в виде документа DOM. Если ответ сервера не является документом XML,

броузер может вывести сообщение об ошибке. Однако такое поведение зависит от реализации конкретного броузера, поскольку в каждом из них JavaScript и интерфейс DOM реализованы по-своему.

О проблеме устойчивости кода к ошибкам при чтении формата XML мы вскоре поговорим, а пока предположим, что наш документ XML не содержит ошибок, и посмотрим, как выполняется его чтение. Как известно, документ XML должен содержать хотя бы *корневой элемент*. В нашем случае таковым является элемент `<response>`. Как правило, прежде чем начать работу с документом, необходимо получить ссылку на его корневой элемент, как мы делаем это в нашем упражнении:

```
// получить ссылку на корневой элемент XML
xmlRoot = xmlResponse.documentElement;
```

На следующем шаге создаются два массива, в одном из которых будут храниться названия книг, а во втором – их ISBN. Делается это с помощью функции интерфейса DOM `getElementByTagName`, выполняющей синтаксический разбор всего документа XML и извлекающей из него элементы с заданными именами:

```
// получить ссылки на массивы с названиями книг и их ISBN
titleArray = xmlRoot.getElementsByTagName("title");
isbnArray = xmlRoot.getElementsByTagName("isbn");
```

Разумеется, возможности DOM далеко не исчерпывают способы чтения файлов XML. Более мощными возможностями обладает язык XPath, позволяющий создавать весьма замысловатые запросы для извлечения данных из документа XML.

Два массива, которые были нами получены, являются массивами элементов DOM. В данном случае текст, который необходимо отобразить, находится в первых дочерних элементах элементов `title` и `isbn` (первый дочерний элемент – это текстовый элемент, содержащий данные).

```
    // сгенерировать HTML-код
    var html = "";
    // обойти в цикле массивы и создать структуру HTML
    for (var i=0; i<titleArray.length; i++)
        html += titleArray.item(i).firstChild.data +
                ", " + isbnArray.item(i).firstChild.data + "<br/>";
    // получить ссылку на элемент <div>
    myDiv = document.getElementById("myDivElement");
    // вывести полученный код HTML
    myDiv.innerHTML = "Сервер говорит: <br />" + html;
}
```

Строки, выделенные жирным шрифтом, участвуют в создании структуры HTML, которая затем вставляется в элемент `<div>` на странице `books.html`.

Обработка ошибок и возбуждение исключений

Мы уже говорили, что, если документ, который вы пытаетесь прочитать, имеет некорректную структуру, каждый броузер может отреагировать на это по-своему. Мы провели небольшой эксперимент и удалили закрывающий тег `</response>` из файла `books.xml`. Броузер Firefox вывел сообщение об ошибке в консоли JavaScript, но кроме этого никаких других сообщений пользователю выведено не было. Конечно, это не очень хорошо, поскольку редко кто держит открытой консоль JavaScript при просмотре сайтов.

Открыть консоль JavaScript в Firefox (рис. 2.8) можно из меню `Tools→ JavaScript Console`. Дополнительную информацию о консоли JavaScript и других замечательных инструментах, которые помогут вам при отладке, см. в приложении B по адресу *http://ajaxphp.packtpub.com*.

***Рис. 2.8.** Очень удобная консоль JavaScript в Firefox*

Особенно неприятно, что все проверенные нами броузеры, за исключением Internet Explorer (всех версий), не смогли перехватить эту ошибку с помощью конструкции `try/catch`, которую мы поместили точно в том месте, где возникала ошибка. Подобно Firefox, Mozilla 1.7 также не породила никаких сообщений об ошибках, и, что еще хуже, она ничего не сообщила даже в консоли JavaScript. Она просто проигнорировала ошибку и повела себя так, как будто ничего не случилось и никаких данных она не получала, что наглядно продемонстрировано на рис. 2.9 (в Fireox была получена точно такая же страница).

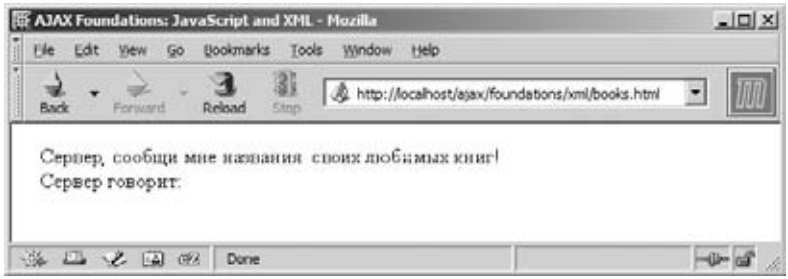

***Рис. 2.9.** Mozilla сокрыла факт появления ошибки*

Opera ведет себя более дружелюбно (по крайней мере, по отношению к разработчику). Она полностью проигнорировала конструкцию `try/catch`, которая должна была перехватить ошибку, но вывела весьма подробное ее описание. Хотя это очень удобно для разработчика, однако вы едва ли захотите, чтобы ваши пользователи увидели нечто подобное (рис. 2.10).

Рис. 2.10. *Opera выводит очень подробные сообщения об ошибках*

К моменту написания этих строк Internet Explorer оказался единственным броузером, в котором конструкция `try/catch` смогла перехватить исключение и вывести, пусть и достаточно бесполезное, сообщение об ошибке (рис. 2.11).

Рис. 2.11. *Исключение, перехваченное в Internet Explorer*

Так или иначе, но веб-броузеры не очень хорошо справляются с отлавливанием ошибок в сценариях, как следовало бы ожидать. Некоторые типы ошибок невозможно перехватить обычной конструкцией `try/catch`, поэтому особое значение приобретают альтернативные решения (потому что решение проблемы – это всегда хорошая новость). Код функции `handleServerResponse` можно сделать более устойчивым к ошибкам:

```
// обработать ответ, полученный от сервера
function handleServerResponse()
{
    // прочитать сообщение, полученное от сервера
```

```
    var xmlResponse = xmlHttp.responseXML;
    // перехватить потенциально возможные ошибки в IE и Opera
    if (!xmlResponse || !xmlResponse.documentElement)
        throw("Неверная структура документа XML:\n" + xmlHttp.responseText);
    // перехватить потенциально возможные ошибки в Firefox
    var rootNodeName = xmlResponse.documentElement.nodeName;
    if (rootNodeName == "parsererror")
        throw("Неверная структура документа XML:\n" + xmlHttp.responseText);
    // получить ссылку на корневой элемент XML
    xmlRoot = xmlResponse.documentElement;

// получить ссылки на массивы с названиями книг и их ISBN
        titleArray = xmlRoot.getElementsByTagName("title");
    isbnArray = xmlRoot.getElementsByTagName("isbn");
    // сгенерировать HTML код
    var html = "";
    // обойти в цикле массивы и создать структуру HTML
    for (var i=0; i<titleArray.length; i++)
        html += titleArray.item(i).firstChild.data +
                ", " + isbnArray.item(i).firstChild.data + "<br/>";
    // получить ссылку на элемент <div>
    myDiv = document.getElementById("myDivElement");
    // вывести полученный код HTML
    myDiv.innerHTML = "Server says: <br />" + html;
}
```

В Internet Explorer и Opera, если документ XML имеет некорректный формат, свойство `documentElement` объекта `xmlResponse` будет содержать значение `null`. В Firefox документ XML будет расценен как правильный, но сам документ будет замещен другим документом XML, содержащим описание ошибки (да, интересный способ сообщить об ошибке), и в этом случае корневым элементом документа будет элемент `parseerror`.

Когда мы обнаруживаем, что с полученным документом XML что-то не так, мы возбуждаем (`throw`) исключительную ситуацию. Операция возбуждения исключительной ситуации означает создание пользовательского исключения, и для этого в JavaScript предназначено ключевое слово `throw`. Это исключение будет перехвачено конструкцией `catch` в `handleServerResponse` и выведено передпользователем (рис. 2.12).

Я допускаю, что следующая часть кода может озадачить вас:

```
if (!xmlResponse || !xmlResponse.documentElement)
        throw("Неверная структура документа XML:\n" + xmlHttp.responseText);
```

Очевидно, что, если в `xmlResponse` содержится пустая ссылка, мы рискуем породить другую ошибку при попытке обратиться по ней к свойству `documentElement`. Но на практике интерпретатор JavaScript вычисляет логические выражения полностью только тогда, когда это необходимо, и делает это слева направо. В нашем конкретном случае, если выражение `(!xmlResponse)` дает значение `true`, второе выражение вообще не будет вычисляться, поскольку конечный результат в любом случае будет

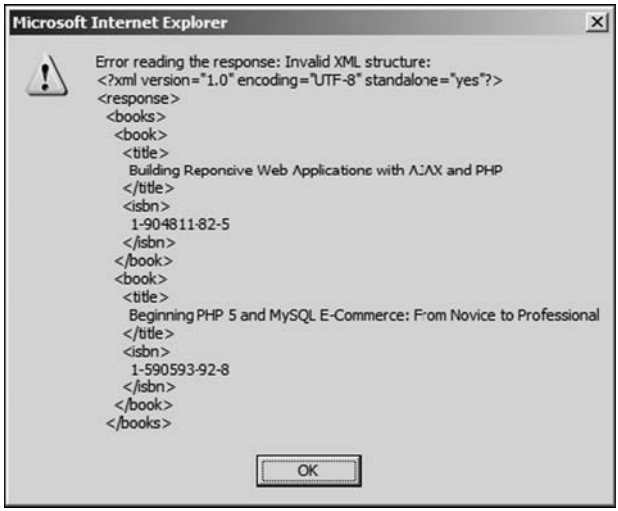

Рис. 2.12. *Сообщение об ошибке, которое было получено и выведено всеми протестированными броузерами*

`hfdty true`. Эта особенность имеется не только в JavaScript, но и в других языках программирования и называется **неполным вычислением логических выражений**, более подробно об этом можно прочитать по адресу *http://www.webreference.com/javascript/reference/core/expr.html*.

Создание структур XML

XML и DOM будут встречаться вам повсюду. В этой главе вы уже использовали интерфейс DOM для создания структуры элементов HTML в существующем объекте DOM – `document`, вы также узнали, как читать документы XML, принимаемые от сервера. Очень важный момент, которого мы пока еще не касались, – это создание новых документов XML средствами DOM в JavaScript. Это может вам потребоваться, если возникнет необходимость создать документ XML на стороне клиента и отправить его на сервер.

Мы не будем рассматривать много примеров и коснемся лишь недостающей части. Вся хитрость создания нового документа XML заключается в том, чтобы создать его. Когда вы добавляли новые элементы в код HTML, вы использовали неявный объект `document`, но это совершенно не годится для создания новых документов.

Когда в JavaScript средствами DOM создается новый документ, мы оказываемся перед лицом той же самой проблемы, как и в случае создания объекта `XMLHttpRequest`, – способ создания объекта зависит от типа броузера. Следующая функция не зависит от типа броузера и возвращает новый экземпляр объекта DOM:

```
function createDomObject()
{
    // переменная для хранения ссылки на объект DOM
    var xmlDoc;
    // создать документ XML
    if (document.implementation && document.implementation.createDocument)
    {
        xmlDoc = document.implementation.createDocument("", "", null);
    }
    // следующая ветка предназначена для Internet Explorer
    else if (window.ActiveXObject)
    {
        xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlDoc)
        alert("Ошибка создания объекта DOM.");
    else
        return xmlDoc;
}
```

После вызова этой функции можно использовать созданный объект DOM по своему усмотрению. Более подробно о создании объектов DOM рассказано здесь: *http://www.webreference.com/programming/javascript/domwrapper/index.html*. За информацией по применению объектов DOM обращайтесь к статьям, посвященным интерфейсу DOM, упоминавшимся ранее в этой главе.

Подведение итогов

В этой главе мы рассмотрели множество тем. Начать работать с HTML, JavaScript, CSS, DOM, XML и `XMLHttpRequest` не так-то просто, особенно если некоторые из этих технологий не знакомы вам. Везде, где вы чувствовали себя неуверенно, загляните по рекомендованным ссылкам на ресурсы в Интернете. Когда вы почувствуете себя достаточно уверенно, переходите к главе 3, где вы узнаете, как использовать PHP и MySQL на стороне сервера и как заставить их взаимодействовать с клиентами, поддерживающими технологию AJAX.

3

Технологии, применяемые на стороне сервера: PHP и MySQL

Если основной смысл технологии AJAX заключается в создании более интеллектуальных клиентов, то серверы, общающиеся с этими клиентами, должны обладать не меньшей интеллектуальностью, иначе они просто не смогут сотрудничать друг с другом.

Глава 2 была посвящена исключительно чтению с сервера статического текста или файлов XML. В этой главе мы переместимся на сторону сервера и начнем работать с PHP, который способен динамически генерировать выходные данные, и с MySQL, который будет хранить наши данные и управлять ими. Из этой главы вы узнаете:

- Как применять PHP для реализации функциональных возможностей сервера.
- Как организовать взаимодействие клиента и сервера и как передавать серверу дополнительные параметры запроса.
- Как использовать XML на стороне клиента и сервера.
- Как посредством сценариев PHP избежать потенциальных проблем с безопасностью в JavaScript.
- Как выполнять повторяющиеся действия на стороне клиента.
- Как работать с базами данных MySQL.
- Как можно оптимизировать архитектуру вашего приложения.

PHP и DOM

В главе 2 вы научились выполнять асинхронное чтение данных с сервера. Механизм чтения в значительной степени стандартизован, и те же самые функции вы будете применять на протяжении всей этой книги, однако эти данные, получаемые от сервера, имели важную отличительную особенность – они хранились в статических файлах (текстовых или в формате XML).

В большинстве реальных ситуаций на стороне сервера потребуется организовать промежуточную обработку информации и динамически генерировать выходные данные. В этой книге мы будем реализовывать функциональность сервера средствами PHP. Если ваши познания в PHP не отличаются особой глубиной, поиск в Интернете по фразе «php tutorial» даст вам огромное количество интереснейших ссылок, включая официальный учебник по PHP, размещенный по адресу *http://php.net/tut.php*.[1] Если вы предпочитаете обучаться на практических примерах, рекомендуем книгу Кристиана Дари (Cristian Darie) и Михая Бусики (Mihai Bucica) «Beginning PHP 5 and MySQL E-Commerce: From Novice to Professional», посвященную электронной торговле.

Вы можете даже привлечь приложения Suggest и Autocompletion, которые мы будем разбирать в главе 6, для поиска справочных материалов по функциям PHP. Эти приложения вы найдете по адресу *http://ajaxphp.packtpub.com/ajax/suggest/*.

В первом упражнении этой главы вы напишете сценарий на языке PHP, использующий функции интерфейса DOM для генерации выходных данных в формате XML, которые будут читаться клиентом. Функциональность DOM в PHP очень напоминает ту, что имеется в JavaScript, а официальную документацию по нему можно найти по адресу *http://www.php.net/manual/en/ref.dom.php*.

Содержимое документа, создаваемого сценарием, практически будет повторять содержимое статического файла из главы 2, но на сей раз оно будет генерироваться динамически:

```
<response>
  <books>
    <book>
      <title>Building Responsive Web Applications with AJAX and PHP</title>
      <isbn>1-904811-82-5</isbn>
    </book>
  </books>
</response>
```

Время действовать – взаимодействие AJAX и PHP

1. В каталоге `foundations` создайте подкаталог с именем `php`.
2. В каталоге `php` создайте файл с именем `phptest.html` и добавьте в него следующий код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
```

[1] Русскоязычную информацию по PHP можно получить на множестве ресурсов Интернета: *http://php.spb.ru/*, *http://phpclub.ru/*, *http://www.php5.ru/*. – *Примеч. науч. ред.*

```
    <head>
        <title>Практические примеры AJAX: Использование PHP DOM</title>
        <script type="text/javascript" src="phptest.js"></script>
    </head>
    <body onload="process()">
        Книга об AJAX, вышедшая в 2006 году:
        <br />
        <div id="myDivElement" />
    </body>
</html>
```

3. **Содержимое файла сценария** `phptest.js`**, исполняемого на стороне клиента, практически идентично содержимому файла** `books.js` **из упражнения, которое мы разбирали в главе 2. Немногочисленные изменения выделены жирным шрифтом:**

```
// переменная для хранения ссылки на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
```

```
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
           alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// вызывается для чтения файла с сервера
function process()
{
    // продолжать только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // попытаться установить соединение с сервером
        try
        {
            // инициировать чтение файла с сервера
            xmlHttp.open("GET", "phptest.php", true);
            xmlHttp.onreadystatechange = handleRequestStateChange;
            xmlHttp.send(null);
        }
        // вывести сообщение об ошибке в случае неудачи
        catch (e)
        {
            alert("Невозможно соединиться с сервером:\n" + e.toString());
        }
    }
}

// эта функция вызывается при изменении состояния запроса HTTP
function handleRequestStateChange()
{
    // когда readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if (xmlHttp.status == 200)
        {
            try
            {
                // обработать ответ, полученный от сервера
                handleServerResponse();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                alert("Ошибка чтения ответа: " + e.toString());
            }
        }
        else
        {
```

```
            // вывести сообщение о состоянии
            alert("Возникли проблемы во время получения данных:\n" +
                  xmlHttp.statusText);
        }
    }
}

// обрабатывает ответ, полученный от сервера
function handleServerResponse()
{
    // прочитать сообщение, полученное от сервера
    var xmlResponse = xmlHttp.responseXML;
    // предотвратить потенциально возможные ошибки в IE и Opera
    if (!xmlResponse || !xmlResponse.documentElement)
        throw("Неверная структура XML:\n" + xmlHttp.responseText);
    // предотвратить потенциально возможные ошибки в Firefox
    var rootNodeName = xmlResponse.documentElement.nodeName;
    if (rootNodeName == "parsererror") throw("Invalid XML structure");
    // получить ссылку на корневой элемент документа XML
    xmlRoot = xmlResponse.documentElement;
    // получить ссылки на массивы с названиями книг и их ISBN
    titleArray = xmlRoot.getElementsByTagName("title");
    isbnArray = xmlRoot.getElementsByTagName("isbn");
    // сгенерировать HTML код
    var html = "";
    // обойти в цикле массивы и создать структуру HTML
    for (var i=0; i<titleArray.length; i++)
        html += titleArray.item(i).firstChild.data +
                ", " + isbnArray.item(i).firstChild.data + "<br/>";
    // получить ссылку на элемент <div>
    myDiv = document.getElementById("myDivElement");
    // вывести полученный код HTML
    myDiv.innerHTML = html;
}
```

4. **И наконец, файл** `phptest.php`**:**

```
<?php
// определить формат выходных данных как xml
header('Content-Type: text/xml');
// создать новый документ XML
$dom = new DOMDocument();

// создать корневой элемент <response>
$response = $dom->createElement('response');
$dom->appendChild($response);

// создать элемент <books> и добавить его как дочерний,
// по отношению к <response>
$books = $dom->createElement('books');
$response->appendChild($books);

// создать элемент title для элемента book
```

```
$title = $dom->createElement('title');
$titleText = $dom->createTextNode
    ('Building Reponsive Web Applications with AJAX and PHP');
$title->appendChild($titleText);

// создать элемент isbn для элемента book
$isbn = $dom->createElement('isbn');
$isbnText = $dom->createTextNode('1-904811-82-5');
$isbn->appendChild($isbnText);

// создать элемент <book>
$book = $dom->createElement('book');
$book->appendChild($title);
$book->appendChild($isbn);

// добавить элемент <book>, как дочерний по отношению к элементу <books>
$books->appendChild($book);

// переписать структуру XML в строковую переменную
$xmlString = $dom->saveXML();
// вывести строку XML
echo $xmlString;
?>
```

5. Для начала посмотрим, что возвращает сценарий `phptest.php`. Откройте в броузере страницу *http://localhost/ajax/foundations/php/phptest.php* и убедитесь, что сценарий возвращает корректный (well-formed) документ XML (рис. 3.1).

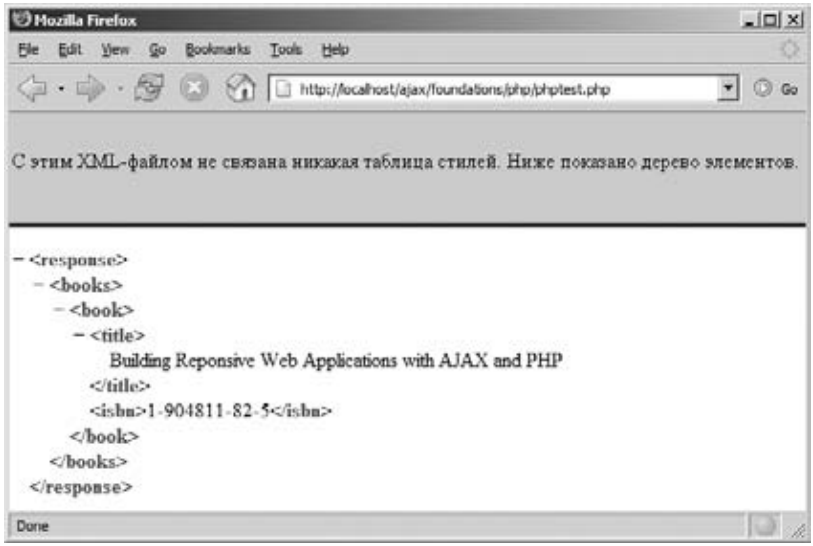

***Рис. 3.1.** Простой документ XML, сгенерированный PHP*

Примечание

Если вы получили иной результат, то проверьте не только код сценариев, но и правильность установки PHP. О том, как настроить рабочее окружение, рассказано в приложении А.

6. Убедившись, что сервер возвращает корректный ответ, протестируйте работу приложения в целом, открыв страницу *http://localhost/ajax/foundations/php/phptest.html* (рис. 3.2).

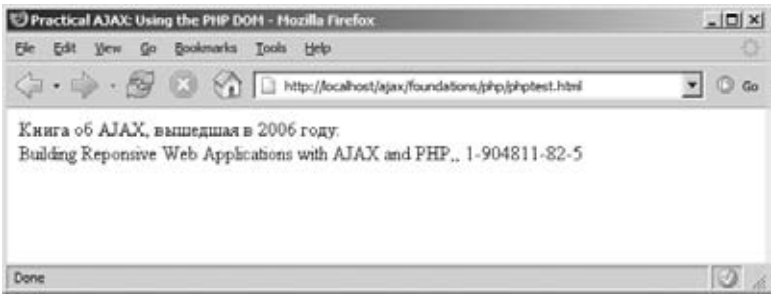

***Рис. 3.2.** AJAX и PHP*

Что происходит внутри?

Если вы собираетесь создавать документы XML не только на стороне клиента, но и на стороне сервера, то должны выбрать способ, которым это будет делаться, – средствами DOM или с помощью операций конкатенации строк. Ваш сценарий PHP, `phptest.php`, начинается с настройки типа выходных данных:

```
<?php
// определить формат выходных данных как xml
header('Content-Type: text/xml');
```

Описание функции `header` языка PHP можно найти по адресу *http://www.php.net/manual/en/function.header.php* (не забывайте, что к поиску описания функций можно привлечь приложение Suggest, которое отправит вас прямо на нужную страницу).

Примечание

В JavaScript мы заключали строковые значения в двойные кавычки, а в PHP всегда будем употреблять одиночные. Они обрабатываются интерпретатором языка гораздо быстрее, более безопасны и к тому же снижают вероятность появления ошибок программирования. Работа со строками в PHP рассмотрена по адресу *http://php.net/types.string*. Там же можно прочитать еще две статьи на ту же тему: *http://www.sitepoint.com/print/quick-php-tips* и *http://www.jeroenmulder.com/weblog/2005/php_single_and_double_quotes.php*.

Интерфейс DOM в PHP во многом напоминает DOM в JavaScript и не содержит ничего для вас необычного. Создание документа XML начинается с создания объекта DOM, который в PHP представлен классом `DOMDocument`:

```
// создать новый документ XML
$dom = new DOMDocument();
```

Создание документа продолжается с помощью таких методов, как `createElement`, `createTextNode`, `appendChild` и им подобных:

```
// создать корневой элемент <response>
$response = $dom->createElement('response');
$dom->appendChild($response);

// создать элемент <books> и добавить его как дочерний,
// по отношению к <response>
$books = $dom->createElement('books');
$response->appendChild($books);
...
```

В заключение мы сохраняем созданный документ XML в виде строки с помощью функции `saveXML` и посредством оператора `echo` выводим ее:

```
// переписать структуру XML в строковую переменную
$xmlString = $dom->saveXML();
// вывести строку XML
echo $xmlString;
?>
```

После этого документ будет прочитан клиентом и отображен на экране, с использованием технологий, которые мы уже обсуждали в главе 2.

Примечание

Чаще всего вам придется создавать документы на стороне сервера и отправлять их клиенту, но, само собой разумеется, документы XML можно передавать и в обратном направлении. В главе 2 вы видели, как можно создавать документы XML с помощью DOM JavaScript. Созданный документ можно отправить сценарию PHP (посредством метода GET или POST – см. следующее упражнение). Прочитать документ XML из PHP также можно средствами интерфейса DOM или простейшего прикладного программного интерфейса, называемого **SimpleXML**. В работе с SimpleXML вы попрактикуетесь в главе 9, когда будете создавать приложение для чтения лент новостей (RSS Reader).

Передача параметров и обработка ошибок в PHP

В предыдущем упражнении были проигнорированы два наиболее распространенных аспекта, связанных с разработкой сценариев PHP:

- Обычно клиенту необходимо передавать какие-либо параметры сценарию, который работает на стороне сервера.

- Теперь, когда клиентская сторона достаточно хорошо защищена от появления ошибок, необходимо предусмотреть обработку ошибок и на стороне сервера.

Передавать параметры сценарию PHP можно методом `GET` или `POST`. Обработка ошибок в PHP производится при помощи характерных для него методик. В следующем упражнении вы передадите параметры сценарию PHP и реализуете механизм обработки ошибочных ситуаций, который можно будет проверить, подставляя заведомо ошибочные значения в запрос. Приложение будет выглядеть так, как показано на рис. 3.3.

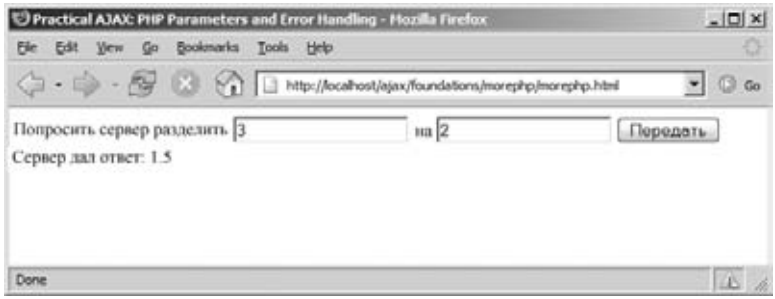

Рис. 3.3. *Передача параметров и обработка ошибок в PHP*

Эта страница асинхронно отправляет серверу два числа и ожидает получить от него частное. Сервер, если не будет ошибок, возвратит результат в виде документа XML:

```
<?xml version="1.0"?>
<response>1.5</response>
```

В случае появления ошибок вместо документа XML сервер возвратит обычный текст, содержащий сообщение об ошибке, которое будет перехвачено и опознано клиентом (после того как мы закончим это упражнение, вы поймете почему).

Время действовать – передача параметров и обработка ошибок в PHP

1. В каталоге `foundations` создайте подкаталог `morephp`.
2. В каталоге `morephp` создайте файл с именем `morephp.html`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>
      Практические примеры AJAX: Параметры и обработка ошибок в PHP
    </title>
    <script type="text/javascript" src="morephp.js"></script>
  </head>
```

```
  <body>
    Попросить сервер разделить
    <input type="text" id="firstNumber" />
    на
    <input type="text" id="secondNumber" />
    <input type="button" value="Передать" onclick="process()" />
    <div id="myDivElement" />
  </body>
</html>
```

3. **Создайте новый файл с именем** `morephp.js`**:**

```
// переменная для хранения ссылки на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
      alert("Ошибка создания объекта XMLHttpRequest.");
   else
```

```
        return xmlHttp;
}

// вызывается для чтения файла с сервера
function process()
{
    // продолжать только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // попытаться установить соединение с сервером
        try
        {
            // получить два значения, введенные пользователем
            var firstNumber = document.getElementById("firstNumber").value;
            var secondNumber = document.getElementById("secondNumber").value;
            // создать строку с параметрами
            var params = "firstNumber=" + firstNumber +
                         "&secondNumber=" + secondNumber;
            // инициировать асинхронный запрос HTTP
            xmlHttp.open("GET", "morephp.php?" + params, true);
            xmlHttp.onreadystatechange = handleRequestStateChange;
            xmlHttp.send(null);
        }
        // вывести сообщение об ошибке в случае неудачи
        catch (e)
        {
            alert("Невозможно соединиться с сервером:\n" + e.toString());
        }
    }
}

// эта функция вызывается при изменении состояния запроса HTTP
function handleRequestStateChange()
{
    // когда readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if (xmlHttp.status == 200)
        {
            try
            {
                // обработать ответ, полученный от сервера
                handleServerResponse();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                alert("Ошибка чтения ответа: " + e.toString());
            }
         }
         else
```

```
        {
            // вывести сообщение о состоянии
            alert("Возникли проблемы во время получения данных:\n" +
                  xmlHttp.statusText);
        }
    }
}

// обрабатывает ответ, полученный от сервера
function handleServerResponse()
{
    // получить ответ сервера в виде объекта DOM XML
    var xmlResponse = xmlHttp.responseXML;
    // предотвратить потенциально возможные ошибки в IE и Opera
    if (!xmlResponse || !xmlResponse.documentElement)
        throw("Invalid XML structure:\n" + xmlHttp.responseText);
    // предотвратить потенциально возможные ошибки в Firefox
    var rootNodeName = xmlResponse.documentElement.nodeName;
    if (rootNodeName == "parsererror")
        throw("Invalid XML structure:\n" + xmlHttp.responseText);
    // получить ссылку на корневой элемент документа XML
    xmlRoot = xmlResponse.documentElement;
    // поверить корректность принятого документа XML
    if (rootNodeName != "response" || !xmlRoot.firstChild)
        throw("Неверный формат документа XML:\n" + xmlHttp.responseText);
    // значение, которое требуется отобразить, находится
    // в дочернем элементе корневого элемента <response>
    responseText = xmlRoot.firstChild.data;
    // отобразить результат перед пользователем
    myDiv = document.getElementById("myDivElement");
    myDiv.innerHTML = "Сервер дал ответ: " + responseText;
}
```

4. **Создайте файл с именем** `morephp.php`**:**

```
<?php
// загрузить модуль обработки ошибок
require_once('error_handler.php');
// указать, что выходные данные будут иметь формат XML
header('Content-Type: text/xml');
// вычислить частное
$firstNumber = $_GET['firstNumber'];
$secondNumber = $_GET['secondNumber'];
$result = $firstNumber / $secondNumber;
// создать новый документ XML
$dom = new DOMDocument();
// создать корневой элемент <response> и добавить его в документ
$response = $dom->createElement('response');
$dom->appendChild($response);
// добавить частное в виде дочернего текстового узла в элемент <response>
$responseText = $dom->createTextNode($result);
$response->appendChild($responseText);
// записать документ XML в строковую переменную
```

```
$xmlString = $dom->saveXML();
// вывести строку
echo $xmlString;
?>
```

5. **И наконец, создайте файл** `error_handler.php` **со сценарием обработки ошибок:**

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

6. **Откройте в броузере страницу** *http://localhost/ajax/foundations/morephp/morephp.html* **и поэкспериментируйте с ней.**

Что происходит внутри?

Большая часть кода, работающая на стороне клиента, вам должна быть уже знакома, поэтому сфокусируем внимание на серверной части, где у нас есть два файла: `morephp.php` **и** `error_handler.php`**.**

Предполагается, что сценарий `morephp.php` **выведет документ XML, содержащий результат деления двух чисел. Однако он начинается с процедуры загрузки функции обработки ошибок. Эта функция должна перехватить любую ошибку, возникающую в сценарии, создать сообщение с более подробным ее описанием, чем это делает обработчик по умолчанию, и передать сообщение клиенту.**

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
```

Примечание

PHP 5 поддерживает механизм исключений, как и другие объектно-ориентированные языки программирования. Однако в PHP 5 можно использовать лишь свои собственные объекты исключений, которые генерируются командой `throw`, и перехватывать их конструкцией `catch`, тем не менее они могут оказать существенную помощь в создании приложений со сложной архитектурой и улучшить

код. Само ядро PHP в случае появления ошибок не генерирует исключений. Вероятно, это связано с необходимостью сохранения обратной совместимости – когда возникает ошибочная ситуация, PHP 5 не возбуждает исключения, а генерирует **ошибки**, которые представляют собой намного более примитивный способ обработки проблемных ситуаций во время исполнения. Например, нельзя перехватить ошибку, обработать ее локально и продолжить исполнение сценария обычным образом, как это допускают исключения. Вместо этого лучше всего определить функцию, которая будет запускаться автоматически. Она будет вызвана перед тем, как сценарий аварийно завершит свою работу, и даст вам последний шанс выполнить некоторые заключительные операции (записать сообщение об ошибке в файл журнала, закрыть соединение с базой данных или сообщить вашему посетителю что-нибудь «дружественное»).

В нашем коде сценарий `error_handler.php` предназначен для обработки ошибок. Он просто принимает ошибку и трансформирует сообщение о ней в нечто более понятное, чем сообщение по умолчанию. Обратите внимание: `error_handler.php` перехватывает большинство ошибок, но далеко не все! Фатальные ошибки не могут быть перехвачены кодом PHP, и они генерируют выходные данные, которые вы не сможете контролировать. Скажем, ошибки синтаксического анализа, которые возникают, если забыть вставить символ `$` перед именем переменной, прерывают работу интерпретатора еще до того, как начнется исполнение сценария, по этой причине они не могут быть перехвачены кодом PHP, но сообщения о них записываются в журнал веб-сервера Apache.

Примечание

Очень важно следить за содержимым журнала Apache, когда ваши сценарии PHP ведут себя странным образом. По умолчанию журнал сохраняется в файле `Apache2\logs\error.log`, и он поможет вам избавиться от множества неприятностей.

Установив функцию-обработчик, мы определяем формат выходных данных как XML и делим первое принятое число на второе. Обратите внимание, что значения, переданные методом `GET`, мы извлекаем из массива `$_GET`. Если параметры передаются методом `POST`, то извлекать их надо из массива `$_POST`. Еще можно использовать массив `$_REQUEST`, в который помещаются параметры, переданные любым способом (включая `cookies`), но лучше этого не делать, поскольку скорость работы при этом заметно снижается.

```
// указать, что выходные данные будут иметь формат XML
header('Content-Type: text/xml');
// вычислить частное
$firstNumber = $_GET['firstNumber'];
$secondNumber = $_GET['secondNumber'];
$result = $firstNumber / $secondNumber;
```

Операция деления сгенерирует ошибку, если переменная `$secondNumber` содержит значение `0`. В этом случае мы предполагаем, что ошибка будет перехвачена сценарием обработки ошибок. Обратите внимание, что в реальной ситуации более профессионально было бы проверить значение переменной перед выполнением операции деления, но сейчас нас интересует именно поведение сценария обработки ошибок.[1]

Вычисленный результат вставляется в документ XML и отправляется клиенту, точно так же, как и в предыдущем упражнении:

```
// создать новый документ XML
$dom = new DOMDocument();
// создать корневой элемент <response> и добавить его в документ
$response = $dom->createElement('response');
$dom->appendChild($response);
// добавить частное в виде дочернего текстового узла в элемент <response>
$responseText = $dom->createTextNode($result);
$response->appendChild($responseText);
// записать документ XML в строковую переменную
$xmlString = $dom->saveXML();
// вывести строку
echo $xmlString;
?>
```

А теперь заглянем в сценарий обработки ошибок — `error_handler.php`. Этот файл играет роль ловушки для любых сообщений об ошибках, сгенерированных PHP, и выводит более осмысленное сообщение, представленное на рис. 3.4 (оно может быть выведено кодом JavaScript):

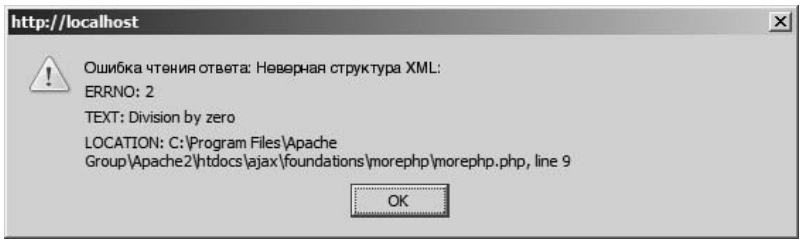

***Рис. 3.4.** Удобочитаемое сообщение об ошибке*

Если бы мы не предусматривали обработку ошибок, сообщение выглядело бы так, как на рис. 3.5.

---

[1] Это извечный вопрос программирования на любом уровне: проверять предварительно значения данных или перехватывать событие уже возникшей ошибки. Если бы во всех случаях можно было предусмотреть все возможные недопустимые комбинации данных, то не потребовались бы большие усилия на разработку средств реакции на исключительные ситуации практически во всех средствах программирования. В данном случае простая проверка на 0 является возможным решением, но только в силу исключительной простоты ситуации в этом примере. — *Примеч. науч. ред.*

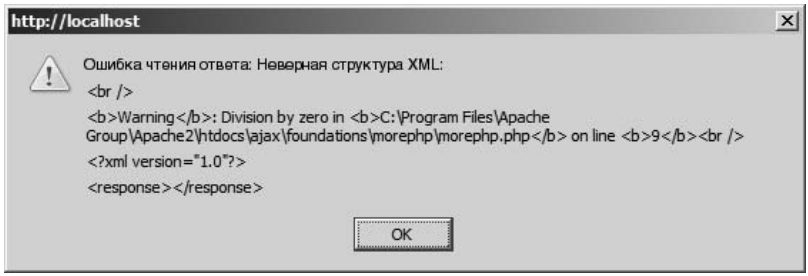

***Рис. 3.5.** Неудобочитаемое сообщение об ошибке*

Примечание

Сообщение об ошибке будет выглядеть так, как показано на рис. 3.5, только если в файле настроек `php.ini` параметр `display_errors` равен `On`. По умолчанию он имеет значение `Off`, и сообщение об ошибке просто записывается в журнал Apache, но, написав специальный код, вы тоже сможете вывести это сообщение. Однако в окончательной версии приложения ни тот, ни другой вариант неприемлемы. Никогда не надо показывать своим пользователям подобные отладочные сообщения.

Итак, что же происходит в недрах сценария `error_handler.php`? Прежде всего сценарий назначает новый обработчик ошибок с помощью функции `set_error_handler`:

```
<?php
// установить функцию error_handler как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
```

Когда возникает ошибка, мы в первую очередь вызываем функцию `ob_clean()`, чтобы удалить из выходного буфера все, что туда успело попасть, например текст `<response></response>` (рис. 3.5).

```
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
```

Разумеется, если вы предпочитаете сохранять эту информацию для нужд отладки, то можете закомментировать вызов `ob_clean()`. Фактическое сообщение об ошибке собирается из содержимого переменных `$errNo`, `$errStr`, `$errFile` и `$errLine` и дополнительных символов перевода строки, которые вставляются с помощью функции `chr`.

```
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
```

```
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

Примечание

Схема обработки ошибок, представленная нами, чрезвычайно упрощена и хороша только на начальных этапах разработки и отладки кода. В окончательной версии приложения надо выводить перед пользователем более дружественные сообщения, не содержащие каких-либо технических подробностей. При желании оформлять сообщения об ошибках как документы XML, которые будут передаваться клиенту, имейте в виду, что ошибки синтаксического анализа и фатальные ошибки не смогут обрабатываться вашей функцией и будут вести себя так, как это определено в файле конфигурации PHP (`php.ini`).

Кроме того, пользователь может попытаться выполнить несколько запросов одновременно (вы можете сделать это, быстро, несколько раз подряд щелкнув по кнопке Отправить). Если вы попытаетесь отправить запрос в то время, когда объект `XMLHttpRequest` еще занят обслуживанием предыдущего запроса, его метод `open` сгенерирует исключение. Наш код хорошо защищен от ошибок с помощью конструкций `try/catch`, но само сообщение об ошибке выглядит не слишком дружественно (рис. 3.6).

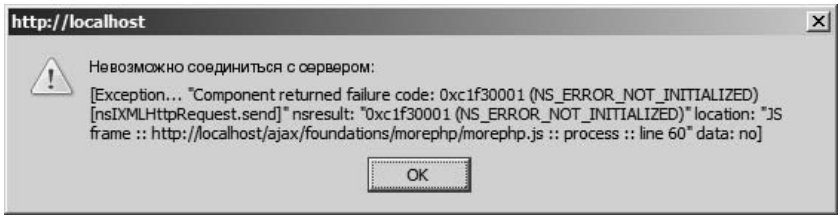

Рис. 3.6. *Попытка выполнения запроса в то время, когда XMLHttpRequest занят обработкой предыдущего запроса*

Может быть, это сообщение и будет вам полезно, но вы наверняка предпочтете иметь возможность по-разному реагировать на разные типы ошибок. Скажем, вывести замечание на странице или более дружественное сообщение: «Пожалуйста, повторите попытку чуть позже», изменив функцию `process()` так, как показано в следующем фрагменте кода:

```
// вызывается для чтения файла с сервера
function process()
{
    // продолжать только если в xmlHttp не пустая ссылка
    if (!xmlHttp) return;
    // не пытаться выполнить запрос к серверу,
    // если XMLHttpObject занят обработкой предыдущего запроса
    if !(xmlHttp.readyState == 0 || xmlHttp.readyState == 4)
     alert("Невозможно соединиться с сервером, повторите попытку чуть позже.");
    else
```

```
    {
        // попытаться установить соединение с сервером
        try
        {
            // получить два значения, введенные пользователем
            var firstNumber = document.getElementById("firstNumber").value;
           var secondNumber = document.getElementById("secondNumber").value;
            // создать строку с параметрами
            var params = "firstNumber=" + firstNumber +
                         "&secondNumber=" + secondNumber;
            // инициировать асинхронный запрос HTTP
            xmlHttp.open("GET", "morephp.php?" + params, true);
            xmlHttp.onreadystatechange = handleRequestStateChange;
            xmlHttp.send(null);
        }
        // вывести сообщение об ошибке в случае неудачи
        catch (e)
        {
            alert("Невозможно соединиться с сервером:\n" + e.toString());
        }
    }
}
```

Но в любом случае конкретный способ обработки этих ошибок может существенно варьировать, в зависимости от необходимости. В этой книге вам встретятся и другие решения:

- Иногда мы будем просто игнорировать ошибки.
- Иногда – выводить свое собственное сообщение, как это было показано выше.

В большинстве случаев следует попытаться вообще избежать появления ошибок – проблему всегда проще предотвратить. Например, известно несколько способов избежать появления ошибки, связанной с занятостью объекта `XMLHttpRequest`, когда вы пытаетесь выполнить запрос, в то время как объект еще занят обработкой предыдущего запроса:

- Для каждого запроса можно открывать новое соединение (создать новый объект `XMLHttpRequest`) с сервером. Этот метод достаточно прост в реализации и может с успехом применяться в большинстве случаев, но вообще старайтесь избегать его, поскольку он может отрицательно сказаться на производительности сервера (ваш сценарий будет продолжать посылать новые запросы, несмотря на то что сервер еще не успел ответить на предыдущие), а кроме того, эта методика не гарантирует, что вы получите ответы в том же порядке, в каком были посланы запросы (особенно при большой нагрузке на сервер или в медленных сетях).
- Запросы можно ставить в *очередь* и отправлять их по одному по мере возможности (этот метод реализован в нескольких упражнениях

из этой книги, включая упражнение проверки правильности заполнения формы, и в приложении Chat).

- Наконец, можно игнорировать новые запросы, чтобы код не пытался выполнять несколько запросов через то же самое соединение, и использовать существующий код обработки ошибок.

Соединение с удаленным сервером и безопасность сценариев JavaScript

Вы можете удивиться, обнаружив вдруг, что упражнение, которое мы только что закончили рассматривать, работает без проблем только потому, что серверный сценарий (PHP), вызываемый асинхронно, исполняется на том же самом сервере, откуда была получена исходная страница HTML.

Веб-броузеры очень строго (и по-разному) ограничивают возможность доступа к сетевым ресурсам из кода JavaScript. Если вы обратитесь из своего кода JavaScript к другому серверу, то последнему безопаснее заявить, что это недопустимо. Этому и посвящено следующее упражнение, но сначала немного теории.

Итак, код JavaScript работает с привилегиями родительского файла HTML. По умолчанию, когда вы открываете страницу HTML на сервере, код JavaScript сопутствующий этой странице, сможет обращаться только к этому серверу. Любой другой сервер будет расцениваться как потенциальный злоумышленник, и эта ситуация обрабатывается в разных броузерах по-разному (к сожалению).

Internet Explorer относится к категории дружественных броузеров, это означает, что он может предоставить более широкую функциональность, но за счет снижения уровня безопасности. В нем реализована модель безопасности, основанная на **зонах**. Всего имеется четыре зоны: «Интернет», «Местная интрасеть», «Надежные узлы» и «Ограниченные узлы». Для каждой зоны предусмотрены различные настройки уровня безопасности, которые вы можете изменить, вызвав диалог `Сервис | Свойства обозревателя | Безопасность`. Когда осуществляется доступ к узлу сети, автоматически выбирается соответствующая ему зона, и вступают в силу заданные для нее параметры безопасности.

Параметры безопасности каждой из зон могут сильно отличаться, в зависимости от системы. По умолчанию Internet Explorer предоставляет неограниченные права сценариям, загруженным из локальных файлов (т. е. не с веб-сервера и даже не с локального веб-сервера). Так, если вы попытаетесь загрузить файл `c:\ajax\...`, сценарий отработает без каких-либо проблем (правда, перед его запуском броузер может предупредить вас о том, что сценарий будет исполнен с неограниченными правами). Если код JavaScript был загружен через HTTP (скажем, с локального ресурса *http://localhost/ajax/.../ping.html*) и попы-

тается отправить запрос HTTP другому серверу, Internet Explorer автоматически выведет диалог подтверждения, где пользователь должен будет подтвердить право на выполнение действия.

Firefox и броузеры, разработанные на базе Mozilla, реализуют более строгую и более сложную модель безопасности, основанную на **привилегиях**. Эти броузеры не выводят диалог подтверждения, а код JavaScript должен сам запросить возможность выполнения требуемых действий, обращаясь для этого к средствам API, специфичного для Mozilla. Если таковая возможность имеется, броузер выведет перед пользователем диалог подтверждения и в зависимости от его решения, даст (или не даст) необходимое разрешение коду JavaScript. В противном случае броузер будет полностью игнорировать запросы, посылаемые сценарием. По умолчанию броузеры, основанные на Mozilla, пропускают запросы, идущие от сценариев, загруженных из локального ресурса (`file:///`), и полностью игнорируют запросы, идущие от сценариев, полученных по HTTP, если эти сценарии не имеют цифровой подписи (хотя эти настройки по умолчанию могут быть изменены пользователем). Подробнее о сценариях с цифровой подписью см. по адресу *http://www.mozilla.org/projects/security/components/signed-scripts.html*.

В следующем упражнении вы создадите программу на JavaScript, которая будет считывать последовательность случайных чисел с ресурса *http://www.random.org*. Этот сайт предоставляет онлайновую службу, которая генерирует *по-настоящему случайные числа*. Страница, описывающая порядок доступа к серверу через HTTP, находится по адресу *http://www.random.org/http.html*. При разработке программ, которые будут обращаться к этой службе, необходимо принять во внимание рекомендации, опубликованные по адресу *http://www.random.org/guidelines.html*. И в заключение, чтобы увидеть, на что похожи случайные числа, откройте в броузере страницу *http://www.random.org/cgi-bin/randnum* (при открытии страницы без параметров по умолчанию перед вами будет выведено 100 случайных чисел, в диапазоне от 1 до 100). Наш клиент будет запрашивать по одному случайному числу за раз, из диапазона от 1 до 100, отправляя запрос по адресу *http://www.random.org/cgi-bin/randnum?num=1&min=1&max=100*.

Время действовать – соединение с удаленным сервером

1. Создайте новый каталог с именем `ping` в каталоге `foundations`.
2. В каталоге `ping` создайте новый файл с именем `ping.html` и добавьте в него следующий код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>
            Практические примеры AJAX: Соединение с удаленным сервером
```

```
        </title>
        <script type="text/javascript" src="ping.js"></script>
    </head>
    <body onload="process()">
        Сервер, верни мне случайное число!<br/>
        <div id="myDivElement" />
    </body>
</html>
```

3. Создайте новый файл с именем `ping.js` и добавьте в него следующий код, результат работы которого представлен на рис. 3.7:

```
// переменная для хранения ссылки на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// переменные для хранения адреса удаленного сервера
// и параметров запроса
var serverAddress = "http://www.random.org/cgi-bin/randnum";
var serverParams = "num=1" +   // количество запрашиваемых случайных чисел
                   "&min=1" +  // минимальное случайное число
                   "&max=100"; // максимальное случайное число
// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
```

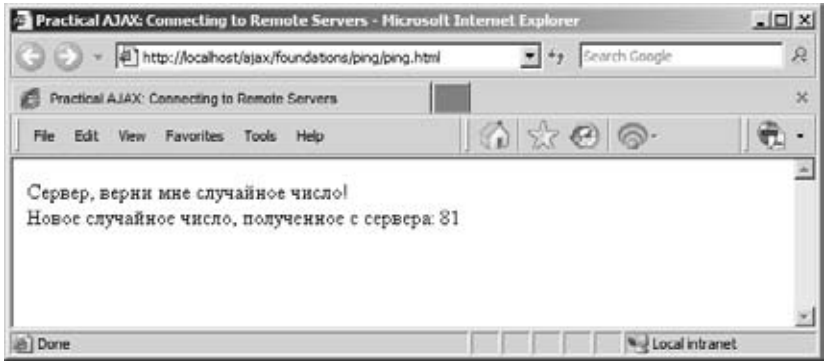

Рис. 3.7. *Соединение с удаленным сервером*

```
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// производит асинхронное обращение к серверу
function process()
{
    // продолжать только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // попытаться установить соединение с сервером
        try
        {
        // запросить права доступа к удаленному серверу
        // в броузерах, основанных на коде Mozilla
            try
            {
                // этот код сгенерирует ошибку (которую мы проигнорируем)
                // если броузер не принадлежит к семейству Mozilla
                netscape.security.PrivilegeManager.enablePrivilege(
                    'UniversalBrowserRead');
            }
            catch(e) {} // игнорировать ошибку
            // инициировать доступ к серверу
           xmlHttp.open("GET", serverAddress + "?" + serverParams, true);
            xmlHttp.onreadystatechange = handleRequestStateChange;
            xmlHttp.send(null);
        }
        // вывести сообщение об ошибке в случае неудачи
        catch (e)
        {
```

```
            alert("Невозможно соединиться с сервером:\n" + e.toString());
        }
    }
}

// эта функция вызывается при изменении состояния запроса HTTP
function handleRequestStateChange()
{
    // когда readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if (xmlHttp.status == 200)
        {
            try
            {
                // обработать ответ, полученный от сервера
                handleServerResponse();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                alert("Ошибка чтения ответа: " + e.toString());
            }
        }
        else
        {
            // вывести сообщение о состоянии
            alert("Возникли проблемы во время получения данных:\n" +
                  xmlHttp.statusText);
        }
    }
}

// обрабатывает ответ, полученный от сервера
function handleServerResponse()
{
    // прочитать сообщение, полученное от сервера
    var response = xmlHttp.responseText;
    // получить ссылку на элемент <div>
    myDiv = document.getElementById('myDivElement');
    // вывести полученный код HTML
    myDiv.innerHTML = "Новое случайное число, полученное с сервера: "
                      + response + "<br/>";
}
```

4. Откройте страницу *http://localhost/ajax/foundations/ping/ping.html*. Internet Explorer с настройками по умолчанию выведет диалоговое окно, где пользователь должен разрешить сценарию соединение с удаленным сервером (рис. 3.8). Firefox и Opera с настройками по

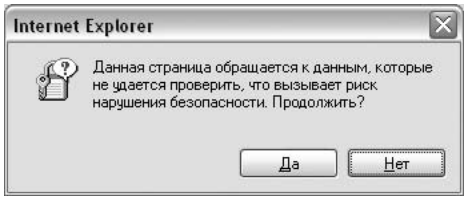

Рис. 3.8. *Internet Explorer запрашивает разрешение на соединение с удаленным сервером*

умолчанию выведут сообщения об ошибках, представленные на рис. 3.9 и 3.10 соответственно.

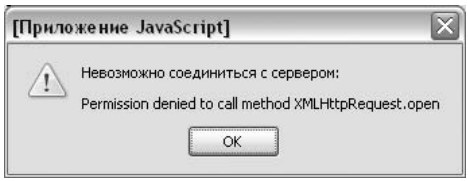

Рис. 3.9. *Firefox закрывает доступ*

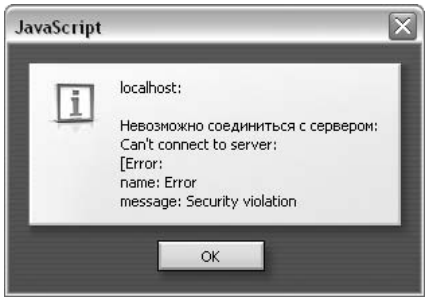

Рис. 3.10. *Opera закрывает доступ*

5. А теперь попробуйте открыть тот же самый файл HTML, но на этот раз прямо из файловой системы. Путь к файлу должен быть похожим на *file:///c:/Apache2/htdocs/ajax/foundations/ping/ping.html*. С настройками по умолчанию Internet Explorer исполнит сценарий без проблем, потому что страница находится в зоне надежных узлов. Firefox запросит подтверждение (рис. 3.11). Opera выведет то же самое сообщение об ошибке, которое мы уже видели на рис. 3.10.

Что происходит внутри?

Opera – это действительно самый безопасный броузер в мире. Нет никакой возможности «уговорить» Opera 8.5 позволить коду JavaScript обратиться к серверу, отличному от того, с которого был получен этот код.

Internet Explorer ведет себя в соответствии с настройками зон безопасности. По умолчанию он оказывает максимальное доверие локальным

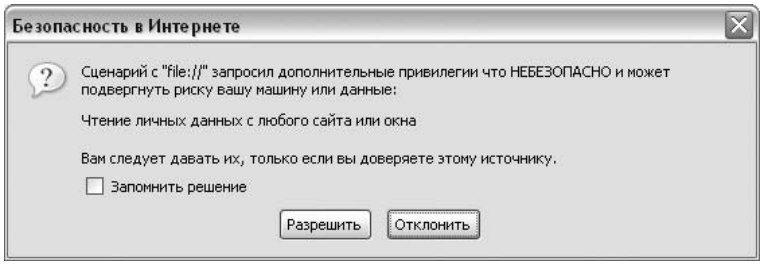

Рис. 3.11. *Firefox запрашивает разрешение на доступ к удаленному серверу*

файлам и запрашивает разрешение, когда сценарий, полученный из Интернета, пытается выполнить потенциально опасную операцию.

Работая с Firefox, надо вежливо спросить у него разрешение, если хотите, чтобы все прошло гладко. Проблема, однако, состоит в том, что он даже слушать вас не будет, если сценарий не подписан или загружен из локального файла (`file://`). Тем не менее в большинстве случаев вы вполне можете попросить пользователя изменить настройки броузера.

Можно заставить Firefox «внимательно прислушиваться» ко всем просьбам, даже к тем, которые исходят от неподписанных сценариев. Для этого надо ввести `about:config` в адресной строке и изменить значение параметра `signed.applets.codebase_principal_support` на `true`.

Ниже приводится отрывок кода, который запрашивает у Firefox разрешение на доступ к удаленному серверу:

```
// запросить права доступа к удаленному серверу
// в броузерах, основанных на коде Mozilla
try
{
    // этот код сгенерирует ошибку (которую мы проигнорируем),
    // если броузер не принадлежит к семейству Mozilla
    netscape.security.PrivilegeManager.enablePrivilege(
         'UniversalBrowserRead');
}
catch(e) {} // игнорировать ошибку
```

Любые ошибки, возникающие при исполнении этого кода, игнорируются, для чего применяется конструкция `try/catch`, т. к. этот код имеет смысл только в броузерах из семейства Mozilla и в других броузерах генерирует ошибку.

Доверенный сценарий на стороне сервера

Совершенно очевидно, что, если только вы не создаете какое-либо решение, которое позволит вам управлять окружением, например, заставить всех ваших пользователей работать с Internet Explorer или Firefox (в этом случае придется подписывать сценарии или вручную

переконфигурировать броузеры), доступ к удаленным серверам из JavaScript не рассматривается.

Затруднение ликвидируется достаточно просто – надо не обращаться к удаленному серверу напрямую из кода JavaScript, а написать сценарий PHP, который будет работать на вашем сервере и обращаться к удаленному серверу от имени клиента. Эта методика графически представлена на рис. 3.12.

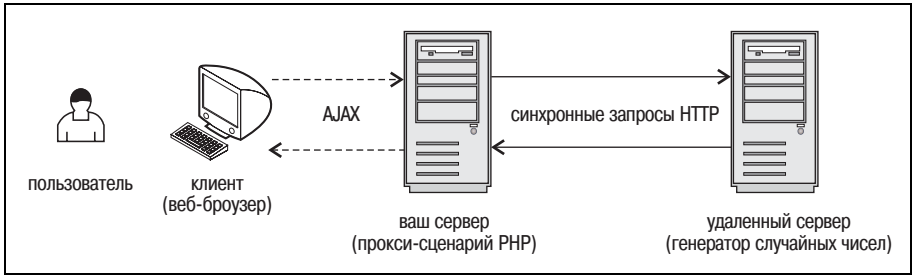

Рис. 3.12. *Использование доверенного сценария PHP для доступа к удаленному серверу*

Для чтения данных с удаленного сервера в сценарии PHP применяется функция `file_get_contents`, описание которой вы найдете по адресу *http://www.php.net/manual/en/function.file-get-contents.php*.

Примечание

Популярную (и очень мощную) альтернативу функции `file_get_contents` представляет библиотека **Client URL Library** (**CURL**). Более подробные сведения об этой библиотеке есть по адресам *http://curl.haxx.se, http://www.php.net/curl* и *http://www.zend.com/zend/tut/tutorial-thome3.php*. Однако для удовлетворения простейших нужд вполне достаточно функции `file_get_contents`.

Попробуем реализовать эту схему. Функциональность приложения будет той же самой, что и в предыдущем упражнении: получить случайное число и отобразить его, но на этот раз оно будет работать во всех броузерах.

Время действовать – применение доверенного сценария на стороне сервера для организации доступа к удаленному серверу

1. В каталоге `foundations` создайте подкаталог `proxyping`.
2. В подкаталоге `proxyping` создайте файл `proxyping.html`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
```

```
        <title>
            Практические примеры AJAX: Доступ к удаленному серверу
            через доверенный сценарий PHP
        </title>
        <script type="text/javascript" src="proxyping.js"></script>
    </head>
    <body onload="process()">
        Сервер, верни мне случайное число!<br/>
        <div id="myDivElement" />
    </body>
</html>
```

3. **В том же каталоге создайте файл** `proxyping.js`**. Обратите внимание, что содержимое этого файла во многом повторяет содержимое файла** `ping.js`**, а различия выделены жирным шрифтом. (Мы удалили из функции** `process()` **код, который был предназначен для обслуживания модели безопасности в броузере Mozilla, изменили адрес сервера, убрали параметр** `num` **из сценария, поскольку теперь мы будем запрашивать только одно число за раз, и добавили код обработки ошибок.)**

```
// переменная для хранения ссылки на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// переменные для хранения адреса удаленного сервера
// и параметров запроса
var serverAddress = "proxyping.php ";
var serverParams = "min=1" +   // минимально возможное случайное число
                   "&max=100"; // максимально возможное случайное число
// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
```

```
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// производит асинхронное обращение к серверу
function process()
{
    // продолжать, только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // попытаться установить соединение с сервером
        try
        {
            // инициировать доступ к серверу
            xmlHttp.open("GET", serverAddress + "?" + serverParams, true);
            xmlHttp.onreadystatechange = handleRequestStateChange;
            xmlHttp.send(null);
        }
        // вывести сообщение об ошибке в случае неудачи
        catch (e)
        {
            alert("Невозможно соединиться с сервером:\n" + e.toString());
        }
    }
}

// эта функция вызывается при изменении состояния запроса HTTP
function handleRequestStateChange()
{
    // когда readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if (xmlHttp.status == 200)
        {
            try
```

```
            {
                // обработать ответ, полученный от сервера
                handleServerResponse();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                alert("Ошибка чтения ответа: " + e.toString());
            }
        }
        else
        {
            // вывести сообщение о состоянии
            alert("Возникли проблемы во время получения данных:\n" +
                  xmlHttp.statusText);
        }
    }
}

// обрабатывает ответ, полученный от сервера
function handleServerResponse()
{
    // прочитать сообщение, полученное от сервера
    var response = xmlHttp.responseText;
    // если ответ содержит более 3 символов или не содержит ни одного,
    // то мы предполагаем, что получили сообщение об ошибке от сервера
    if(response.length > 3 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
    // получить ссылку на элемент <div>
    myDiv = document.getElementById("myDivElement");
    // вывести полученный код HTML
    myDiv.innerHTML = "Сервер ответил: " + response + "<br/>";
}
```

4. Создайте доверенный сценарий PHP с именем `proxyping.php`:

```
<?php
// загрузить модуль обработки ошибок
require_once('error_handler.php');
// не дать возможность клиентскому броузеру помещать результаты в кэш
header('Expires: Wed, 23 Dec 1980 00:30:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
// получить параметры запроса
$num = 1; // этот параметр жестко задан на сервере
$min = $_GET['min'];
$max = $_GET['max'];
// собрать в одну строку адрес удаленного сервера и параметры
$serverAddress = 'http://www.random.org/cgi-bin/randnum';
$serverParams = 'num=' . $num .  // количество запрашиваемых чисел
                '&min=' . $min . // минимально возможное число
```

```
                '&max=' . $max;  // максимально возможное число
// получить случайное число с удаленного сервера
$randomNumber = file_get_contents($serverAddress . '?' . $serverParams);
// вывести полученное случайное число
echo $randomNumber;
?>
```

5. И наконец, добавим функцию обработки ошибок. Да, это увеличит объем работы с клавиатурой, но добавит полезные возможности в приложение (код этого сценария можно скопировать из другого упражнения, поскольку он не претерпел никаких изменений). Создайте новый файл `error_handler.php` и добавьте в него следующий код:

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

6. Откройте страницу *http://localhost/ajax/foundations/proxyping/proxyping.html* в броузере (хоть бы даже и в Opera) и полюбуйтесь на случайное число, которое вы получите (рис. 3.13).

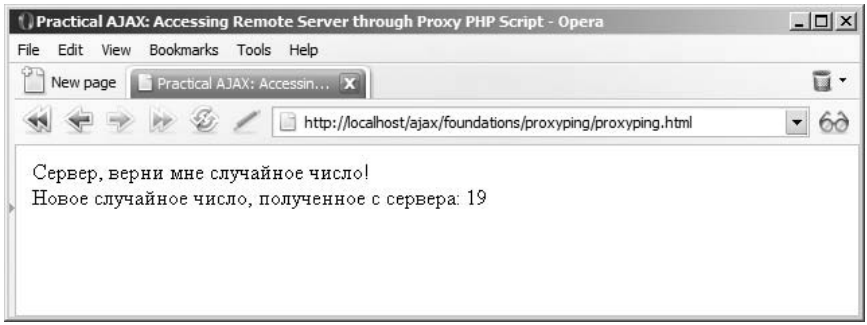

Рис. 3.13. *Использование доверенного сценария PHP для доступа к удаленному серверу*

Что происходит внутри?

Коду JavaScript позволено обращаться к серверу, с которого он был загружен. Мы поместили на наш сервер сценарий `proxypng.php`, который обращается к генератору случайных чисел от имени клиента.

Чтобы оставить за клиентом возможность определять диапазон, в котором должно находиться случайное число, сценарий PHP принимает параметры `min` и `max`, которые затем передаются серверу генератора случайных чисел. Мы не принимаем от клиента параметр `num`, поскольку теперь не предусматриваем для клиента возможность запросить более одного числа за раз. В данном упражнении мы предполагаем, что если ответ от сервера содержит более 3 символов, значит, получен отчет об ошибке:

```
// обрабатывает ответ, полученный от сервера
function handleServerResponse()
{
    // прочитать сообщение, полученное от сервера
    var response = xmlHttp.responseText;
    // если ответ содержит более 3 символов или не содержит ни одного,
    // то мы предполагаем, что получили сообщение об ошибке от сервера
    if(response.length > 3 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
```

Примечание

Ошибки возможны как на стороне клиента, так и на стороне сервера. Мы сделали все возможное, чтобы защитить клиента, поместив блоки `try/catch` в ключевые части кода. Но ошибка на стороне сервера доставляется клиенту не так, как ошибка, возникшая в самом клиенте. Поэтому мы должны проанализировать данные, полученные от сервера, и если они выглядят не так, как мы ожидаем, то должны сами сгенерировать ошибку с помощью оператора `throw`.

Если параметр `display_errors` в файле `php.ini` выключен, то ошибки синтаксического анализа файла сценария или фатальные ошибки заносятся только в файл журнала сервера Apache (`Apache/logs/error.log`), а результатом работы сценария будет пустая посылка. Поэтому, если мы принимаем ответ, который не содержит данных, мы предполагаем, что что-то произошло на сервере, и создаем сообщение об ошибке на стороне клиента.

Так, попытавшись открыть страницу при отсутствии доступа к Интернету (удаленный сервер недоступен), вы получите сообщение об ошибке, представленное на рис. 3.14 (текст сообщения будет отличаться от приведенного здесь, если параметр `display_errors` в файле `php.ini` выключен).

Код сценария `proxyping.php` ничего не делает с параметрами, полученными методом `GET`, он лишь отправляет их серверу генератора случайных чисел. Стоит обратить внимание на один интересный момент, а именно на способ, которым мы устанавливаем **срок действия страницы**. Установка срока действия страницы очень важна для нас, по-

Рис. 3.14. *Сообщение об ошибке, возникающей из-за отсутствия соединения с Интернетом*

скольку сервер всегда вызывается по одному и тому же URL, и броузер клиента может решить поместить результат обращения в кэш, а нам это совершенно не нужно, т. к. в этом случае числа, получаемые клиентом, перестали бы быть случайными.[1]

```
<?php
// загрузить модуль обработки ошибок
require_once('error_handler.php');
// не дать возможность клиентскому броузеру помещать результаты в кэш
header('Expires: Wed, 23 Dec 1980 00:30:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
```

Прекрасную статью, посвященную проблеме кэширования страниц и PHP, можно найти по адресу *http://www.sitepoint.com/article/php-anthology-2-5-caching*. Вся остальная часть сценария с помощью функции `file_get_contents` получает случайное число от удаленного сервера и передает его клиенту.

```
// получить параметры запроса
$num = 1; // этот параметр жестко задан на сервере
$min = $_GET['min'];
$max = $_GET['max'];
// собрать в одну строку адрес удаленного сервера и параметры
$serverAddress = 'http://www.random.org/cgi-bin/randnum';
$serverParams = 'num=' . $num .  // количество запрашиваемых чисел
                '&min=' . $min . // минимально возможное число
                '&max=' . $max;  // максимально возможное число
// получить случайное число с удаленного сервера
$randomNumber = file_get_contents($serverAddress . '?' . $serverParams);
// вывести полученное случайное число
echo $randomNumber;
?>
```

[1] Значения стали бы повторяющимися. – *Примеч. науч. ред.*

Основные принципы выполнения повторяющихся асинхронных запросов

Очень часто при разработке приложений AJAX приходится создавать клиентские сценарии, которые должны получать данные с сервера через регулярные интервалы времени. Такие ситуации чрезвычайно распространены, с многими из них вы встретитесь в этой книге, а еще чаще – в своих реальных проектах.

JavaScript предлагает четыре функции, способные помочь в реализации алгоритмов выполнения повторяющихся задач (через регулярные интервалы времени или по заданному расписанию), – `setTimeout`, `setInterval`, `clearTimeout` и `clearInterval`, которые могут применяться примерно так:

```
// использование функций setTimeout и clearTimeout
timerId = window.setTimeout("function()", интервал_в_миллисекундах);
window.clearTimeout(timeId);
// использование функций setInterval и clearInterval
timerId = window.setInterval("function()", интервал_в_миллисекундах);
window.clearInterval(timeId);
```

Функция `setTimeout` запускает указанную функцию один раз через заданный интервал времени. Функция `setInterval` запускает указанную функцию многократно через заданные интервалы времени, либо пока не будет вызвана функция `clearInterval`. В большинстве приложений AJAX мы отдаем предпочтение функции `setTimeout`, потому что она обеспечивает большую гибкость при организации доступа к серверу.

В демонстрационных целях мы расширим клиентскую часть, которая получает случайные числа, добавив в нее следующие улучшения:

- Послав запрос серверу, мы будем ожидать, пока не придет ответ, содержащий случайное число, а затем с помощью функции `setTimeout` перезапустим данную последовательность действий (пошлем серверу новый запрос) через одну секунду. Таким образом, интервал между двумя запросами будет составлять одну секунду плюс время, которое займет прием случайного числа. Для того чтобы посылать запросы через точные интервалы времени, надо обратиться к функции `setInterval`, но тогда придется проверять, не занят ли объект `XMLHttpRequest` обработкой предыдущего запроса (что вполне возможно, если сеть слишком медленная или сервер перегружен).
- В этом новом примере мы также время от времени будем проверять готовность сервера. Служба генератора случайных чисел имеет буфер случайных чисел, предназначенный для удовлетворения запросов. Проверить степень заполнения буфера может любой желающий, для чего он должен обратиться по адресу *http://www.random.org/cgi-bin/checkbuf*. Наша программа будет проверять эту страницу через каждые 10 запросов и выполнять очередную серию запросов, только если степень заполнения буфера не ниже 50%.

Внешний вид приложения показан на рис. 3.15.

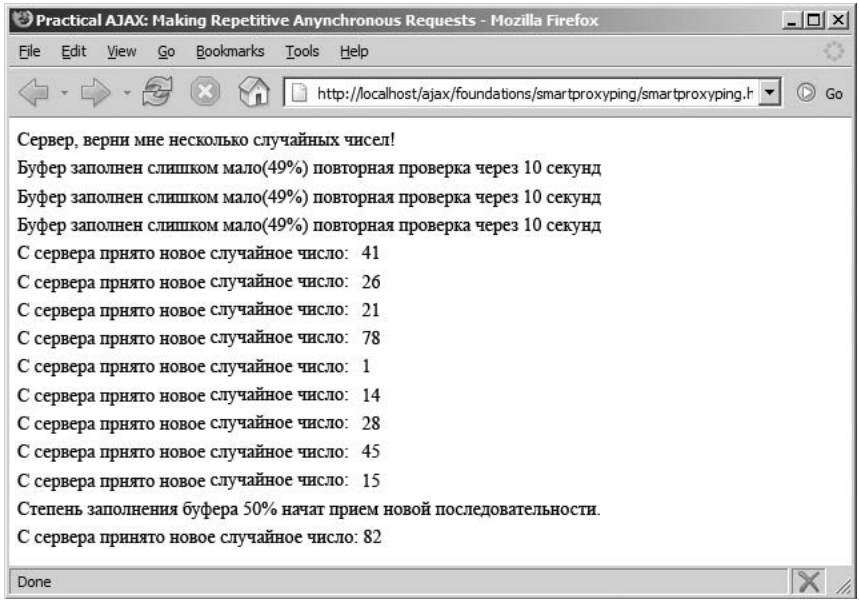

Рис. 3.15. *Выполнение повторяющихся асинхронных запросов*

Любая повторяющаяся последовательность действий должна откуда-то начинаться. В нашем приложении все начинается в функции `process()`. Там мы будем решать, какой запрос послать серверу: мы можем запросить новое случайное число или проверить степень заполнения буфера на сервере генератора случайных чисел. Мы будем проверять степень заполнения буфера через каждые 10 запросов и по умолчанию не будем запрашивать новые числа, если степень заполнения его упадет ниже 50%. Алгоритм представлен блок-схемой (рис. 3.16).

По умолчанию функция `setTimeout` вызывается только в случае успешного завершения запроса HTTP (ее вызов в блоке `catch` не предусматривается). (В зависимости от особенностей приложения можно продолжать попытки вызвать сервер, даже если возникла ошибка.)

Время действовать – реализация алгоритма выполнения повторяющихся действий

1. В каталоге `foundations` создайте каталог `smartproxyping`.
2. В каталоге `smartproxyping` создайте файл `smartproxyping.html`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>
```

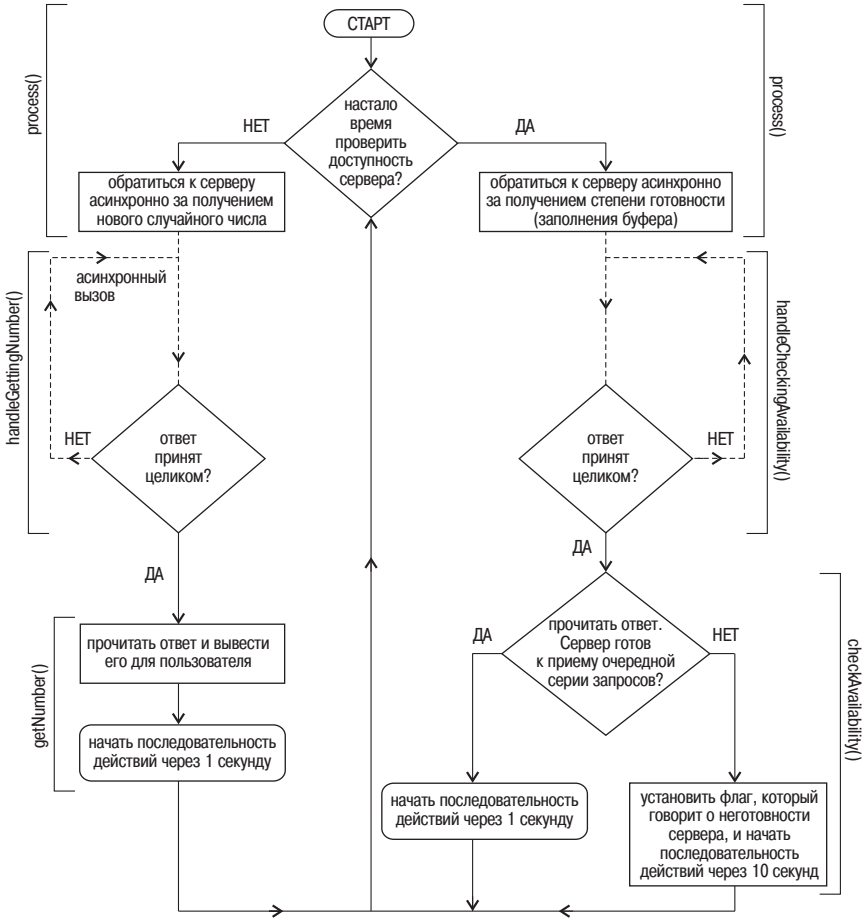

Рис. 3.16. *Блок-схема алгоритма приема случайных чисел*

```
            Практические примеры AJAX: Выполнение повторяющихся
            асинхронных запросов
        </title>
        <script type="text/javascript" src="smartproxyping.js"></script>
    </head>
    <body onload="process()">
        Сервер, дай мне несколько случайных чисел!<br/>
        <div id="myDivElement" />
    </body>
</html>
```

3. В том же каталоге создайте файл `smartproxyping.js`**:**

```
// переменная для хранения ссылки на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
```

```
// переменные для хранения адреса удаленного сервера и параметров запроса
var serverAddress = "smartproxyping.php";
var getNumberParams = "action=GetNumber" + // получить новое
                                           // случайное число
                      "&min=1" +  // минимально возможное случайное число
                      "&max=100"; // максимально возможное случайное число
var checkAvailabilityParams = "action=CheckAvailability";
// переменные, используемые при проверке готовности сервера
var requestsCounter = 0; // количество полученных случайных чисел
var checkInterval = 10;  // интервал между проверками готовности сервера
var updateInterval = 1;  // интервал между попытками получить новое число
var updateIntervalIfServerBusy = 10; // время ожидания
                                     // в случае занятости сервера
var minServerBufferLevel = 50; // минимально допустимый
                               // уровень заполнения буфера

// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или его более старая версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
```

```
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// производит асинхронное обращение к серверу
function process()
{
    // продолжать только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // попытаться установить соединение с сервером
        try
        {
            // если это первый вызов функции или было
            // выполнено заданное количество запросов,
            // необходимо проверить готовность сервера,
            // иначе запросить новое случайное число
            if (requestsCounter % checkInterval == 0)
            {
                // проверить готовность сервера
                xmlHttp.open("GET", serverAddress + "?" +
                             checkAvailabilityParams, true);
                xmlHttp.onreadystatechange = handleCheckingAvailability;
                xmlHttp.send(null);
            }
            else
            {
                // получить новое число
               xmlHttp.open("GET", serverAddress + "?" + getNumberParams,
                             true);
                xmlHttp.onreadystatechange = handleGettingNumber;
                xmlHttp.send(null);
            }
        }
        catch(e)
        {
            alert("Невозможно соединиться с сервером:\n" + e.toString());
        }
    }
}

// эта функция вызывается при изменении состояния запроса HTTP
function handleCheckingAvailability()
{
    // когда readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if (xmlHttp.status == 200)
        {
```

```
            try
            {
                // обработать ответ, полученный от сервера
                checkAvailability();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                alert("Ошибка чтения данных о готовности сервера:\n" +
                      e.toString());
            }
        }
        else
        {
            // вывести сообщение о состоянии
            alert("Ошибка чтения данных о готовности сервера:\n" +
                  e.toString());
        }
    }
}

// обрабатывает ответ, полученный от сервера
function checkAvailability()
{
    // прочитать сообщение, полученное от сервера
    var response = xmlHttp.responseText;
    // если ответ слишком длинный или не содержит ни одного символа,
    // то мы предполагаем, что приняли сообщение об ошибке с сервера
    if(response.length > 5 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
    // получить ссылку на элемент <div>
    myDiv = document.getElementById("myDivElement");
    // вывести полученный код HTML
    if (response >= minServerBufferLevel)
    {
        // вывести новое сообщение для пользователя
        myDiv.innerHTML += "Степень заполнения буфера " + response + "%, "
                    + "начат прием новой последовательности чисел. <br/>";
        // нарастить счетчик принятых чисел
        requestsCounter++;
        // возобновить последовательность действий
        setTimeout("process();", updateInterval * 1000);
    }
    else
    {
        // вывести новое сообщение для пользователя
        myDiv.innerHTML += "Буфер заполнен слишком мало ("
                           + response + "%), "
                           + "повторная поверка через "
                           + updateIntervalIfServerBusy
                           + " секунд. <br/>";
```

```
        // возобновить последовательность действий
        setTimeout("process();", updateIntervalIfServerBusy * 1000);
    }
}

// эта функция вызывается при изменении состояния запроса HTTP
function handleGettingNumber()
{
    // когда readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if (xmlHttp.status == 200)
        {
            try
            {
                // обработать ответ, полученный от сервера
                getNumber();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                alert("Ошибка чтения нового числа: " + e.toString());
            }
        }
        else
        {
            // вывести сообщение о состоянии
            alert("Ошибка чтения нового числа:\n" + xmlHttp.statusText);
        }
    }
}

// обрабатывает ответ, полученный от сервера
function getNumber()
{
    // прочитать сообщение, полученное от сервера
    var response = xmlHttp.responseText;
    // если ответ слишком длинный или не содержит ни одного символа,
    // то мы предполагаем, что приняли сообщение об ошибке с сервера
    if(response.length > 5 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
    // получить ссылку на элемент <div>
    myDiv = document.getElementById("myDivElement");
    // вывести полученный код HTML
    myDiv.innerHTML += "С сервера принято новое случайное число: "
                       + response + "<br/>";
    // нарастить счетчик запросов
    requestsCounter++;
    // возобновить последовательность действий
    setTimeout("process();", updateInterval * 1000);
}
```

4. **В том же каталоге создайте файл** `smartproxyping.php`**:**

```
<?php
// загрузить модуль обработки ошибок
require_once('error_handler.php');
// не дать возможность клиентскому броузеру помещать результаты в кэш
header('Expires: Wed, 23 Dec 1980 00:30:00 GMT'); // время в прошлом
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
// определить выполняемое действие
$action = $_GET['action'];
// проверить готовность или запросить новое число?
if ($action == 'GetNumber')
{
    $num = 1; // количество задается жестко, так как клиент не умеет
              // обрабатывать большее количество чисел
    $min = $_GET['min'];
    $max = $_GET['max'];
    // адрес сервера и параметры вызова
    $serverAddress = 'http://www.random.org/cgi-bin/randnum';
    $serverParams = 'num=' . $num .  // количество случайных чисел
                    '&min=' . $min . // минимально возможное случайное число
                   '&max=' . $max;  // максимально возможное случайное число
    // принять случайное число с удаленного сервера
    $randomNumber = file_get_contents($serverAddress . '?' .
                                      $serverParams);
    // вывести полученное число
    echo $randomNumber;
}
elseif ($action == 'CheckAvailability')
{
    // адрес страницы, которая возвращает степень заполнения буфера
    $serverAddress = 'http://www.random.org/cgi-bin/checkbuf';
    // степень заполнения принимается в форме 'x%'
    $bufferPercent = file_get_contents($serverAddress);
    // извлечь число
    $buffer = substr($bufferPercent, 0, strlen($bufferPercent) - 2);
    // вывести число
    echo $buffer;
}
else
{
    echo 'Получен неверный запрос.';
}
?>
```

5. **В том же каталоге создайте файл** `error_handler.php`**, который должен быть совершенно идентичным одноименному файлу из предыдущего упражнения:**

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

6. Откройте страницу *http://localhost/ajax/foundations/smartproxyping/smartproxyping.html*. Результат должен быть похож на то, что показано на рис. 3.15.

Что происходит внутри?

Наш клиент из данного примера знает, как время от времени выполнять проверку готовности сервера. Для этих целей сервер генератора случайных чисел предоставляет страницу *http://www.random.org/cgi-bin/checkbuf*.

Код JavaScript в `smartproxyping.js` начинается с определения ряда глобальных переменных, которые используются для управления поведением программы:

```
// переменная для хранения ссылки на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// переменные для хранения адреса удаленного сервера и параметров запроса
var serverAddress = "smartproxyping.php";
var getNumberParams = "action=GetNumber" + // получить новое случайное число
                      "&min=1" +  // минимально возможное случайное число
                      "&max=100"; // максимально случайное число
var checkAvailabilityParams = "action=CheckAvailability";
// переменные, используемые при проверке готовности сервера
var requestsCounter = 0; // количество полученных случайных чисел
var checkInterval = 10;  // интервал между проверками готовности сервера
var updateInterval = 1;  // интервал между попытками получить новое число
var updateIntervalIfServerBusy = 10; // время ожидания
                                     // в случае занятости сервера
var minServerBufferLevel = 50; // минимально допустимый
                               // уровень заполнения буфера
```

В этих переменных хранятся данные, необходимые для выполнения запросов к серверу. В getNumberParams содержится строка с параметрами запроса нового случайного числа, а в переменной checkAvailabilityParams – строка с параметрами запроса степени заполнения буфера. Другие переменные предназначены для управления интервалами между асинхронными запросами.

По сравнению с предыдущими упражнениями есть нововведение – здесь ответы сервера обрабатываются двумя функциями: handleCheckingAvailability и handleGettingNumber. В зависимости от запрошенного у сервера действия функция process назначает то одну, то другую в качестве функции обратного вызова.

В этой программе функция process() вызывается не один раз, а много, и каждый раз она должна решить, какое действие будет следующим – запросить ли новое случайное число или надо проверить степень заполнения буфера? Принять решение помогает переменная requestsCounter, в которой хранится количество случайных чисел, полученных с момента последней проверки готовности:

```
function process()
{
    // ...
    if (requestsCounter % checkInterval == 0)
    {
        // проверить готовность сервера
        xmlHttp.open("GET", serverAddress + "?" +
                     checkAvailabilityParams, true);
        xmlHttp.onreadystatechange = handleCheckingAvailability;
        xmlHttp.send(null);
    }
    else
    {
        // получить новое число
        xmlHttp.open("GET", serverAddress + "?" + getNumberParams, true);
        xmlHttp.onreadystatechange = handleGettingNumber;
        xmlHttp.send(null);
    }
    //...
}
```

Функции handleCheckingAvailability и handleGettingNumber похожи друг на друга – это узкоспециализированные версии функции handleRequestStateChange, известной вам по другим упражнениям. Их основная задача состоит в том, чтобы дождаться ответа сервера и вызвать вспомогательную функцию (checkAvailability или getNumber) для его обработки.

Обратите внимание на параметр action, по значению которого сценарий PHP определяет, какое действие надо выполнить. На стороне сервера в сценарии smartproxyping.php после загрузки модуля обработки

ошибок считывается параметр `action`, и на основе его значения принимается решение о дальнейших действиях:

```
<?php
// загрузить модуль обработки ошибок
require_once('error_handler.php');
// не дать возможность клиентскому броузеру помещать результаты в кэш
header('Expires: Wed, 23 Dec 1980 00:30:00 GMT'); // время в прошлом
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
// определить выполняемое действие
$action = $_GET['action'];
// проверить готовность или запросить новое число?
if ($action == 'GetNumber')
{
    // ...
```

Если была запрошена операция `GetNumber`, то мы с помощью функции PHP `file_get_contents` получаем от удаленного сервера новое случайное число:

```
if ($action == 'GetNumber')
{
    $num = 1; // количество задается жестко, т. к. клиент не умеет
              // обрабатывать большее количество чисел
    $min = $_GET['min'];
    $max = $_GET['max'];
    // адрес сервера и параметры вызова
    $serverAddress = 'http://www.random.org/cgi-bin/randnum';
    $serverParams = 'num=' . $num .  // количество случайных чисел
                    '&min=' . $min . // минимально возможное случайное число
                    '&max=' . $max;  // максимально возможное случайное число
    // принять случайное число с удаленного сервера
    $randomNumber = file_get_contents($serverAddress . '?' . $serverParams);
    // вывести полученное число
    echo $randomNumber;
}
```

Если была запрошена операция `CheckAvailability`:

```
elseif ($action == 'CheckAvailability')
{
    // адрес страницы, которая возвращает степень заполнения буфера
    $serverAddress = 'http://www.random.org/cgi-bin/checkbuf';
    // степень заполнения принимается в форме 'x%'
    $bufferPercent = file_get_contents($serverAddress);
    // извлечь число
    $buffer = substr($bufferPercent, 0, strlen($bufferPercent) - 2);
    // вывести число
    echo $buffer;
}
```

Обратите внимание, что запросы, выполняемые функцией `file_get_contents`, не асинхронные, но это нам и не требуется. Сценарий PHP не поддерживает постоянное соединение с клиентом и может работать столько времени, сколько потребуется, чтобы получить ответ от удаленного сервера. На стороне клиента функции `checkAvailability` и `getNumber` принимают ответы, которые мы создали в сценарии PHP. Эти функции начинаются с чтения ответа и проверки его размера:

```
// обрабатывает ответ, полученный от сервера
function getNumber()
{
    // прочитать сообщение, полученное от сервера
    var response = xmlHttp.responseText;
    // если ответ слишком длинный или не содержит ни одного символа,
    // то мы предполагаем, что приняли сообщение об ошибке с сервера
    if(response.length > 5 || response.length == 0)
        throw(response.length == 0 ? "Server error" : response);
```

Примечание

Это один из способов, позволяющих убедиться, что сценарий PHP отработал без ошибок. Проверка успешности завершения сценария, основанная на размере ответа, примитивна, но это достаточно эффективный метод. Фатальные ошибки не могут быть перехвачены кодом сценария PHP, и поэтому реализовать универсальный и мощный механизм обработки ошибок чрезвычайно сложно.

В окончательной версии приложения недостаточно обнаружить ошибки; надо также подумать о том, как их обрабатывать, и здесь выбор из огромного количества вариантов зависит от конкретных обстоятельств. Имейте в виду – *пользователей не интересуют технические подробности ошибок*. Например, в нашем сценарии достаточно было вывести сообщение: «Сервер временно недоступен. Пожалуйста, повторите попытку позже».

Однако если вы захотите выводить точные сведения об ошибках, учтите, что при выводе ваших ошибок добавляется символ перевода строки `\n`, а сообщения о фатальных ошибках PHP выводятся в формате HTML. Сообщения, выводимые в отдельных диалоговых окнах, придется как-то отформатировать.

После вывода результатов мы перезапускаем последовательность действий, для чего вызываем функцию `setTimeout`:

```
    // возобновить последовательность действий
    setTimeout("process();", updateInterval * 1000);
}
```

Работа с MySQL

Реализация приложений любого типа, которые динамически генерируют выходные данные, требует наличия системы хранения и управления данными. Чаще всего для хранения данных в приложениях применяются **системы управления реляционными базами данных**

(СУРБД) – специализированное программное обеспечение, предназначенное для хранения и управления данными.

Подобно многим другим компонентам, базы данных не являются составной частью AJAX, но при создании веб-приложений вам едва ли удастся обойтись без них. В этой книге мы продемонстрируем простые примеры приложений, которые не нуждаются в больших объемах данных, но обойтись без базы данных все-таки не могут. Для примеров из этой книги в качестве СУРБД мы выбрали MySQL, весьма популярную среди разработчиков, применяющих PHP. Однако алгоритмы взаимодействия с базами данных настолько универсальны, что вы без труда сможете переориентировать их на работу с другими СУБД.

Для создания приложения, использующего базу данных, необходимо знать:

1. Как создавать таблицы в базе данных, которые будут хранить ваши данные.
2. Как писать запросы SQL.
3. Как соединиться с базой данных из сценария на языке PHP.
4. Как посылать запросы SQL и как извлекать результаты выполнения этих запросов.

Примечание

Еще раз напомним, что в данной книге мы сможем охватить лишь самые основы работы с PHP и базами данных MySQL.[1] Очень хорошо составленные руководства к PHP и MySQL вы без труда найдете в Интернете.

Создание таблиц

Для того чтобы создавать таблицы в базе данных, необходимо знать основные принципы построения реляционных БД. Таблица состоит из **столбцов (полей)** и **строк (записей)**.[2] Создавая таблицу, надо определить ее поля, которые могут обладать различными характеристиками. Далее мы обсудим:

- Первичные ключи
- Типы данных
- Поля NULL и NOT NULL
- Автоинкрементные поля
- Индексы

[1] И документация, и сами программы, относящиеся к MySQL, широко представлены и в русскоязычном Интернете (*http://www.mysql.ru/*). – *Примеч. науч. ред.*

[2] Теория реляционных баз данных оперирует также понятиями атрибутов (применительно к столбцам) и кортежей (применительно к строкам). Они, наверное, точнее всего отражают суть, особенно это касается столбцов. – *Примеч. науч. ред.*

Первичный ключ – это специальное поле (или группа полей) таблицы, значение которого уникально для каждой записи. Поле, являющееся первичным ключом, не допускает хранения повторяющихся значений. Когда первичный ключ состоит более чем из одного поля, то требование уникальности предъявляется ко всему набору полей, а не к каждому из них по отдельности. Чисто технически `PRIMARY KEY` является ограничением целостности (правилом), которое накладывается на поле, но для удобства, когда мы говорим «первичный ключ», мы обычно подразумеваем само поле, на которое наложено ограничение целостности `PRIMARY KEY`. При создании ограничения `PRIMARY KEY` одновременно для заданного поля создается уникальный индекс, который значительно ускоряет операции поиска по таблице.

Каждое поле имеет определенный **тип данных**, который определяет его размер и поведение. Существует три основных категории типов данных (*числовые типы*, *символьные и строковые типы*, и *типы* хранения сведений о *дате и времени*), каждая из категорий подразделяется на несколько типов данных. За полной информацией по данной теме обращайтесь к официальной документации по MySQL 5, по адресу *http://dev.mysql.com/doc/refman/5.0/en/data-types.html*.[1]

При проектировании будущей таблицы вы должны определить, какие поля обязательно должны иметь конкретные значения, и отметить их как **NOT NULL**. Такое определение говорит о том, что поле не может быть пустым, то есть не может хранить значения `NULL`. Смысл значения `NULL` – *не определено*. Если при чтении содержимого таблицы вы увидите значение `NULL`, это говорит о том, что значение данного поля не было задано. Обратите внимание: пустые строки, или строки, содержащие одни пробелы, или число «0» (для числовых полей) являются действительными (не `NULL`) значениями. Поля, составляющие первичный ключ, не могут содержать значения `NULL`.

Существует возможность вместо (или совместно) определения `NOT NULL` для некоторых полей определить **значение по умолчанию**. В этом случае, если при создании новой записи для такого поля не задается конкретное значение, в него будет записано значение по умолчанию. В качестве значения по умолчанию можно также указать функцию, которая будет вызываться для получения значения по умолчанию каждый раз, когда это необходимо.

Еще один способ генерации новых значений предоставляют **автоинкрементные** (`auto_increment`) поля.[2] Возможность автоматического инкремента зачастую используется для определения полей первичного ключа, являющиеся представлением некоторого рода идентификато-

---

[1] Есть документация на русском языке на том же ресурсе (*http://downloads.mysql.com/docs/refman-4.0-ru.html.zip*). – *Примеч. науч. ред.*

[2] В терминологии некоторых СУБД называемые также счетчиками. – *Примеч. науч. ред.*

ров, которые вы предпочтете генерировать автоматически. Атрибут auto_increment может быть установлен только для числовых полей. Он обеспечивает генерацию новых значений, которые будут автоматически увеличиваться на 1, таким образом, ни одно значение не сможет быть сгенерировано дважды.

Индексы – это объекты базы данных, используемые для повышения скорости выполнения операций. Индекс – это структура, которая значительно ускоряет *поиск* по заданному полю (или полям), но замедляет выполнение операций изменения и добавления новых записей (поскольку индексы также должен быть перестроены во время этих операций). Правильно подобранная комбинация индексов может дать вашему приложению огромный прирост скорости. В примерах из этой книги мы будем полагаться на индексы, которые будут строиться для полей, составляющих первичный ключ.

Создавать таблицы в базе данных можно с помощью запросов SQL или с помощью визуального интерфейса. Ниже приводится пример SQL-выражения, которое создает простую таблицу данных:

```
CREATE TABLE users
(
    user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    user_name VARCHAR(32) NOT NULL,
    PRIMARY KEY (user_id)
);
```

Созданную таблицу можно изменить посредством оператора ALTER TABLE или вообще удалить ее оператором DROP TABLE. Для быстрого удаления и повторного создания таблицы применяется оператор TRUNCATE TABLE (фактически это операция удаления всех записей в таблице, но этот способ намного быстрее и к тому же сбрасывает автоинкрементные индексы).

В каждом упражнении приводится код SQL, создающий все необходимые таблицы. Исполнять этот код можно такой программой, как phpMyAdmin[1] (процедура установки описана в приложении А). Чтобы исполнить код SQL с помощью phpMyAdmin, необходимо соединиться[2] с базой данных, выбрав ее имя из списка Database, и щелкнуть по вкладке SQL основной панели, как показано на рис. 3.17.

Кроме этого phpMyAdmin предоставляет возможность создавать таблицы визуально, используя формы, как показано на рис. 3.18.

---

[1] Русскоязычную информацию и саму программу phpMyAdmin можно найти по адресу *http://www.php-myadmin.ru/*; на момент перевода книги последней стабильной версией была 2.8.0.3. – *Примеч. науч. ред.*

[2] Соединение с БД (точнее с ее системой управления), даже при размещении ее на локальном компьютере, происходит аналогично сетевому соединению. – *Примеч. науч. ред.*

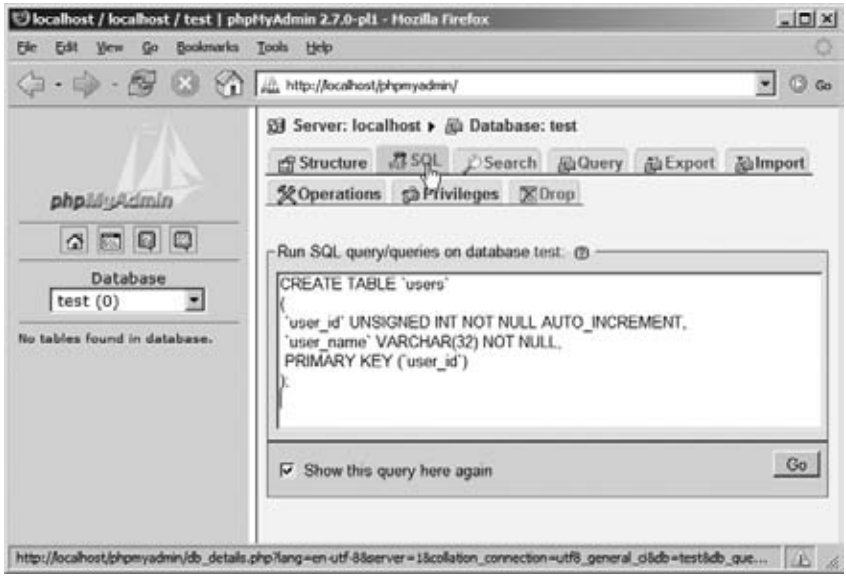

Рис. 3.17. *Исполнение кода SQL с помощью phpMyAdmin*

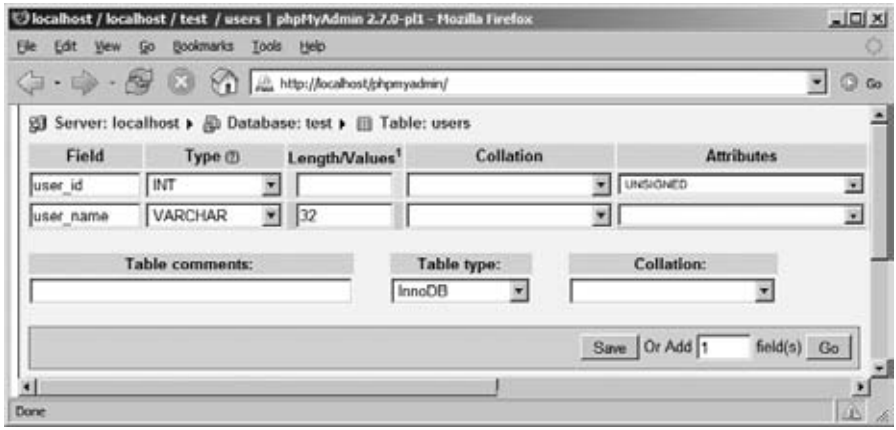

Рис. 3.18. *Создание новой таблицы с помощью phpMyAdmin Designer*

Примечание

Если наличие параметра `Table type` (рис. 3.18) вызвало у вас недоумение, прочитайте это примечание. MySQL отличается от других СУБД тем, что распространяется с различными механизмами управления данными. Наиболее популярные – **MyISAM** и **InnoDB**. Самое интересное, что в одной и той же базе данных могут находиться таблицы разных типов, и тип каждой определяется на этапе ее создания (иначе будет взято значение по умолчанию, а в большинстве конфигураций это тип MyISAM). Каждый из механизмов имеет свои достоинства и недостатки. Самым мощным считается InnoDB, полностью отвечающий требо-

ваниям **ACID** (**Atomicity, Consistency, Isolation, Durability – атомарность, непротиворечивость, изолированность, надежность**), предъявляемым к транзакциям; он обеспечивает блокировку на уровне отдельных записей, внешние ключи, ссылочную целостность и многие другие функциональные возможности. Основное преимущество механизма MyISAM перед остальными состоит в поддержке полнотекстового поиска и (возможно) в высокой скорости.

Действия с данными

Управление данными осуществляется командами языка SQL DML (Structured Query Language – Data Manipulation Language – языка манипулирования данными): `SELECT`, `INSERT`, `UPDATE` и `DELETE`, позволяющими извлекать, добавлять, изменять и удалять записи из таблиц с данными.[1] Это очень мощные и гибкие команды. Базовый синтаксис:

```
SELECT <column list>
FROM <table name(s)>
[WHERE <restrictive condition(s)>]

INSERT INTO <table name> [(column list)]
VALUES (column values)

UPDATE <table name>
SET <column name> = <new value> [, <column name> = <new value> ... ]
[WHERE <restrictive condition>]

DELETE FROM <table name>
[WHERE <restrictive condition>]
```

Вот несколько основных положений, которые необходимо иметь в виду:

- Из соображений удобочитаемости код SQL может записываться в одной или в нескольких строках.
- Несколько подряд исполняемых команд SQL разделяются символом точка с запятой (`;`).
- Значения, помещенные на синтаксических диаграммах в квадратные скобки, могут отсутствовать. (Будьте осторожны при работе с оператором `DELETE` – если не задать ограничительное условие, будут удалены *все* записи в таблице.)
- В операторе `SELECT` вместо списка полей можно указать символ `*`, что соответствует выборке всех полей в таблице.
- Код SQL безразличен к регистру символов, но лучше записывать операторы SQL символами верхнего регистра, а имена таблиц и полей – символами нижнего регистра. Следование рекомендациям еще никому не мешало.

[1] Огромное количество информации по SQL можно найти на сайте *http://www.sql.ru/*. – *Примеч. науч. ред.*

Проверить действие этих команд можно на таблице `users`, которую мы описали ранее. Откройте вкладку `SQL` в phpMyAdmin и выполните такие команды:

```
INSERT INTO users (user_name) VALUES ('john');
INSERT INTO users (user_name) VALUES ('sam');
INSERT INTO users (user_name) VALUES ('ajax');

SELECT user_id, user_name FROM users;

UPDATE users SET user_name='cristian' WHERE user_id=1;

SELECT user_id, user_name FROM users;

DELETE FROM users WHERE user_id=3;

SELECT * FROM users WHERE user_id>1;
```

В книге вам встретится немало более сложных примеров запросов, которые будут объясняться по мере необходимости. Помните, язык SQL – это очень обширная тема, поэтому вам наверняка придется обратиться за дополнительной информацией к другим источникам, если ранее вы не занимались созданием запросов на языке SQL.

Соединение с базой данных и исполнение запросов

В наших примерах код, ответственный за соединение с базой данных, будет записываться на языке PHP. Как показано на рис. 3.19, базы данных никогда не будут доступны клиенту напрямую – только через промежуточный сценарий PHP, реализующий бизнес-логику приложения.

Чтобы получить доступ к данным, сценарий PHP должен пройти процедуру аутентификации в базе данных.

Система безопасности базы данных, равно как и другие типы систем безопасности, включает в себя два важных понятия: **аутентификацию** и **авторизацию**. Аутентификация – это процедура однозначной идентификации пользователя, выполняемая с помощью какого-либо механизма регистрации (обычно путем ввода имени пользователя и пароля).

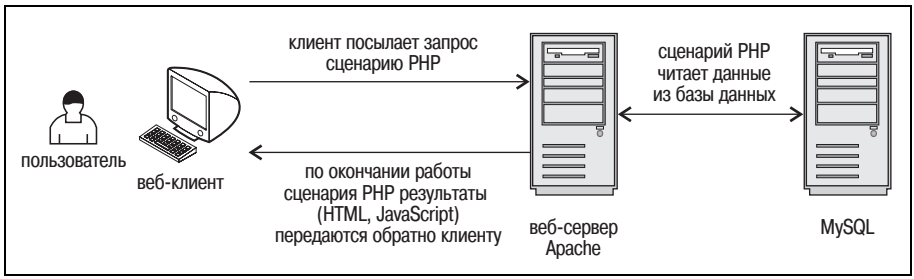

Рис. 3.19. *Пользователь подключается к базе данных через промежуточное программное обеспечение*

Авторизация определяет ресурсы, которые будут доступны, и действия, которые сможет выполнить аутентифицированный пользователь.

Если система безопасности MySQL настроена так, как показано в приложении А, то соединение с локальным сервером MySQL, с базой данных `ajax`, будет открываться под пользователем **ajaxuser** и с паролем **practical**. Эти сведения сохраняются в файле конфигурации `config.php`, который нетрудно изменить в случае необходимости. Сценарий `config.php` выглядит так:

```
<?
// сведения, необходимые для соединения с базой данных
define('DB_HOST', 'localhost');[1]
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

Эти сведения будут использоваться для выполнения операций в базе данных. Любая операция с базой данных состоит из трех обязательных действий:

1. Открытия соединения с базой данных.
2. Выполнения запросов SQL и чтения результатов.
3. Закрытия соединение с базой данных.

Хорошей практикой считается как можно более позднее открытие соединения с базой данных и как можно более раннее его закрытие, т. к. на поддержку открытого соединения с базой данных расходуются ресурсы сервера. В следующем отрывке кода показан простой сценарий PHP, который открывает соединение с базой данных, читает некоторые данные из нее и закрывает соединение:

```
// соединиться с базой данных
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);
// запрос SQL, который необходимо исполнить
$query = 'SELECT user_id, user_name FROM users';
// исполнить запрос
$result = $mysqli->query($query);
// выполнить какие-либо действия с результатами...
// ...
// закрыть входной поток
$result->close();
// закрыть соединение с базой данных
$mysqli->close();
```

[1] Только для случая размещения MySQL на том же компьютере (локально), на котором работает и веб-сервер с PHP, что совсем необязательно – в крупных системах они могут быть запущены на различных узлах локальной сети; общая работоспособность описываемых в книге технологий при этом не изменится. – *Примеч. науч. ред.*

Примечание

Обратите внимание, что доступ к MySQL обеспечивается библиотекой `mysqli`. Это новейшая версия библиотеки `mysql`, с улучшенными характеристиками, которая предоставляет как процедурный, так и объектно-ориентированный интерфейсы доступа к MySQL и поддерживает расширенные возможности MySQL. Если у вас установлена одна из старых версий MySQL или PHP, которые не поддерживают `mysqli`, то вместо нее можно использовать библиотеку `mysql`.

В следующем упражнении нет технологий, применяемых в AJAX. Это просто пример взаимодействия с базой данных MySQL из кода PHP.

Время действовать – PHP и MySQL

1. Соединитесь с базой данных и создайте таблицу с именем `users`, исполнив следующий код:

```
CREATE TABLE users
(
    user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    user_name VARCHAR(32) NOT NULL,
    PRIMARY KEY (user_id)
);
```

2. Исполнив следующие команды `INSERT`, заполните таблицу `users` данными:

```
INSERT INTO users (user_name) VALUES ('bogdan');
INSERT INTO users (user_name) VALUES ('filip');
INSERT INTO users (user_name) VALUES ('mihai');
INSERT INTO users (user_name) VALUES ('emilian');
INSERT INTO users (user_name) VALUES ('paula');
INSERT INTO users (user_name) VALUES ('cristian');
```

Примечание

Поле `user_id` определено как автоинкрементное, поэтому его значение будет генерироваться самой базой данных

3. В каталоге `foundations` создайте подкаталог с именем `mysql`.

4. В каталоге `mysql` создайте файл с именем `config.php` и добавьте в него код, изменив значения в соответствии с конфигурацией базы данных:

```
<?php
// сведения, необходимые для соединения с базой данных
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

5. **Теперь добавьте в каталог стандартный файл со сценарием обработки ошибок –** `error_handler.php`**. Вы можете просто скопировать его из предыдущих упражнений:**

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

6. **Создайте файл с именем** `index.php` **и добавьте в него следующий код:**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Практические примеры AJAX: PHP и MySQL</title>
    </head>
    <body>
<?php
// загрузить сценарий обработки ошибок и конфигурационный файл
require_once('error_handler.php');
require_once('config.php');
// открыть соединение с базой данных
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);
// запрос SQL, который необходимо исполнить
$query = 'SELECT user_id, user_name FROM users';
// исполнить запрос
$result = $mysqli->query($query);
// обойти в цикле результаты запроса
while ($row = $result->fetch_array(MYSQLI_ASSOC))
{
    // извлечь значения полей user id и name
    $user_id = $row['user_id'];
    $user_name = $row['user_name'];
    // выполнить действия с данными (здесь мы просто выводим их)
    echo 'Имя пользователя #' . $user_id . ' - ' . $user_name . '<br/>';
}
// закрыть входной поток
```

```
$result->close();
// закрыть соединение с базой данных
$mysqli->close();
?>
    </body>
</html>
```

7. Проверьте работу сценария, открыв в броузере страницу *http://localhost/ajax/foundations/mysql/index.php* (рис. 3.20).

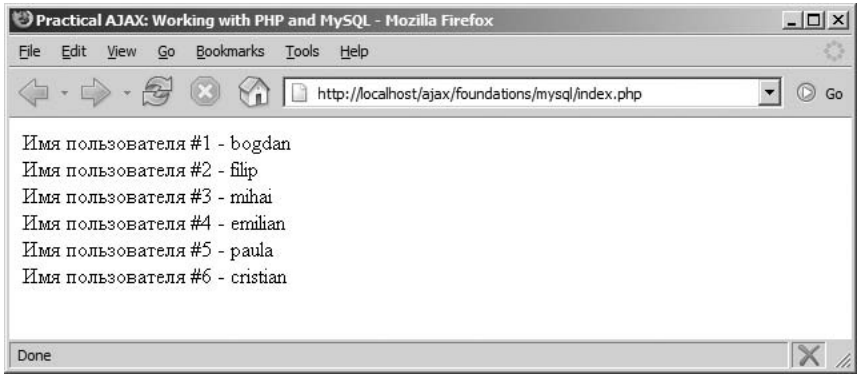

***Рис. 3.20.** Эти имена пользователей прочитаны из базы данных*

Что происходит внутри?

Прежде всего обратите внимание на то, что здесь не задействуется технология AJAX – этот пример просто демонстрирует возможность доступа к данным из PHP. Все самое интересное сосредоточено в файле `index.php`. Работа сценария начинается с загрузки модуля обработки ошибок и конфигурационного сценария:

```
<?php
// загрузить сценарий обработки ошибок и конфигурационный файл
require_once('error_handler.php');
require_once('config.php');
```

Затем открывается новое соединение с базой данных:

```
// открыть соединение с базой данных
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_DATABASE);
```

Обратите внимание, что при открытии соединения мы указываем сведения, относящиеся к базе данных, расположенной на сервере, а не к самому серверу. Мы соединяемся с базой данных `ajax`, содержащей таблицу `users`, созданную ранее. Запросы, выполняемые через открытое соединение, могут считаться операциями доступа к таблице `users`:

```
// запрос SQL, который необходимо исполнить
$query = 'SELECT user_id, user_name FROM users';
// исполнить запрос
```

```
$result = $mysqli->query($query);
```

После исполнения этих команд переменная `$result` будет содержать указатель на поток с результатами, из которого мы будем считывать данные строку за строкой посредством метода `fetch_array`. Этот метод возвращает массив полей текущей записи и перемещает указатель на следующую запись. Мы анализируем полученные результаты в цикле `while` запись за записью, пока не достигнем конца потока, извлекая из каждой записи отдельные ее поля:

```
// обойти в цикле результаты запроса
while ($row = $result->fetch_array(MYSQLI_ASSOC))
{
    // извлечь значения полей user id и name
    $user_id = $row['user_id'];
    $user_name = $row['user_name'];
    // выполнить действия с данными (здесь мы просто выводим их)
    echo 'Имя пользователя #' . $user_id . ' - ' . $user_name . '<br/>';
}
```

В заключение мы закрываем соединение, чтобы не расходовать понапрасну ресурсы сервера и не блокировать доступ к записям, т. к. это может отрицательно сказаться на работе других запросов, выполняющихся в это же время:

```
// закрыть входной поток
$result->close();
// закрыть соединение с базой данных
$mysqli->close();
?>
```

Технология обертывания и разделения функциональности

В этом разделе главы мы приведем базовую схему организации кода, на которой будут основываться все последующие упражнения. Большая часть из этих основных строительных блоков уже была продемонстрирована, за исключением класса, реализующего бизнес-логику на стороне сервера. Его мы продемонстрируем в следующем упражнении.

До сих пор весь код, работающий на стороне сервера, мы оформляли в виде единственного файла PHP. Чтобы добиться более высокой гибкости, мы разобьем серверную функциональность на два файла:

- Первый сценарий, `appname.php` (где `appname` соответствует названию приложения), представляет собой основную точку доступа для кода JavaScript, работающего на стороне клиента. Он обрабатывает входные параметры, принимаемые методами `POST` и `GET`, и на основе этих параметров принимает решения.

- Второй сценарий, `appname.class.php`, содержит вспомогательный класс `Appname`, отвечающий за реализацию фактической функциональности, необходимой для работы. В зависимости от запрошенных действий из сценария `appname.php` вызываются соответствующие методы этого класса.

Для полного понимания кода необходимо знать основы ООП и особенности их применения в PHP. Мы не рассматриваем эти аспекты в данной книге, но сделаем ряд замечаний, которые следует иметь в виду:

- ООП основано на понятии класса, который является прототипом будущих объектов.[1] Классы состоят из членов класса, куда входят методы (внутренние функции класса), конструктор, деструктор и поля класса (в других объектно-ориентированных языках классы могут состоять из еще большего количества типов членов). Поля класса очень похожи на переменные, но они видимы только в контексте класса.
- В классах можно реализовать два специальных метода – *конструктор* и *деструктор*. Конструктор `_construct()` вызывается автоматически при создании нового объекта этого класса. Конструктор удобен для начальной инициализации различных членов класса, поскольку он обязательно будет вызван, как только будет создан новый объект класса.
- Деструктор `_destruct()` вызывается автоматически при уничтожении объекта. Деструкторы очень удобны для выполнения заключительных операций. В большинстве упражнений соединение с базой данных закрывается именно в деструкторе, благодаря чему оно не останется открытым и не будет зря расходовать ресурсы сервера.
- Это правда, что для повышения производительности лучше открывать соединение с базой данных непосредственно тогда, когда оно необходимо, а не в конструкторе класса, а закрывать соединение лучше сразу же, как только в нем отпадет необходимость, а не в деструкторе. Однако мы для этих целей задействуем конструкторы и деструкторы, чтобы не усложнять код и избежать ошибок, оставив соединение открытым по забывчивости.

Обращаясь к членам класса, надо указывать имя объекта как часть полного имени. Для того чтобы обратиться к члену класса из метода этого же класса, применяется специальный объект `$this` (ссылка), который ссылается на текущий экземпляр класса.

Общедоступный интерфейс класса состоит из *общедоступных* (*public*) членов, к которым можно обращаться из-за пределов класса и которые могут использоваться программами, создавшими экземпляр класса. Члены класса могут иметь области видимости *public* (*общедоступные*),

[1] Понятие «класс» соответствует описанию типа переменных, тогда как сами экземпляры переменных этого класса называются «объектами». – *Примеч. науч. ред.*

private (*частные*) или *protected* (*защищенные*). Члены класса с областью видимости `private` могут использоваться только внутри класса, а члены с областью видимости `protected` – производными классами.

Разделение функциональности приложения на отдельные уровни – очень важный момент, т. к. позволяет создавать гибкие и расширяемые приложения, которые нетрудно дополнить новой функциональностью. В книгах Кристиана Дари (Cristian Darie) и Михая Бусики (Mihai Bucica) рассказано даже о том, как применять механизм шаблонов Smarty, который позволяет еще больше отделить логику представления от шаблонов HTML, чтобы дизайнеры могли не оглядываться на реализацию программной части сайта.

Примечание

Проектируя архитектуру приложения, необходимо помнить, что его мощь, гибкость и масштабируемость прямо пропорциональны времени, потраченному на проектирование и написание основного кода. Справочная информация по этой теме доступна по адресу *http://ajaxphp.packtpub.com/ajax/*.

В этом заключительном упражнении мы создадим простое, но вполне завершенное приложение AJAX, которое называется **friendly**. В нем реализовано множество методик, с которыми мы уже встречались. Приложение будет иметь стандартную структуру, составленную из следующих файлов:

- `index.html` – файл, изначально загружаемый пользователем. Он содержит код JavaScript, выполняющий асинхронные обращения к сценарию `friendly.php`.
- `friendly.css` – содержит каскадные таблицы стилей, используемые приложением.
- `friendly.js` – файл с кодом JavaScript, загружаемый клиентом вместе с файлом `index.html`. Он выполняет асинхронные обращения к сценарию PHP `friendly.php`, реализующему различные действия, обеспечивающие интерфейс с пользователем.
- `friendly.php` – сценарий PHP, размещаемый на том же самом сервере, что и `index.html`. Он обеспечивает функциональность сервера, отвечая на асинхронные запросы из кода JavaScript в `index.html`. Помните: очень важно, чтобы эти файлы размещались на одном и том же сервере, поскольку код JavaScript, исполняемый на стороне клиента, может не получить разрешение на доступ к другим серверам. В большинстве случаев файл `friendly.php` будет обращаться к функциональности еще одного файла PHP, `friendly.class.php`.
- `friendly.class.php` – сценарий PHP, который содержит класс с именем `Friendly`. Этот класс реализует бизнес-логику приложения. Он отвечает за взаимодействие с базой данных и выполнение прочих действий.

- `config.php` – хранит глобальные настройки приложения, например сведения, необходимые для установления соединения с базой данных.
- `error_handler.php` – реализует механизм обработки ошибок, который преобразует текст сообщения об ошибке в удобочитаемый формат.

Приложение `Friendly` через настраиваемые интервалы времени (по умолчанию 5 секунд) считывает две случайные записи из таблицы `users`, созданной в предыдущем упражнении, и два случайных числа с сервера, генератора случайных чисел, с которым мы также встречались ранее в этой главе. Основываясь на этих данных, сервер создает примерно такое сообщение: «Пользователь **paula** работает совместно с пользователем **emilian** над проектом #33», которое будет прочитано клиентом и выведено на экран, как показано на рис. 3.21.

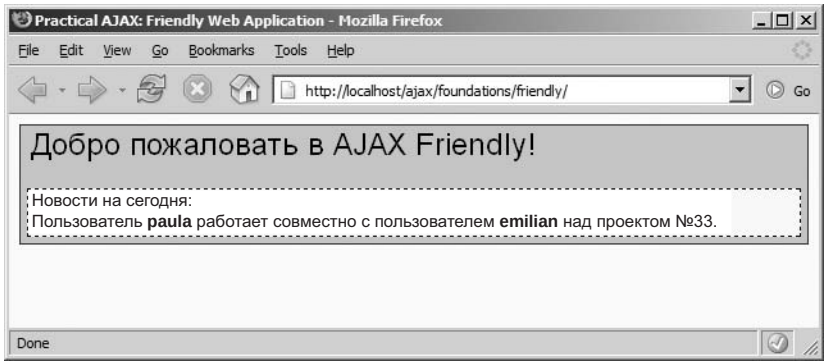

Рис. 3.21. *Веб-приложение Friendly*

При выполнении асинхронного запроса приложение выводит надпись «Чтение нового сообщения с сервера...» (это сообщение можно прочитать, потому что сервер делает искусственную задержку, имитирующую выполнение сложных и продолжительных действий).

Приложение может быть сконфигурировано так, чтобы выводить подробные сообщения об ошибках (что бывает удобно во время отладки), как показано на рис. 3.22, или более дружественные сообщения, как показано на рис. 3.23.

Итак, мы определили, что собираемся делать, и можем приступать...

Время действовать – приложение Friendly

1. В этом упражнении предполагается использовать таблицу `users`, которую мы создали ранее. Если вы еще не сделали этого, пожалуйста, выполните шаги 1 и 2 из предыдущего упражнения.
2. Создайте в каталоге `foundations` подкаталог с именем `friendly`.
3. Создайте файл с именем `index.html` и добавьте в него следующий код:

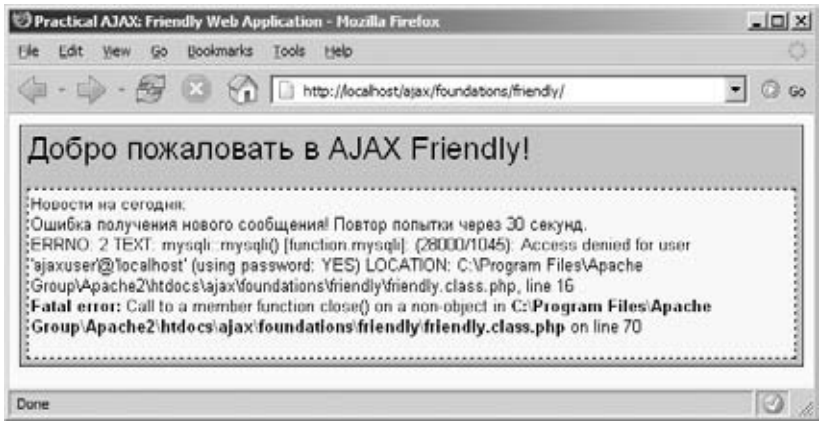

Рис. 3.22. *Если пароль доступа к базе данных утерян, выводится подробное сообщение об ошибке*

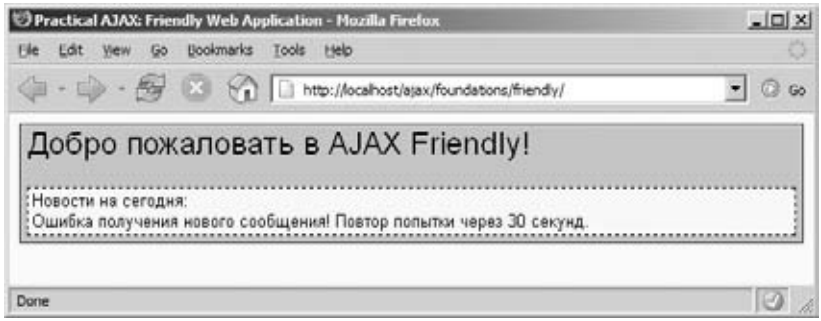

Рис. 3.23. *Более дружественное сообщение об ошибке*

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
    <head>
        <title>Практические примеры AJAX: Веб-приложение Friendly</title>
        <link href="friendly.css" rel="stylesheet" type="text/css"/>
        <script type="text/javascript" src="friendly.js"></script>
    </head>
    <body onload="process()">
        <noscript>
            <strong>
                Для работы этого примера требуется броузер
                с включенной поддержкой JavaScript!
                <br/><br/>
            </strong>
        </noscript>
        <div class="project">
            <span class="title">Добро пожаловать в AJAX Friendly!</span>
```

```
            <br/><br/>
            <div class="news">
                Новости на сегодня:
                <div id="myDivElement" />
            </div>
        </div>
    </body>
</html>
```

4. **Добавьте новый файл с именем** `friendly.css`**:**

```
body
{
    font-family: Arial, Helvetica, sans-serif;
    font-size: small;
    background-color: #fffccc;
}
input
{
    margin-bottom: 3px;
    border: #000099 1px solid;
}
.title
{
    font-size: x-large;
}
div.project
{
    background-color: #99ccff;
    padding: 5px;
    border: #000099 1px solid;
}
div.news
{
    background-color: #fffbb8;
    padding: 2px;
    border: 1px dashed;
}
```

5. **Теперь добавьте новый файл с исходным текстом на языке JavaScript –** `friendly.js`**:**

```
// переменная для хранения ссылки на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// переменная для хранения адреса удаленного сервера и параметров запроса
var serverAddress = "friendly.php?action=GetNews";
// переменные, которые определяют как часто выполняются обращения к серверу
var updateInterval = 5; // время ожидания перед запросом нового сообщения
var errorRetryInterval = 30; // время ожидания после ошибки сервера
// если установлено значение true, выводятся подробные
// сообщения об ошибках
var debugMode = true;
```

```
// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// эта функция выводит на страницу новое сообщение
function display($message)
{
    // получить ссылку на элемент <div>
    myDiv = document.getElementById("myDivElement");
    // вывести сообщение
    myDiv.innerHTML = $message + "<br/>";
}

// эта функция выводит сообщение об ошибке
function displayError($message)
```

```
{
    // вывести подробное сообщение, если debugMode = true
    display("Ошибка получения нового сообщения! Повтор попытки через " +
            errorRetryInterval + " секунд." +
            (debugMode ? "<br/>" + $message : ""));
    // перезапустить последовательность действий
    setTimeout("process();", errorRetryInterval * 1000);
}

// выполняет асинхронные обращения к серверу
function process()
{
    // продолжать только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // попытаться установить соединение с сервером
        try
        {
            // удалите эту строку, если вам не нравится сообщение
            // «Чтение нового...»
            display("Чтение нового сообщения с сервера...")
           // послать асинхронный запрос HTTP на получение нового сообщения
            xmlHttp.open("GET", serverAddress, true);
            xmlHttp.onreadystatechange = handleGettingNews;
            xmlHttp.send(null);
        }
        catch(e)
        {
            displayError(e.toString());
        }
    }
}

// эта функция вызывается при изменении состояния запроса HTTP
function handleGettingNews()
{
    // когда readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if (xmlHttp.status == 200)
        {
            try
            {
                // обработать ответ, полученный от сервера
                getNews();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                displayError(e.toString());
            }
```

```
        }
        else
        {
            // вывести сообщение об ошибке
            displayError(xmlHttp.statusText);
        }
    }
}

// обрабатывает ответ, полученный от сервера
function getNews()
{
    // прочитать сообщение, полученное от сервера
    var response = xmlHttp.responseText;
    // ошибка сервера?
    if (response.indexOf("ERRNO") >= 0
          || response.indexOf("error") >= 0
          || response.length == 0)
        throw(response.length == 0 ? "Server error." : response);
    // вывести сообщение
    display(response);
    // перезапустить последовательность действий
    setTimeout("process();", updateInterval * 1000);
}
```

6. **Перейдем к сценариям, работающим на стороне сервера. Начнем с файла** `friendly.php`**:**

```
<?php
// загрузить модуль обработки ошибок
require_once('error_handler.php');
require_once('friendly.class.php');
// не дать возможность клиентскому броузеру помещать результаты в кэш
header('Expires: Wed, 23 Dec 1980 00:30:00 GMT'); // время в прошлом
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
// прочитать тип операции
$action = $_GET['action'];
// получить новости
if ($action == 'GetNews')
{
    // создать новый экземпляр класса Friendly
    $friendly = new Friendly();
    // с его помощью получить новое сообщение
    $news = $friendly->getNews();
    // передать сообщение клиенту
    echo $news;
}
else
{
    echo 'Ошибка: сервер не понял команду.';
```

```
}
?>
```

7. **Создайте файл** `friendly.class.php` **и добавьте в него следующий код:**

```
<?php
// загрузить модуль обработки ошибок
require_once ('error_handler.php');
// загрузить конфигурацию
require_once ('config.php');

// класс Friendly, отвечающий за реализацию функциональности веб-приложения
class Friendly
{
    // соединение с базой данных
    private $mMysqli;

    // конструктор, открывает соединение с базой данных
    function __construct()
    {
        $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                    DB_DATABASE);
    }

    // генерирует новое сообщение
    public function getNews()
    {
        // строка сообщения
        $news = 'На сегодня новостей нет.';
        // предложение SQL, которое выбирает двух случайных
        // пользователей из базы данных
        $query = 'SELECT user_name FROM users ' .
                 'ORDER BY RAND() ' .
                 'LIMIT 2';
        // выполнить запрос
        $result = $this->mMysqli->query($query);
        // извлечь записи
        $row1 = $result->fetch_array(MYSQLI_ASSOC);
        $row2 = $result->fetch_array(MYSQLI_ASSOC);
        // закрыть входной поток
        $result->close();
        // создать сообщение
        if (!$row1 || !$row2)
        {
            $news = 'Для работы над проектом требуется большее
                     число пользователей!';
        }
        else
        {
            // создать сообщение в формате HTML
            $name1 = '<b>' . $row1['user_name'] . '</b>';
            $name2 = '<b>' . $row2['user_name'] . '</b>';
            $randNum = $this->getRandomNumber();
```

```
            $news = 'Пользователь ' . $name1 .
                    ', совместно с пользователем ' . $name2 .
                    ' работают над проектом #' . $randNum . '.';
        }
        // вернуть строку сообщения
        return $news;
    }

    // возвращает случайное число в диапазоне от 1 до 100
    private function getRandomNumber()
    {
        // задержка исполнения на четверть секунды
        usleep(250000);
        // адрес удаленного сервера и параметры запроса
        $serverAddress = 'http://www.random.org/cgi-bin/randnum';
        $serverParams = 'num=1&min=1&max=100';
        // получить случайное число с удаленного сервера
        $randomNumber = file_get_contents($serverAddress . '?' .
                                          $serverParams);
        // вернуть случайное число
        return trim($randomNumber);
    }

    // деструктор, закрывает соединение с базой данных
    function __destruct()
    {
        $this->mMysqli->close();
    }
}
?>
```

8. **Добавьте конфигурационный файл** `config.php`**:**

```
<?php
// сведения, необходимые для соединения с базой данных
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

9. **И наконец, добавьте сценарий обработки ошибок** `error_handler.php`**:**

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
```

```
                    'LOCATION: ' . $errFile .
                    ', line ' . $errLine;
        echo $error_message;
        // прервать дальнейшую работу сценария PHP
        exit;
    }
    ?>
```

10. Откройте страницу *http://localhost/ajax/foundations/friendly/*.

Что происходит внутри?

Большинство реализованных в приложении принципов уже были рассмотрены, поэтому мы лишь кратко проанализируем новые моменты. Начнем со стороны клиента. В файле `index.html` появился новый для нас элемент `<noscript>`. Он обеспечивает минимальную функциональность приложения в броузерах, которые не имеют поддержки JavaScript или в которых она отключена:

```
<body onload="process()">
    <noscript>
        <strong>
            Для работы этого примера требуется броузер
            с включенной поддержкой JavaScript!
            <br/><br/>
        </strong>
    </noscript>
```

Броузеры, в которых поддержка JavaScript включена, будут игнорировать все, что находится между тегами `<noscript>` и `</noscript>`. Остальные броузеры будут отображать заключенный в них код HTML.

В файле `friendly.js` имеется несколько неожиданных для нас моментов:

- Мы сгруппировали действия по обработке сообщений, отображаемых перед пользователем, в виде двух функций: `display` и `displayError`. Обе они в качестве входного параметра принимают сообщение, которое необходимо вывести, но функция `displayError` выводит сообщение только в том случае, если в переменной `debugMode` находится значение `true` (определение этой переменной находится в начале файла).
- Функция `displayError` вызывается из блока `catch` после перехвата исключения, возникшего где-либо, и обращается к функции `setTimeout` для повторного запуска последовательности действий, посылающих запросы серверу. Можно изменить продолжительность интервала от появления ошибки до передачи нового запроса, изменив значение переменной `errorRetryInterval`.
- Частота обращений к серверу за получением новых сообщений определяется значением переменной `updateInterval`.
- В функции `getNews()` реализован упрощенный механизм проверки на ошибку в ответе сервера. Этот механизм проверяет наличие слова «`ERRNO`» (генерируется обработчиком ошибок на стороне сервера)

или «error» (генерируется автоматически ядром PHP в случае фатальной ошибки или ошибки синтаксического анализа текста сценария) в тексте сообщения. Кроме того, ошибкой считается прием пустого сообщения (если параметр displayErrors в конфигурационном файле php.ini имеет значение off, то текст сообщения об ошибке не генерируется). В любом из этих случаев возбуждается исключение, которое будет перехвачено механизмом обработки ошибок, чтобы проинформировать пользователя о возникшей проблеме.

На стороне сервера все начинается со сценария friendly.php, который вызывается клиентом. Самая важная часть friendly.php – это создание нового экземпляра класса Friendly (определение класса находится в файле friendly.class.php) и вызов метода getNews():

```
// прочитать тип операции
$action = $_GET['action'];
// получить новости
if ($action == 'GetNews')
{
    // создать новый экземпляр класса Friendly
    $friendly = new Friendly();
    // с его помощью получить новое сообщение
    $news = $friendly->getNews();
    // передать сообщение клиенту
    echo $news;
}
```

Все самое интересное на стороне сервера заключено в файле friendly.class.php, вызываемом из friendly.php для выполнения фактических действий. В файле friendly.class.php находится определение класса Friendly, который состоит из следующих членов:

- $mMysqli – частное поле, которое хранит сведения о соединении с базой данных на протяжении всего времени жизни объекта.
- __construct() – конструктор класса. Он инициализирует переменную $mMysqli, открывая соединение с базой данных. Поскольку конструктор вызывается автоматически в момент создания экземпляра класса, можно с уверенностью считать, что соединение с базой данных установлено и доступно в любом из методов класса.
- __destruct() – деструктор класса. Он закрывает соединение с базой данных. Деструктор вызывается автоматически при уничтожении экземпляра класса.
- getRandomNumber() – этот защищенный (квалификатор private) вспомогательный метод возвращает случайное число. Защищенные методы не могут вызываться извне из программы, создавшей экземпляр класса, они предназначены исключительно для использования внутри самого класса. Код метода getRandomNumber уже знаком нам по предыдущим упражнениям – он обращается к серверу random.org за получением случайного числа. Функция PHP usleep предназначена

для создания искусственной задержки в четверть секунды, чтобы можно было подольше насладиться созерцанием сообщения «Чтение нового сообщения с сервера...».

- `getNews()` – это общедоступный (квалификатор public) метод, который может вызываться внешней программой для получения нового сообщения. Метод извлекает два случайных имени пользователей из базы данных, с помощью метода `getRandomNumber` получает случайное число и составляет примерно такое сообщение: «Пользователь *x* совместно с пользователем *y* работают над проектом #*z*». (Да, не слишком романтично, но мы не смогли придумать ничего более интересного – извините!) Обратите внимание на то, как используется специальный объект `$this` для доступа к `$mMysqli` и `getRandomNumber()`. К членам класса можно обратиться только через экземпляр класса, а в PHP `$this` представляет собой ссылку на текущий экземпляр класса.

Подведение итогов

Надеемся, что вы получили удовольствие, разбирая небольшие примеры из этой главы, потому что впереди нас ждет еще большее их количество! В этой главе были рассмотрены технологии, применяемые в типичных приложениях AJAX на стороне сервера. Мы разобрали несколько упражнений, наделенных простейшей функциональностью, и PHP прекрасно справился с задачей по их обеспечению. Кроме того, вы узнали об основных принципах взаимодействия с базами данных и познакомились с простейшими операциями на примере первой созданной в этой книге таблицы.

В последующих главах вам встретятся еще более интересные примеры, которые включают в себя более сложный код, реализующий их функциональные возможности. В главе 4 вы создадите страницу с поддержкой валидации заполнения форм, которая будет работать, даже если клиент не поддерживает JavaScript и AJAX.

4

Верификация заполнения форм в AJAX

Проверка правильности входных данных – одно из обязательных требований, предъявляемых к качеству и уровню безопасности прикладного ПО. В случае веб-приложений такая проверка еще важнее, поскольку приложение доступно широкому кругу пользователей, обладающих различными навыками и уровнем подготовки.

Проверка правильности входных данных – это не игрушки. Неверные входные данные могут представлять определенную угрозу функциональности приложения и даже вызвать разрушение самой чувствительной его части – базы данных.

Проверка правильности входных данных подразумевает проверку соответствия данных, введенных пользователем, заранее определенным правилам, которые устанавливаются приложением. Например, если дата должна вводиться в формате ГГГГ-ММ-ДД, то строка «Февраль 28» может рассматриваться как неправильная. Электронные адреса и номера телефонов – это еще один пример данных, подлежащих проверке на соответствие установленному формату.

Примечание

Тщательно описывайте правила проверки входных данных в виде отдельного документа и неукоснительно следуйте им при выполнении проверки входных данных.

Исторически сложилось так, что верификация заполнения формы производилась на стороне сервера уже после передачи формы. В некоторых случаях простейшие виды проверок выполнялись с помощью JavaScript на стороне клиента, например проверка адресов электронной почты или факт заполнения поля имени пользователя.

Традиционным методикам проверки правильности заполнения форм свойственны следующие проблемы:

- Методика проверки на стороне сервера усложняется ограничениями протокола HTTP, который не предоставляет информации о своем состоянии. Необходим специальный программный код, решающий эту проблему, иначе после передачи ошибочных данных пользователю будет возвращена пустая (незаполненная) форма.
- Отправив форму, пользователь вынужден ждать, пока страница полностью не будет загружена вновь. После обнаружения каждой ошибки страницу необходимо загружать вновь целиком.

В этой главе мы создадим приложение верификации заполнения формы, в котором реализуем старые добрые традиционные методики и добавим к ним возможности технологии AJAX, сделав тем самым форму более дружественной и более динамичной.

Даже если вы предусматриваете верификацию с применением технологии AJAX, проверка на стороне сервера все равно нужна, поскольку сервер – это последняя линия обороны в войне против ошибочных данных. Код JavaScript, передаваемый клиенту, может оказаться не только заблокированным в настройках броузера, но и легко может быть изменен или обойден.

Код всех примеров этой главы можно найти по адресу *http://ajax-php.packtpub.com*.

Реализация проверки правильности в AJAX

Приложение, которое мы создадим в этой главе, выполняет окончательную верификацию заполнения формы на стороне сервера после того, как она отправлена, а также реализует выполнение дополнительных проверок с применением технологии AJAX, пока пользователь путешествует по форме. Окончательная проверка на стороне сервера выполняется в соответствии с алгоритмом, блок-схема которого приведена на рис. 4.1.

Заключительная проверка правильности заполнения формы на стороне сервера обязательна всегда. Если кто-либо отключит поддержку JavaScript в своем броузере, то верификация заполнения с применением технологии AJAX просто не будет выполняться. Это даст возможность злоумышленнику повредить важные данные на сервере (например, внедрив код SQL).

Совет

Всегда проверяйте на сервере данные, получаемые от пользователя.

Приложение, которое мы создадим, будет проверять регистрационную форму, показанную на рис. 4.2, как на стороне клиента с применением технологии AJAX, так и на стороне сервера с помощью традиционных технологий:

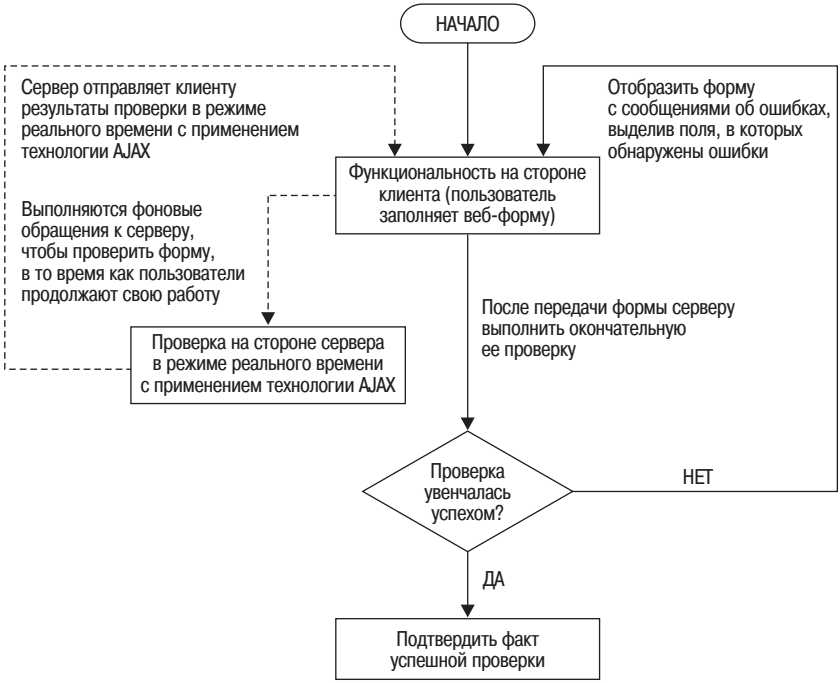

Рис. 4.1. *Алгоритм непрерывного выполнения проверки, которая не мешает пользователям продолжать свою деятельность*

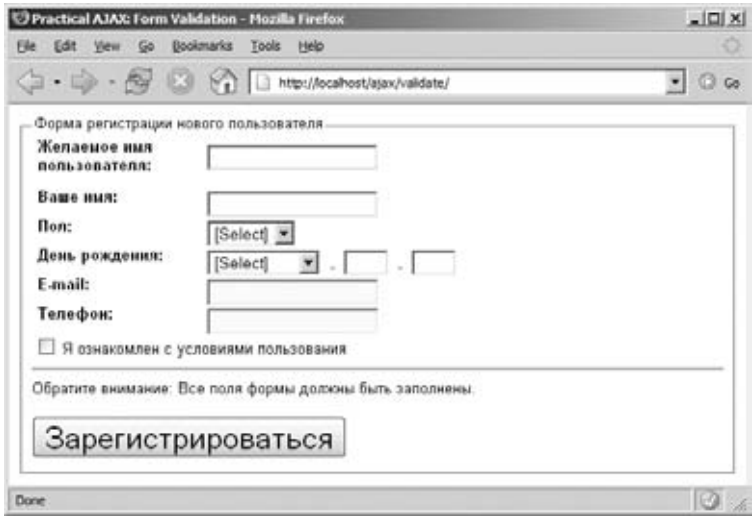

Рис. 4.2. *Форма регистрации пользователя*

- С помощью технологии AJAX – какое-либо поле формы теряет фокус ввода (событие `onblur`). Значение поля в этот момент передается серверу, который проверяет правильность данных и возвращает результат (0 – в случае ошибки, 1 – в случае успеха). Если значение не пройдет проверку, пользователь получит сообщение об ошибке (рис. 4.3).
- С помощью PHP – форма передается на сервер целиком. Это обычная операция проверки правильности заполнения всей формы в соответствии с определенными правилами. Если ошибки не обнаружены и входные данные признаны корректными, броузер перенаправляется на страницу, сообщающую об успешном завершении регистрации (рис. 4.4). Если проверка завершается неудачей, то пользователь возвращается на страницу с формой, в которой поля, содержащие ошибочные сведения, будут выделены цветом.

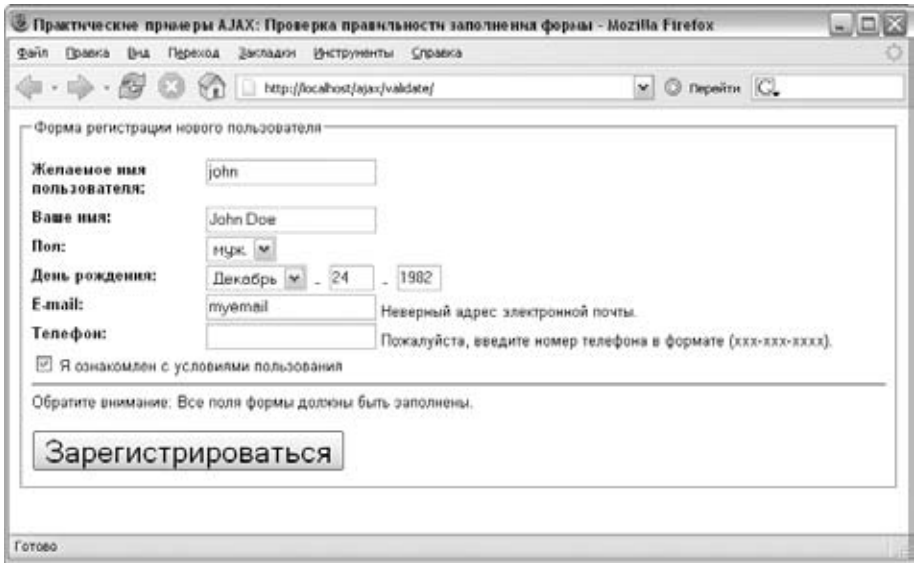

Рис. 4.3. *Функция проверки правильности заполнения в действии*

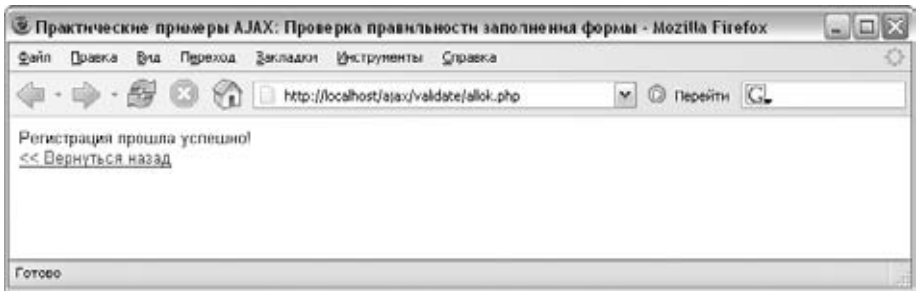

Рис. 4.4. *Успешное завершение регистрации*

Как на стороне клиента, так и на стороне сервера введенные данные проверяются по следующим правилам:

- Имя пользователя не должно содержаться в базе данных.
- Поле с именем не может быть пустым.
- Обязательно должен быть выбран пол.
- Обязательно должен быть выбран месяц рождения.
- День рождения должен быть допустимым числом (между 1 и 31).
- Год рождения должен быть допустимым числом (между 1900 и 2000).
- День месяца должен быть корректным (соответствовать номеру месяца).
- Адрес электронной почты должен быть записан в корректном формате, например *fillip@yahoo.co.uk* или *cristian@subdomain.domain.com*.
- Номер телефона должен быть записан в формате xxx-xxx-xxxx.
- Поле «Я ознакомился с условиями пользования» должно быть отмечено.

Работа приложения иллюстрируется на рис. 4.2–4.4.

Безопасность AJAX в многопоточной среде

Часть кода безопасна в многопоточной среде, если она корректно функционирует в случае одновременного обращения к ней в контексте нескольких потоков исполнения. Эта глава содержит первый пример, в котором внешний фактор – пользователь – непосредственно влияет на запросы AJAX. Для того чтобы проверить корректность введенных данных, мы должны отправлять асинхронные запросы всякий раз, когда то или иное поле теряет фокус ввода или когда пользователь выбирает значение из списка.

Такой подход таит в себе опасность, которая проявляется, только когда пользователь слишком быстро перемещается по полям ввода или когда соединение с сервером слишком медленное. В этих случаях веб-приложение может пытаться посылать новые запросы с помощью объекта `XMLHttpRequest`, который уже занят ожиданием ответа на предыдущий запрос (это приведет к появлению ошибки, и приложение перестанет работать должным образом).

В зависимости от конкретных обстоятельств идеальное решение может заключаться в том, чтобы:

- Создавать новый экземпляр объекта `XMLHttpRequest` для каждого сообщения, которое необходимо передавать серверу. Этот метод достаточно прост в реализации, но может привести к ухудшению производительности сервера, если одновременно будет послано слишком много запросов, и он не гарантирует, что ответы будут приходить в том же порядке, в каком посылались запросы.

- Ставить сообщения в очередь и передавать их одно за другим по мере освобождения объекта `XMLHttpRequest`. Запросы будут отправляться в порядке их поступления. Очереди особенно важны в тех приложениях, в которых важен порядок следования сообщений.
- Запланировать автоматическую отправку запросов через определенное время. Этот метод напоминает прием с очередями в том смысле, что одновременно не посылается более одного запроса, но он не гарантирует соблюдения порядка передачи запросов, как и порядка поступления ответов.
- Игнорировать сообщения.

В этой главе мы впервые выбираем реализацию на основе очередей. Когда поле теряет фокус ввода, соответствующий запрос на проверку правильности помещается в очередь. Когда объект `XMLHttpRequest` будет готов обслужить очередной запрос, ему будет передано первое сообщение из очереди.

Очередь строится по принципу **первый пришел – первый ушел (First-In, First-Out – FIFO)**. Это гарантирует, что запросы будут отправляться серверу в требуемом порядке. Чтобы получить представление о том, как все это работает, перейдите на демонстрационную страницу к этой главе, расположенную на сайте книги (или выполните упражнение), и быстро нажмите клавишу табуляции несколько раз подряд, подождите, и вы увидите, как одно за другим будут появляться сообщения на странице.

Обратите внимание, что задумываться над решением подобных проблем имеет смысл только в тех приложениях, где элементы могут вызвать несинхронизированные обращения к серверу. В противном случае, в таких приложениях, как Friendly из главы 3, где новые запросы инициируются только после получения ответа, нет смысла реализовывать код, безопасный в многопоточной среде.

Пришло время писать код.

Время действовать – проверка правильности заполнения формы с помощью AJAX

Совет

Если вы прочитали главу 3, то у вас уже должна иметься настроенная таблица `users`. Если это так, можете пропустить шаги 1 и 2.

1. Подключитесь к базе данных `ajax` и создайте таблицу `users` с помощью следующего SQL-выражения:

```
CREATE TABLE users
(
  user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  user_name VARCHAR(32) NOT NULL,
```

```
  PRIMARY KEY (user_id)
);
```

2. **Выполните следующий набор команд** `INSERT`**, которые заполнят таблицу** `users` **(поле** `user_id` **определено как автоинкрементное, поэтому оно будет заполняться значениями, генерируемыми самой базой данных):**

```
INSERT INTO users (user_name) VALUES ('bogdan');
INSERT INTO users (user_name) VALUES ('filip');
INSERT INTO users (user_name) VALUES ('mihai');
INSERT INTO users (user_name) VALUES ('emilian');
INSERT INTO users (user_name) VALUES ('paula');
INSERT INTO users (user_name) VALUES ('cristian');
```

3. **В каталоге** `ajax` **создайте подкаталог с именем** `validate`**.**
4. **Начнем с представления страницы. Создайте файл с именем** `validate.css` **и добавьте в него следующий код:**

```
body
{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 0.8em;
    color: #000000;
}

label
{
    float: left;
    width: 150px;
    font-weight: bold;
}

input, select
{
    margin-bottom: 3px;
}

.button
{
    font-size: 2em;
}

.left
{
    margin-left: 150px;
}

.txtFormLegend
{
    color: #777777;
    font-weight: bold;
    font-size: large;
}
```

```
.txtSmall
{
    color: #999999;
    font-size: smaller;
}

.hidden
{
    display: none;
}

.error
{
    display: block;
    margin-left: 150px;
    color: #ff0000;
}
```

5. **Теперь создайте новый файл с именем** `index_top.php` **и добавьте в него следующий код. Этот сценарий будет загружаться основной страницей приложения** `index.php`**.**

```
<?php
// запустить сессию PHP
session_start();
// создать теги HTML <option>
function buildOptions($options, $selectedOption)
{
    foreach ($options as $value => $text)
    {
        if ($value == $selectedOption)
        {
            echo '<option value="' . $value .
                 '" selected="selected">' . $text . '</option>';
        }
        else
        {
            echo '<option value="' . $value . '">' . $text . '</option>';
        }
    }
}

// инициализировать массив с полом
$genderOptions = array("0" => "[Выбрать]",
                       "1" => "Муж.",
                       "2" => "Жен.");
// инициализировать массив со списком месяцев
$monthOptions = array("0" => "[Выбрать]",
                      "1" => "Январь",
                      "2" => "Февраль",
                      "3" => "Март",
                      "4" => "Апрель",
```

```
                        "5" => "Май",
                        "6" => "Июнь",
                        "7" => "Июль",
                        "8" => "Август",
                        "9" => "Сентябрь",
                        "10" => "Октябрь",
                        "11" => "Ноябрь",
                        "12" => "Декабрь");
// инициализировать некоторые переменные сессии, чтобы предотвратить
// вывод замечаний PHP
if (!isset($_SESSION['values']))
{
    $_SESSION['values']['txtUsername'] = '';
    $_SESSION['values']['txtName'] = '';
    $_SESSION['values']['selGender'] = '';
    $_SESSION['values']['selBthMonth'] = '';
    $_SESSION['values']['txtBthDay'] = '';
    $_SESSION['values']['txtBthYear'] = '';
    $_SESSION['values']['txtEmail'] = '';
    $_SESSION['values']['txtPhone'] = '';
    $_SESSION['values']['chkReadTerms'] = '';
}
if (!isset($_SESSION['errors']))
{
    $_SESSION['errors']['txtUsername'] = 'hidden';
    $_SESSION['errors']['txtName'] = 'hidden';
    $_SESSION['errors']['selGender'] = 'hidden';
    $_SESSION['errors']['selBthMonth'] = 'hidden';
    $_SESSION['errors']['txtBthDay'] = 'hidden';
    $_SESSION['errors']['txtBthYear'] = 'hidden';
    $_SESSION['errors']['txtEmail'] = 'hidden';
    $_SESSION['errors']['txtPhone'] = 'hidden';
    $_SESSION['errors']['chkReadTerms'] = 'hidden';
}
?>
```

6. **Теперь создайте файл** `index.php` **и добавьте в него следующий код:**

```
<?php
require_once ('index_top.php');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>
          Практические примеры AJAX: Проверка правильности заполнения формы
        </title>
        <meta http-equiv="Content-Type" content="text/html;
                                        charset=utf-8" />
        <link href="validate.css" rel="stylesheet" type="text/css" />
        <script type="text/javascript" src="validate.js"></script>
```

```
</head>
<body onload="setFocus();">
    <fieldset>
      <legend class="txtFormLegend">New User Registration
                                                        Form</legend>
      <br />
      <form name="frmRegistration" method="post"
          action="validate.php?validationType=php">

          <!-- Имя пользователя -->
          <label for="txtUsername">Желаемое
                                  имя пользователя:</label>
          <input id="txtUsername" name="txtUsername" type="text"
              onblur="validate(this.value, this.id)"
            value="<?php echo $_SESSION['values']['txtUsername'] ?>" />
          <span id="txtUsernameFailed"
             class="<?php echo $_SESSION['errors']['txtUsername'] ?>">
              Имя пользователя уже используется
              или поле не заполнено.
          </span>
          <br />

          <!-- Имя -->
          <label for="txtName">Ваше имя:</label>
          <input id="txtName" name="txtName" type="text"
              onblur="validate(this.value, this.id)"
              value="<?php echo $_SESSION['values']['txtName'] ?>" />
          <span id="txtNameFailed"
              class="<?php echo $_SESSION['errors']['txtName'] ?>">
              Пожалуйста, введите ваше имя.
          </span>
          <br />

          <!-- Пол -->
          <label for="selGender">Пол:</label>
          <select name="selGender" id="selGender"
              onblur="validate(this.value, this.id)">
              <?php buildOptions($genderOptions,
                                 $_SESSION['values']['selGender']); ?>
          </select>
          <span id="selGenderFailed"
              class="<?php echo $_SESSION['errors']['selGender'] ?>">
              Пожалуйста, укажите ваш пол.
          </span>
          <br />

          <!-- День рождения -->
          <label for="selBthMonth">День рождения:</label>

          <!-- Месяц -->
          <select name="selBthMonth" id="selBthMonth"
              onblur="validate(this.value, this.id)">
              <?php buildOptions($monthOptions,
```

```
                                        $_SESSION['values']['selBthMonth']); ?>
                    </select>
                     - 

                    <!-- День -->
                    <input type="text" name="txtBthDay" id="txtBthDay"
                           maxlength="2" size="2"
                        onblur="validate(this.value, this.id)"
                       value="<?php echo $_SESSION['values']['txtBthDay'] ?>" />
                         - 

                    <!-- Год -->
                    <input type="text" name="txtBthYear" id="txtBthYear"
                           maxlength="4" size="2"
onblur="validate(document.getElementById('selBthMonth').options[docu-
ment.getElementById('selBthMonth').selectedIndex].value + '#' +
document.getElementById('txtBthDay').value + '#' + this.value, this.id)"
                      value="<?php echo $_SESSION['values']['txtBthYear'] ?>" />

                    <!-- Проверить месяц, день и год -->
                    <span id="selBthMonthFailed"
                       class="<?php echo $_SESSION['errors']['selBthMonth'] ?>">
                        Пожалуйста, укажите месяц вашего рождения.
                    </span>
                    <span id="txtBthDayFailed"
                        class="<?php echo $_SESSION['errors']['txtBthDay'] ?>">
                        Пожалуйста, введите день вашего рождения.
                    </span>
                    <span id="txtBthYearFailed"
                        class="<?php echo $_SESSION['errors']['txtBthYear'] ?>">
                        Пожалуйста, введите правильный год вашего рождения.
                    </span>
                    <br />

                    <!-- Электронный адрес -->
                    <label for="txtEmail">E-mail:</label>
                    <input id="txtEmail" name="txtEmail" type="text"
                        onblur="validate(this.value, this.id)"
                        value="<?php echo $_SESSION['values']['txtEmail'] ?>" />
                    <span id="txtEmailFailed"
                        class="<?php echo $_SESSION['errors']['txtEmail'] ?>">
                        Неверный адрес электронной почты.
                    </span>
                    <br />

                    <!-- Номер телефона -->
                    <label for="txtPhone">Телефон:</label>
                    <input id="txtPhone" name="txtPhone" type="text"
                        onblur="validate(this.value, this.id)"
                        value="<?php echo $_SESSION['values']['txtPhone'] ?>" />
                    <span id="txtPhoneFailed"
                        class="<?php echo $_SESSION['errors']['txtPhone'] ?>">
                      Пожалуйста, введите номер телефона в формате (xxx-xxx-xxxx).
```

```
                </span>
                <br />

                <!-- Ознакомлен с условиями -->
                <input type="checkbox" id="chkReadTerms" name="chkReadTerms"
                    class="left"
                    onblur="validate(this.checked, this.id)"
                    <?php if ($_SESSION['values']['chkReadTerms'] == 'on')
                        echo 'checked="checked"' ?> />
                    Я ознакомлен с условиями пользования
                <span id="chkReadTermsFailed"
                  class="<?php echo $_SESSION['errors']['chkReadTerms'] ?>">
                    Пожалуйста, ознакомьтесь с условиями пользования.
                </span>

                <!-- Конец формы -->
                <hr />
               <span class="txtSmall">Note: All fields are required.</span>
                <br /><br />
                <input type="submit" name="submitbutton"
                    value="Зарегистрироваться"
                    class="left button" />
            </form>
        </fieldset>
    </body>
</html>
```

7. **Создайте новый файл с именем** `allok.php` **и добавьте в него следующий код:**

```
<?php
// очистить данные сохраненные в сессии
session_start();
session_destroy();
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <title>
    Проверка правильности заполнения формы с использованием технологии AJAX
        </title>
       <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <link href="validate.css" rel="stylesheet" type="text/css" />
    </head>
    <body>
        Регистрация прошла успешно!<br />
        <a href="index.php" title="Go back">&lt;&lt; Go back</a>
    </body>
</html>
```

8. **Создайте новый файл с именем** `validate.js`**. Он реализует функциональность на стороне клиента, включая запросы AJAX:**

```
// переменная для хранения ссылки на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// переменная для хранения адреса удаленного сервера
var serverAddress = "validate.php";
// если установлено значение true, выводятся подробные сообщения об ошибках
var showErrors = true;
// инициализировать кэш запросов
var cache = new Array();

// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // переменная для хранения ссылки на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {} // игнорировать возможные ошибки
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// эта функция выводит сообщение об ошибке
function displayError($message)
```

```
{
    // игнорировать ошибку, если в showErrors находится значение false
    if (showErrors)
    {
        // отключить вывод сообщений об ошибках
        showErrors = false;
        // вывести сообщение об ошибке
        alert("Обнаружена ошибка: \n" + $message);
        // повторная проверка не ранее, чем через 10 секунд
        setTimeout("validate();", 10000);
    }
}

// эта функция выполняет проверку любого поля формы
function validate(inputValue, fieldID)
{
    // продолжать только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // если был принят не пустой аргумент, помещаем его в кэш в виде
        // строки запроса, который будет послан серверу для проверки
        if (fieldID)
        {
            // преобразовать значения, в форму, которая безопасно
            // может быть включена в строку запроса HTTP
            inputValue = encodeURIComponent(inputValue);
            fieldID = encodeURIComponent(fieldID);
            // добавить значения в очередь
            cache.push("inputValue=" + inputValue + "&fieldID=" + fieldID);
        }
        // попытаться установить соединение с сервером
        try
        {
            // продолжать только если объект XMLHttpRequest не занят
            // и кэш не пуст
            if ((xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
                    && cache.length > 0)
            {
                // извлечь новый набор параметров из кэша
                var cacheEntry = cache.shift();
                // послать запрос серверу для проверки извлеченных данных
                xmlHttp.open("POST", serverAddress, true);
                xmlHttp.setRequestHeader("Content-Type",
                                         "application/x-www-form-urlencoded");
                xmlHttp.onreadystatechange = handleRequestStateChange;
                xmlHttp.send(cacheEntry);
            }
        }
        catch (e)
        {
            // вывести сообщение об ошибке при неудачной попытке
```

```
            // установить соединение с сервером
            displayError(e.toString());
        }
    }
}

// эта функция обслуживает ответы HTTP
function handleRequestStateChange()
{
    // когда readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if (xmlHttp.status == 200)
        {
            try
            {
                // прочитать ответ, полученный от сервера
                readResponse();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                displayError(e.toString());
            }
        }
        else
        {
            // вывести сообщение об ошибке
            displayError(xmlHttp.statusText);
        }
    }
}

// читает ответ сервера
function readResponse()
{
    // получить ответ сервера
    var response = xmlHttp.responseText;
    // ошибка сервера?
    if (response.indexOf("ERRNO") >= 0
            || response.indexOf("error:") >= 0
            || response.length == 0)
        throw(response.length == 0 ? "Server error." : response);
    // получить ответ в формате XML (предполагается корректный формат XML)
    responseXml = xmlHttp.responseXML;
    // получить ссылку на корневой элемент
    xmlDoc = responseXml.documentElement;
    result = xmlDoc.getElementsByTagName("result")[0].firstChild.data;
    fieldID = xmlDoc.getElementsByTagName("fieldid")[0].firstChild.data;
    // отыскать элемент HTML, в котором следует вывести сообщение об ошибке
    message = document.getElementById(fieldID + "Failed");
```

```
        // показать или спрятать сообщение
        message.className = (result == "0") ? "error" : "hidden";
        // вызвать validate() еще раз, на случай, если кэш не пуст
        setTimeout("validate();", 500);
    }

    // устанавливает фокус ввода в первое поле формы
    function setFocus()
    {
        document.getElementById("txtUsername").focus();
    }
```

9. **Добавим в приложение бизнес-логику. Начнем с создания файла** `config.php`**:**

```
<?php
// сведения, необходимые для соединения с базой данных
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

10. **Теперь добавим код обработки ошибок в файл** `error_handler.php`**:**

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

11. **Обслуживанием запросов, поступающих от клиента, и выполнением проверки правильности заполнения формы будет заниматься сценарий** `validate.php`**:**

```
<?php
// начать сессию PHP
session_start();
// загрузить модуль обработки ошибок и класс, выполняющий проверку
require_once ('error_handler.php');
```

```
require_once ('validate.class.php');
// создать новый объект, выполняющий проверку
$validator = new Validate();
// прочитать тип проверки (PHP или AJAX?)
$validationType = '';
if (isset($_GET['validationType']))
{
    $validationType = $_GET['validationType'];
}
// тип проверки AJAX или PHP?
if ($validationType == 'php')
{
    // проверка типа PHP выполняется методом ValidatePHP, который
    // возвращает страницу, куда должен быть перенаправлен посетитель
    // (то есть allok.php, если все данные были введены корректно
    // или обратно, на index.php, если были обнаружены ошибки)
    header("Location:" . $validator->ValidatePHP());
}
else
{
    // Проверка типа AJAX выполняется методом ValidateAJAX.
    // Результаты проверки используются
    // для формирования документа XML,
    // который отправляется обратно клиенту
    $response =
        '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>' .
        '<response>' .
        '<result>' .
       $validator->ValidateAJAX($_POST['inputValue'], $_POST['fieldID']) .
        '</result>' .
        '<fieldid>' .
        $_POST['fieldID'] .
        '</fieldid>' .
        '</response>';
    // сгенерировать ответ
    if(ob_get_length()) ob_clean();
    header('Content-Type: text/xml');
    echo $response;
}
?>
```

12. **Класс, реализующий функции проверки, называется** `Validate` **и размещается в файле с именем** `validate.class.php`**:**

```
<?php
// загрузить модуль обработки ошибок и конфигурацию базы данных
require_once ('config.php');
// класс поддерживает проверку правильности типа AJAX и PHP
class Validate
{
    // соединение с базой данных
    private $mMysqli;
```

```
// конструктор, открывает соединение с базой данных
function __construct()
{
    $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                DB_DATABASE);
}

// деструктор, закрывает соединение с базой данных
function __destruct()
{
    $this->mMysqli->close();
}

// поддержка проверки правильности типа AJAX,
// проверяет единственное значение
public function ValidateAJAX($inputValue, $fieldID)
{
    // уточнить, какое поле будет проверяться и выполнить проверку
    switch($fieldID)
    {
        // проверить правильность имени пользователя
        case 'txtUsername':
            return $this->validateUserName($inputValue);
            break;
        // проверить правильность имени
        case 'txtName':
            return $this->validateName($inputValue);
            break;
        // проверить, выбран ли пол
        case 'selGender':
            return $this->validateGender($inputValue);
            break;
        // проверить корректность месяца
        case 'selBthMonth':
            return $this->validateBirthMonth($inputValue);
            break;
        // проверить, правильно указан день рождения
        case 'txtBthDay':
            return $this->validateBirthDay($inputValue);
            break;
        // проверить, правильно ли указан год рождения
        case 'txtBthYear':
            return $this->validateBirthYear($inputValue);
            break;
        // проверить правильность адреса электронной почты
        case 'txtEmail':
            return $this->validateEmail($inputValue);
            break;
        // проверить правильность номера телефона
        case 'txtPhone':
            return $this->validatePhone($inputValue);
            break;
```

```
            // проверить, поставлена ли галочка в поле
            // «Я ознакомлен с условиями пользования»
            case 'chkReadTerms':
                return $this->validateReadTerms($inputValue);
                break;
        }
    }

    // проверяет правильность заполнения всех полей формы
    public function ValidatePHP()
    {
        // признак ошибки, если будет найдена ошибка, в переменную
        // будет записано значение 1.
        $errorsExist = 0;
        // сбросить флаг errors сессии
        if (isset($_SESSION['errors']))
            unset($_SESSION['errors']);
        // по умолчанию все поля считаются правильными
        $_SESSION['errors']['txtUsername'] = 'hidden';
        $_SESSION['errors']['txtName'] = 'hidden';
        $_SESSION['errors']['selGender'] = 'hidden';
        $_SESSION['errors']['selBthMonth'] = 'hidden';
        $_SESSION['errors']['txtBthDay'] = 'hidden';
        $_SESSION['errors']['txtBthYear'] = 'hidden';
        $_SESSION['errors']['txtEmail'] = 'hidden';
        $_SESSION['errors']['txtPhone'] = 'hidden';
        $_SESSION['errors']['chkReadTerms'] = 'hidden';
        // проверить имя пользователя
        if (!$this->validateUserName($_POST['txtUsername']))
        {
            $_SESSION['errors']['txtUsername'] = 'error';
            $errorsExist = 1;
        }
        // проверить имя
        if (!$this->validateName($_POST['txtName']))
        {
            $_SESSION['errors']['txtName'] = 'error';
            $errorsExist = 1;
        }
        // проверить пол
        if (!$this->validateGender($_POST['selGender']))
        {
            $_SESSION['errors']['selGender'] = 'error';
            $errorsExist = 1;
        }
        // проверить месяц рождения
        if (!$this->validateBirthMonth($_POST['selBthMonth']))
        {
            $_SESSION['errors']['selBthMonth'] = 'error';
            $errorsExist = 1;
        }
```

```
        // проверить день рождения
        if (!$this->validateBirthDay($_POST['txtBthDay']))
        {
            $_SESSION['errors']['txtBthDay'] = 'error';
            $errorsExist = 1;
        }
        // проверить год рождения
        if (!$this->validateBirthYear($_POST['selBthMonth'] . '#' .
                $_POST['txtBthDay'] . '#' .
                $_POST['txtBthYear']))
        {
            $_SESSION['errors']['txtBthYear'] = 'error';
            $errorsExist = 1;
        }
        // проверить адрес электронной почты
        if (!$this->validateEmail($_POST['txtEmail']))
        {
            $_SESSION['errors']['txtEmail'] = 'error';
            $errorsExist = 1;
        }
        // проверить номер телефона
        if (!$this->validatePhone($_POST['txtPhone']))
        {
            $_SESSION['errors']['txtPhone'] = 'error';
            $errorsExist = 1;
        }
        // проверить знакомство с условиями
        if (!isset($_POST['chkReadTerms']) ||
                !$this->validateReadTerms($_POST['chkReadTerms']))
        {
            $_SESSION['errors']['chkReadTerms'] = 'error';
            $_SESSION['values']['chkReadTerms'] = '';
            $errorsExist = 1;
        }
        // если ошибок не обнаружено, отправить на заключительную страницу
        if ($errorsExist == 0)
        {
            return 'allok.php';
        }
        else
        {
            // если были найдены ошибки, сохранить данные,
            // введенные пользователем
            foreach ($_POST as $key => $value)
            {
                $_SESSION['values'][$key] = $_POST[$key];
            }
            return 'index.php';
        }
    }
```

```
// проверить имя пользователя
// (не должно быть пустым, и не должно быть зарегистрированным)
private function validateUserName($value)
{
    // отсечь лишние пробелы и экранировать специальные символы
    $value = $this->mMysqli->real_escape_string(trim($value));
    // поле не должно быть пустым
    if ($value == null)
        return 0; // не правильно
    // проверить наличие такого пользователя в базе данных
    $query = $this->mMysqli->query('SELECT user_name FROM users ' .
                                   'WHERE user_name="' . $value . '"');
    if ($this->mMysqli->affected_rows > 0)
        return '0'; // не правильно
    else
        return '1'; // правильно
}

// проверить имя
private function validateName($value)
{
    // отсечь лишние пробелы и экранировать специальные символы
    $value = trim($value);
    // поле не должно быть пустым
    if ($value)
        return 1; // правильно
    else
        return 0; // не правильно
}

// проверить пол
private function validateGender($value)
{
    // пользователь должен указать свой пол
    return ($value == '0') ? 0 : 1;
}

// проверить месяц рождения
private function validateBirthMonth($value)
{
    // номер месяца не должен быть пустым и находиться в диапазоне 1-12
    return ($value == '' || $value > 12 || $value < 1) ? 0 : 1;
}

// проверить день рождения
private function validateBirthDay($value)
{
    // номер дня не должен быть пустым и находиться в диапазоне 1-31
    return ($value == '' || $value > 31 || $value < 1) ? 0 : 1;
}

// поверить год рождения и всю дату в целом
private function validateBirthYear($value)
```

```
        {
            // корректный год должен находиться в диапазоне 1900-2000
            // получить полную дату в формате (mm#dd#yyyy)
            $date = explode('#', $value);
            // дата не может быть верной,
            // если в ней отсутствует день, месяц или год
            if (!$date[0]) return 0;
            if (!$date[1] || !is_numeric($date[1])) return 0;
            if (!$date[2] || !is_numeric($date[2])) return 0;
            // проверить дату
            return (checkdate($date[0], $date[1], $date[2])) ? 1 : 0;
        }

        // проверить адрес электронной почты
        private function validateEmail($value)
        {
           // корректные форматы адресов: *@*.*, *@*.*.*, *.*@*.*, *.*@*.*.*)
            return (!eregi('^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-
                    ]+)*(\.[a-z]{2,3})$', $value)) ? 0 : 1;
        }

        // проверить номер телефона
        private function validatePhone($value)
        {
            // корректный формат номера телефона: ###-###-####
           return (!eregi('^[0-9]{3}-*[0-9]{3}-*[0-9]{4}$', $value)) ? 0 : 1;
        }

        // проверить, ознакомился ли пользователь с условиями пользования
        private function validateReadTerms($value)
        {
            // правильное значение 'true'
            return ($value == 'true' || $value == 'on') ? 1 : 0;
        }
    }
    ?>
```

13. Проверьте приложение, открыв в броузере страницу *http://localhost/ajax/validate/index.php*.

Что происходит внутри?

Технология AJAX позволяет проверять правильность заполнения отдельных полей и своевременно информировать пользователя о появлении каких-либо ошибок. Но изюминка в том, что все эти проверки выполняются незаметно для пользователя! Такая проверка называется ненавязчивой верификацией заполнения формы.

Ненавязчивая верификация дополняется окончательной проверкой правильности всей формы, после того как она будет отправлена пользователем на сервер. На стороне сервера обе разновидности проверки поддерживаются сценарием PHP `validate.php` с помощью еще одного сценария – `validate.class.php`.

Начнем исследование кода со сценария index.php, который обслуживает верификацию, выполняемую на стороне клиента. В этом упражнении основная страница представлена не простым файлом HTML, а сценарием PHP, который динамически генерирует некоторые части страницы. Сделано это для того, чтобы сохранить значения, введенные пользователем, на случай, если форма не пройдет проверку на стороне сервера. В такой ситуации без помощи этого кода PHP при повторной загрузке основной страницы перед пользователем выводилась бы пустая форма.

Сценарий index.php начинается с загрузки вспомогательного сценария index_top.php. Он открывает сессию вызовом функции session_start(), определяет некоторые переменные и функции, которые будут использованы позднее в index.php, и инициализирует некоторые переменные сессии ($_SESSION['values'] и $_SESSION['errors']), чтобы избежать появления сообщений PHP о том, что переменные не инициализированы.

Обратите внимание на событие onload в теге body файла index.php. По этому событию вызывается функция setFocus(), которая определена в файле validate.js. Эта функция устанавливает фокус ввода на первое поле в форме.

Далее в файле index.php вы увидите такую последовательность строк кода, повторяющуюся с небольшими изменениями:

```
<!-- Имя пользователя -->
<label for="txtUsername">Желаемое имя пользователя:</label>
<input id="txtUsername" name="txtUsername" type="text"
       onblur="validate(this.value, this.id)"
       value="<?php echo $_SESSION['values']['txtUsername'] ?>" />
<span id="txtUsernameFailed"
      class="<?php echo $_SESSION['errors']['txtUsername'] ?>">
    Имя пользователя уже используется или поле не заполнено.
</span>
<br />
```

Этот код отображает поле ввода на форме с текстовой меткой и текстом сообщения об ошибке под ним, которое отображается только в том случае, если верификация заполнения формы закончилась неудачей.

Примечание

В этом примере мы выводим текст сообщения об ошибке непосредственно под полем ввода, но вы можете изменить позицию и формат вывода в файле validate.css, изменив свойства класса CSS - error.

Элементы типа input генерируют событие onblur, когда теряют фокус ввода. По этому событию вызывается функция JavaScript – validate(), которой передаются два аргумента – значение поля и идентификатор поля. Эта функция реализует алгоритм проверки с применением технологии AJAX, посылая асинхронные запросы сценарию validate.php.

Сценарию на стороне сервера необходимо знать, какое поле проверяется и его текущее значение.

Атрибут `value` будет пустым при начальной загрузке страницы, но после того как форма будет отправлена, в нем должно сохраниться введенное значение на тот случай, если она не пройдет проверку и ее придется загрузить еще раз. Чтобы сохранить данные, введенные пользователем, мы организуем переменные сессии.

Следующий далее элемент `span` содержит текст сообщения, которое будет выведено в случае неудачной проверки. Этот элемент изначально невидим благодаря принадлежности к классу CSS `hidden`, но в случае появления ошибки мы меняем принадлежность к классу, подставляя имя класса `error`.

Функция `validate` в сценарии `validate.js` отправляет запрос серверу, вызывая сценарий `validate.php` и передавая ему два параметра – идентификатор поля и его значение.

Не забывайте, что объект `XMLHttpRequest` не может выполнять два запроса HTTP одновременно, поэтому, если объект занят обработкой предыдущего запроса, мы сохраняем данные текущего запроса в очереди для последующего использования. Это особенно удобно, если соединение с сетью или с Интернетом слишком медленное. Для хранения данных запросов нам необходима система кэширования, которая реализует структуру очереди FIFO. К счастью, в JavaScript имеется класс `Array`, обладающий всей необходимой нам функциональностью (его методы `push` и `shift`), поэтому мы отводим ему роль кэша:

```
var cache = new Array();
```

Таким образом, функция `validate()` начинается с добавления в кэш данных, требующих проверки (если они были переданы функции).

```
function validate(inputValue, fieldID)
{
    // продолжать, только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // если был принят не пустой аргумент, помещаем его в кэш
        // в виде строки запроса, который будет послан серверу для проверки
        if (fieldID)
        {
            // преобразовать значения, в форму, которая безопасно может
            // быть включена в строку запроса HTTP
            inputValue = encodeURIComponent(inputValue);
            fieldID = encodeURIComponent(fieldID);
            // добавить значения в очередь
            cache.push("inputValue=" + inputValue + "&fieldID=" + fieldID);
        }
```

Она добавляет новый элемент в конец массива кэша. Записи в кэше состоят из двух частей, значения и идентификатора поля, подлежащего

проверке, разделенных символом «&». Обратите внимание: новый элемент добавляется только в том случае, если значение `fieldID` не пустое. Аргумент `fieldID` будет пустым, только если функция вызывается лишь для проверки наличия в кэше запросов, ожидающих обработки.

Примечание

Идентификатор поля и его значение, извлекаемые из кэша, будут переданы серверу для проверки. Чтобы гарантировать их доставку к месту назначения в неизмененном виде, мы экранируем служебные символы с помощью функции `encodeURIComponent`. Это позволяет безопасно передавать любые символы серверу, в том числе и «&», которые в противном случае могут вызвать определенные проблемы. За дополнительной информацией по теме функций экранирования служебных символов в JavaScript обращайтесь к замечательной статье по адресу *http://xkr.us/articles/javascript/encode-compare/*.

Если объект `XMLHttpRequest` свободен и может быть использован для передачи очередного запросы, то мы с помощью функции `shift()` извлекаем из кэша очередное значение (эта функция одновременно удаляет запись из массива, что вполне отвечает нашим потребностям). Обратите внимание, что это значение может оказаться не тем, которое только что было помещено в кэш с помощью функции `push()`, – в случае очередей FIFO первой извлекается самая старая запись.

```
// попытаться установить соединение с сервером
try
{
    // продолжать, только если объект XMLHttpRequest
    // не занят и кэш не пуст
    if ((xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
            && cache.length > 0)
    {
        // извлечь новый набор параметров из кэша
        var cacheEntry = cache.shift();
```

Если объект `XMLHttpRequest` имеет статус 0 или 4, это говорит об отсутствии активных запросов, и у нас имеется возможность отправить новый запрос. Чтобы сформировать новый запрос, мы используем данные, извлеченные из кэша, которые представляют собой уже сформированную строку запроса:

```
        // послать запрос серверу для проверки извлеченных данных
        xmlHttp.open("POST", serverAddress, true);
        xmlHttp.setRequestHeader("Content-Type",
                                 "application/x-www-form-urlencoded");
        xmlHttp.onreadystatechange = handleRequestStateChange;
        xmlHttp.send(cacheEntry);
    }
```

Обработкой ответа сервера будет заниматься функция `handleRequestStateChange`. Она, в свою очередь, вызывает функцию `readResponse()` по-

сле того, как ответ сервера будет принят целиком. Эта функция сначала проверяет, не является ли ответ сервера сообщением об ошибке:

```
// читает ответ сервера
function readResponse()
{
    // получить ответ сервера
    var response = xmlHttp.responseText;
    // ошибка сервера?
    if (response.indexOf("ERRNO") >= 0
            || response.indexOf("error:") >= 0
            || response.length == 0)
        throw(response.length == 0 ? "Server error." : response);
```

После верификации мы считываем ответ сервера, который сообщит нам, верное ли значение находится в поле или нет:

```
    // получить ответ в формате XML (предполагается корректный формат XML)
    responseXml = xmlHttp.responseXML;
    // получить ссылку на корневой элемент
    xmlDoc = responseXml.documentElement;
    result = xmlDoc.getElementsByTagName("result")[0].firstChild.data;
    fieldID = xmlDoc.getElementsByTagName("fieldid")[0].firstChild.data;
```

В зависимости от полученного результата мы изменяем принадлежность элемента к классу CSS, подставляя имя класса `hidden` (если проверка прошла успешно) или `error` (если проверка потерпела неудачу). Принадлежность к классу CSS изменяется с помощью свойства `className`.

```
    // отыскать элемент HTML, в котором следует вывести сообщение об ошибке
    message = document.getElementById(fieldID + "Failed");
    // показать или спрятать сообщение
    message.className = (result == "0") ? "error" : "hidden";
    // вызвать validate() еще раз, на случай, если кэш не пуст
    setTimeout("validate();", 500);
}
```

На стороне сервера проверку выполняет сценарий `validate.php`. Он начинается с загрузки модуля обработки ошибок (`error_handler.php`) и класса `Validate`, который собственно и выполняет проверку (`validate.class.php`). Затем он отыскивает в массиве `GET` переменную с именем `validationType`. Эта переменная существует, только если вся форма целиком была отправлена на сервер, поскольку атрибут формы `action` определен как `validate.php?validationType=php`.

```
$validationType = '';
if (isset($_GET['validationType']))
{
    $validationType = $_GET['validationType'];
}
```

Затем, в зависимости от содержимого переменной $validationType, мы выполняем проверку либо одного поля, либо всей формы сразу.

```
// тип проверки AJAX или PHP?
if ($validationType == 'php')
{
    // проверка типа PHP выполняется методом ValidatePHP, который
    // возвращает страницу, куда должен быть перенаправлен посетитель
    // (то есть allok.php, если все данные были введены корректно
    // или обратно, на index.php, если были обнаружены ошибки)
    header("Location:" . $validator->ValidatePHP());
}
else
{
    // Проверка типа AJAX выполняется методом ValidateAJAX.
    // Результаты проверки используются для формирования документа XML,
    // который отправляется обратно клиенту
    $response =
        '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>' .
        '<response>' .
        '<result>' .
        $validator->ValidateAJAX($_POST['inputValue'], $_POST['fieldID']) .
        '</result>' .
        '<fieldid>' .
        $_POST['fieldID'] .
        '</fieldid>' .
        '</response>';
    // сгенерировать ответ
    if(ob_get_length()) ob_clean();
    header('Content-Type: text/xml');
    echo $response;
}
?>
```

Если мы имеем дело с вариантом классической проверки всей формы на стороне сервера, то вызываем метод validatePHP(), который возвращает имя страницы, куда должен быть перенаправлен броузер (в случае успешной проверки это будет страница allok.php, а в случае ошибки – index.php). Результаты проверки каждого поля сохраняются в переменных сессии, а если потребуется повторно передать форму пользователю, то index.php будет содержать в полях ранее введенные значения.

В случае проверки единственного поля сервер конструирует ответ, который определяет, допустимая ли информация содержится в поле. Ответ представляет собой короткий документ XML, который выглядит так:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response>
    <result>0</result>
    <fieldid>txtUsername</fieldid>
</response>
```

Если результат равен 0, поле txtUsername расценивается как неправильное и должно быть помечено соответствующим образом. Если результат равен 1, содержимое поля считается правильным.

Теперь перейдем к файлу validate.class.php. Конструктор класса открывает соединение с базой данных, а деструктор закрывает его. Затем идут два общедоступных метода: ValidateAJAX (выполняет проверку одного поля) и ValidatePHP (выполняет проверку всей формы).

Для того чтобы проверить правильность единственного поля, необходимо знать два параметра – проверяемое значение ($inputValue) и идентификатор поля ($fieldID). В блоке switch проверяются отдельные поля формы. Эта функция возвращает 0, если проверка потерпела неудачу, и 1, если проверка увенчалась успехом.

Функция проверки правильности всей формы не имеет входных аргументов, т. к. она всегда проверяет всю форму целиком (после того, как форма будет отправлена пользователем). Прежде всего мы инициализируем флаг $errorExist значением 0. Всякий раз, когда проверка очередного поля терпит неудачу, в этот флаг записывается значение 1. В результате по окончании работы функции мы знаем, что проверка формы потерпела неудачу. Затем мы должны удостовериться, что старые переменные сессии сброшены, чтобы гарантировать, что старые ошибки будут очищены.

После этого мы проверяем каждое поле формы в соответствии с заданными нами правилами. Если проверка потерпела неудачу, мы взводим флаг ($errorExist = 1) и записываем значение error в переменную сессии, которая хранит имя класса CSS для сообщения об ошибке. Если в конце функции флаг $errorExist равен 0, значит, проверка формы прошла успешно и мы можем вернуть имя страницы, сообщающей об успешной регистрации нового пользователя.

Если будет обнаружена хотя бы одна ошибка, мы записываем введенные пользователем данные в переменные сессии, основываясь на которых, сценарий index.php затем заполнит форму (не забывайте, что по умолчанию при начальной загрузке страницы все поля пустые). Текущее состояние сохраняется так:

```
foreach ($_POST as $key => $value)
{
    $_SESSION['values'][$key] = $_POST[$key];
}
```

где $_POST – это массив, хранящий имена и значения элементов формы, и мы можем обойти его с помощью цикла foreach. В этом фрагменте мы для каждого элемента в массиве $_POST создаем новый элемент в массиве $_SESSION['values'].

В файле validate.css для нас нет ничего примечательного, что стоило бы отдельного упоминания. Страница allok.php достаточно проста –

она лишь отображает сообщение, подтверждающее благополучное завершение процедуры регистрации.

Подведение итогов

Мы пока не утверждаем, что в совершенстве овладели техникой проверки правильности заполнения формы, мы предоставили лишь рабочую модель, доказывающую основные положения, работающее приложение, которое заботится о данных, введенных пользователем, и гарантирует их корректность.

Нельзя убедиться в правильности заполнения отдельных полей, пользуясь только средствами JavaScript.

Причина, по которой мы выбрали технологию AJAX для верификации заполнения формы, заключается в том, что во многих случаях такая проверка возможна только с привлечением базы данных (как, например, имя пользователя в данном случае). Кроме того, более профессиональным считается отделение всей бизнес-логики приложения (включая и различные проверки) и хранение ее в одном месте на сервере.

Технология AJAX может быть очень удобной, вам так не кажется?

5

Чат AJAX

В современном мире общение между людьми имеет очень большое значение. При этом есть реальная потребность в ускорении этого процесса. Электронная почта, обычные письма, SMS и чат дают людям возможность обмениваться сообщениями. На первое место здесь выходит фактор скорости. Электронная и обычная почта не в состоянии обеспечить высокую скорость получения ответа от собеседника, которую дают сообщения SMS или чат. В этой главе, опираясь на технологию AJAX, мы создадим приложение, которое даст возможность прямого общения по сети (реализующее чат).

Введение в технологию прямого общения по сети

Большинство коммуникаций, происходящих между компьютерами, производятся обычными приложениями.[1] Эти приложения общаются между собой напрямую, обходясь без промежуточных звеньев. Однако такое решение может оказаться неприемлемым для тех, кто работает в компании, политика безопасности которой разрешает пользователям открывать соединения только с портом 80[2] (HTTP). В этом случае можно столкнуться с реальными трудностями.

---

[1] Термин «обычные» (в оригинале desktop applications) здесь предполагает, что это прямые коммуникационные программы, аналогичные большинству других программ типового настольного компьютера. В противовес им автор выделяет класс приложений, взаимодействующих через протокол HTTP как промежуточный транспортный уровень. – *Примеч. науч. ред.*

[2] Здесь и далее речь идет о порте 80 протокола TCP, являющегося транспортным уровнем для протокола HTTP; это широко употребляемое значение порта для HTTP-служб, но используются и другие порты: 8080, 3128, ... – *Примеч. науч. ред.*

Известно достаточно много веб-приложений, предоставляющих возможность аудиовизуального общения по сети. Большинство из них реализованы на Java-апплетах. Апплеты широко прославились тем, что небезупречны в смысле безопасности, а кроме того, они вообще не всегда используют порт 80 для коммуникации. Поэтому они не могут рассматриваться как приемлемое решение, если необходимо организовать общение с корреспондентами за пределами компании.

Решение можно найти, обратившись к технологии AJAX, способной решить эту проблему. Приложив некоторые усилия, можно даже интегрировать в броузер клиент **IRC (Internet Relay Chat)**, систему обмена мгновенными сообщениями, или разработать свое собственное приложение прямого общения по сети, подобное тому, которое мы будем создавать чуть позже.

Вам, наверное, уже надоело слышать, что нельзя установить или использовать свой любимый мессенджер (инструментарий обмена сообщениями) на работе или в интернет-кафе? Надо полагать, вы уже оказывались в такой ситуации. Посмотрим, как же технология AJAX помогает справиться с этой неприятностью.

Чат–приложение, основанное на технологии AJAX

Вероятно, самое сильное впечатление сегодня производит решение, представленное на *www.meebo.com*.[1] Вы могли уже слышать о нем, а если не слышали, то сейчас самое время познакомиться с ним поближе. Первая и сама важная отличительная особенность этого решения в том, что оно позволяет войти в систему обмена мгновенными сообщениями при помощи средств только веб-интерфейса. На рис. 5.1 представлена страница входа в систему Meebo.

Сайт Meebo обеспечивает доступ ко всем этим службам из единственной начальной страницы, оснащенной дружественным пользовательским интерфейсом, всплывающими окнами, Java-апплетами и т. п.[2] Решения на базе технологии AJAX помогут вам забыть обо всех упомянутых выше неприятностях.

Meebo не единственное веб-приложение, предоставляющее возможность прямого общения по сети. Технология AJAX появилась совсем недавно, но уже можно найти массу приложений, предназначенных для организации быстрого обмена сообщениями, и даже целые решения на ее основе:

- *http://www.plasticshore.com/projects/chat/index.html*

[1] Русскоязычная страница: *http://www3.meebo.com/index-ru.html*. – *Примеч. науч. ред.*

[2] Однако система Meebo работает не со всеми броузерами, например с Opera 8.5; особенности поведения броузеров автор достаточно подробно описывал выше. – *Примеч. науч. ред.*

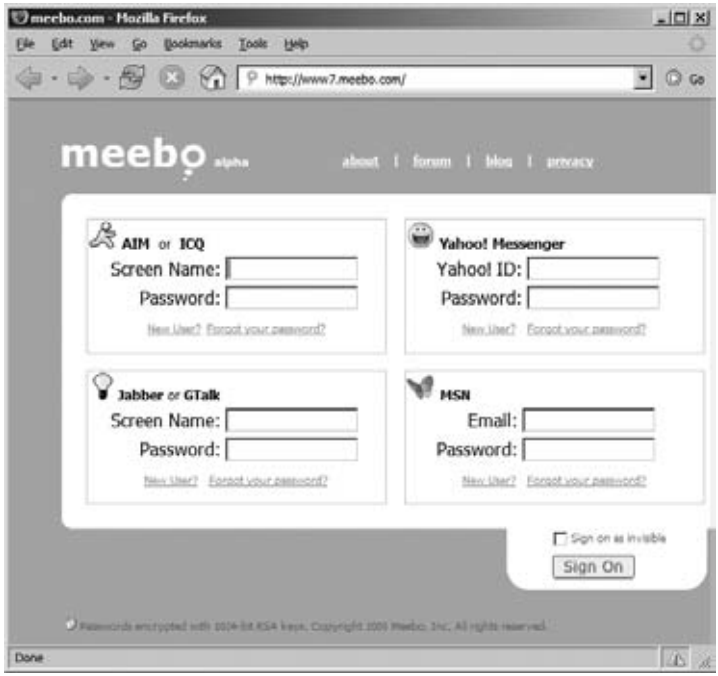

Рис. 5.1. *Meebo*

- *http://treehouse.ofb.net/chat/?lang=en*
- *http://www.chategory.org*
- *http://www.socket7.net/lace/*
- *http://drupal.org/node/27689*

Итак, настало время засучить рукава. На следующих страницах этой главы мы реализуем наше собственное приложение, которое позволит нам напрямую общаться друг с другом по сети.

Реализация чата на основе технологии AJAX

Мы постараемся сделать наше приложение простым, модульным и легко расширяемым. Для этого нам потребуется реализовать модуль входа в систему, комнаты чата, список пользователей и прочее. Чтобы не усложнять приложение, мы постараемся сосредоточиться на главной его цели – создании чата на базе AJAX. Мы реализуем самые основные функции чата: передачу и прием сообщений без необходимости повторной загрузки страницы. Мы также дадим пользователям возможность выбирать цвет, которым будут отображаться их сообщения, поскольку для этого также потребуется привлекать технологию AJAX, это будет еще одним хорошим примером ее применения.

Начав с этого приложения, мы без труда расширим его функциональность, добавляя модули, описанные в предыдущих главах, но в этой главе их не будет. Считайте, что это домашнее задание для тех из вас, кому стало интересно.

Примечание

Для того чтобы сделать этот пример работоспособным, потребуется **библиотека GD**. Инструкции по установке приведены в приложении А и включают поддержку библиотеки GD.

Ознакомиться с функциональностью приложения можно на сайте книги по адресу *http://ajaxphp.packtpub.com/ajax/chat/*. Внешний вид приложения приводится на рис. 5.2.

В этой главе мы впервые будем работать с двумя объектами `XMLHttpRequest`. Первый будет заниматься обновлением окна приложения, а второй – обслуживанием окна выбора цвета (после щелчка мышью по изображению координаты указателя мыши отправляются на сервер, а сервер возвращает код цвета).

Отправляемые сообщения будут сохраняться в очереди (организованной по принципу FIFO), точно так же как мы делали это в главе 4. Благодаря этому сообщения не теряются даже при очень высокой нагрузке на сервер и всегда попадают на сервер в том же порядке, в каком были отправлены. В отличие от других решений, которые можно найти в Интернете, мы не будем загружать сервер новыми сообщениями, пока не будет передано текущее.

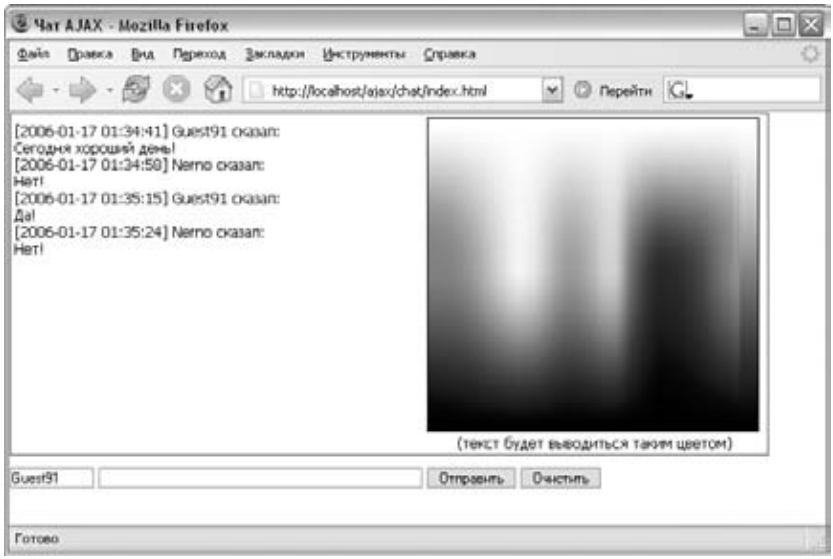

***Рис. 5.2.** Чат на базе AJAX*

Время действовать – чат на основе технологии AJAX

1. Соединитесь с базой данных `ajax` и создайте таблицу `chat` с помощью следующего SQL оператора:

```
CREATE TABLE chat
(
    chat_id int(11) NOT NULL auto_increment,
    posted_on datetime NOT NULL,
    user_name varchar(255) NOT NULL,
    message text NOT NULL,
    color char(7) default '#000000',
    PRIMARY KEY (chat_id)
);
```

2. В каталоге `ajax` создайте подкаталог `chat`.
3. Скопируйте файл[1] `palette.png` (его можно скачать с сайта книги в каталог `chat`).
4. Начнем реализацию приложения со сценариев, работающих на стороне сервера. В каталоге `chat` создайте файл с именем `config.php` и добавьте в него код конфигурации базы данных (измените значения в соответствии с вашей конфигурацией):

```
<?php
// сведения, необходимые для соединения с базой данных
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

5. Теперь добавьте стандартный модуль обработки ошибок `error_handler.php`:

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
```

[1] Здесь не совсем очевидная инструкция; имеется в виду сохранить именно рисунок со страницы сайта в локальный файл с указанным именем. URL этого рисунка: *http://ajaxphp.packtpub.com/ajax/chat/palette.png*. – *Примеч. науч. ред.*

```
                    ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

6. **Создайте файл с именем** `chat.php` **и добавьте в него следующий код:**

```
<?php
// загрузить файл с определением класса Chat
require_once("chat.class.php");
// получить тип выполняемой операции
$mode = $_POST['mode'];
// по умолчанию последний id = 0
$id = 0;
// создать новый экземпляр класса Chat
$chat = new Chat();
// если запрошена операция SendAndRetrieve
if($mode == 'SendAndRetrieveNew')
{
    // получить параметры операции,
    // используемые для добавления нового сообщения
    $name = $_POST['name'];
    $message = $_POST['message'];
    $color = $_POST['color'];
    $id = $_POST['id'];
    // проверить корректность параметров
    if ($name != '' && $message != '' && $color != '')
    {
        // записать сообщение в базу данных
        $chat->postMessage($name, $message, $color);
    }
}
// если запрошена операция DeleteAndRetrieve
elseif($mode == 'DeleteAndRetrieveNew')
{
    // удалить все существующие сообщения
    $chat->deleteMessages();
}
// если запрошена операция Retrieve
elseif($mode == 'RetrieveNew')
{
    // взять идентификатор последнего сообщения, полученного клиентом
    $id = $_POST['id'];
}
// очистить выходной буфер
if(ob_get_length()) ob_clean();
// предотвратить возможность кэширования страницы броузером
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT');
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
```

```
header('Pragma: no-cache');
header('Content-Type: text/xml');
// отправить с сервера новые сообщения
echo $chat->retrieveNewMessages($id);
?>
```

7. **Создайте файл с именем** `chat.class.php` **и добавьте в него следующий код:**

```
<?php
// загрузить конфигурационный файл
require_once('config.php');
// загрузить модуль обработки ошибок
require_once('error_handler.php');

// этот класс содержит реализацию серверной
// функциональности приложения Chat
{
    // соединение с базой данных
    private $mMysqli;

    // конструктор, открывает соединение с базой данных
    function __construct()
    {
        // установить соединение с базой данных
        $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                    DB_DATABASE);
    }

    // деструктор, закрывает соединение с базой данных
    public function __destruct()
    {
        $this->mMysqli->close();
    }

    // очищает таблицу, содержащую сообщения
    public function deleteMessages()
    {
        // собрать строку SQL-запроса, который очистит таблицу
        $query = 'TRUNCATE TABLE chat';
        // выполнить SQL-запрос
        $result = $this->mMysqli->query($query);
    }

    /*
        Метод postMessages записывает сообщение в базу данных
        - $name - имя пользователя, пославшего сообщение
        - $messsage - текст сообщения
        - $color - цвет, выбранный пользователем
    */

    public function postMessage($name, $message, $color)
    {
        // экранировать служебные символы для пущей безопасности
        // прежде чем передавать данные в базу данных
```

```
    $name = $this->mMysqli->real_escape_string($name);
    $message = $this->mMysqli->real_escape_string($message);
    $color = $this->mMysqli->real_escape_string($color);
    // собрать SQL-запрос, который добавит новое сообщение
    $query = 'INSERT INTO chat(posted_on, user_name, message, color) ' .
             'VALUES (NOW(), "' . $name . '" , "' . $message .
             '","' . $color . '")';
    // выполнить запрос
    $result = $this->mMysqli->query($query);
}

/*
    Метод retrieveNewMessages извлекает новые сообщения,
    которые были посланы на сервер.
    - $id – параметр, переданный клиентом, это идентификатор
    последнего сообщения, полученного клиентом.
    Сообщения, с идентификаторами большими, чем $id будут извлечены
    из базы данных и возвращены клиенту в формате XML.
*/
public function retrieveNewMessages($id=0)
{
    // экранировать служебные символы
    $id = $this->mMysqli->real_escape_string($id);
    // собрать SQL-запрос, который извлечет новые сообщения
    if($id>0)
    {
        // извлечь сообщения, более новые, чем $id
        $query =
        'SELECT chat_id, user_name, message, color, ' .
        '       DATE_FORMAT(posted_on, "%Y-%m-%d %H:%i:%s") ' .
        '       AS posted_on ' .
        ' FROM chat WHERE chat_id > ' . $id .
        ' ORDER BY chat_id ASC';
    }
    else
    {
        // на начальной загрузке вернуть только последние 50 сообщений
        $query =
        ' SELECT chat_id, user_name, message, color, posted_on FROM ' .
        '  (SELECT chat_id, user_name, message, color, ' .
       '     DATE_FORMAT(posted_on, "%Y-%m-%d %H:%i:%s") AS posted_on ' .
        '     FROM chat ' .
        '     ORDER BY chat_id DESC ' .
        '       LIMIT 50) AS Last50' .
        ' ORDER BY chat_id ASC';
    }
    // выполнить запрос
    $result = $this->mMysqli->query($query);
    // собрать ответ в формате XML
   $response = '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
    $response .= '<response>';
```

```
        // вывести признак очистки
        $response .= $this->isDatabaseCleared($id);
        // проверить, а были ли новые сообщения в базе данных
        if($result->num_rows)
        {
            // добавить все извлеченные из базы данных сообщения
            // в ответ, который будет послан клиенту
            while ($row = $result->fetch_array(MYSQLI_ASSOC))
            {
                $id = $row['chat_id'];
                $color = $row['color'];
                $userName = $row['user_name'];
                $time = $row['posted_on'];
                $message = $row['message'];
                $response .= '<id>' . $id . '</id>' .
                             '<color>' . $color . '</color>' .
                             '<time>' . $time . '</time>' .
                             '<name>' . $userName . '</name>' .
                             '<message>' . $message . '</message>';
            }
            // закрыть соединение с базой данных
            $result->close();
        }
        // вставить завершающий тег XML в ответ и вернуть его
        $response = $response . '</response>';
        return $response;
    }

    /*
        Метод isDatabaseCleared проверяет, производилось ли удаление
        сообщений с момента последнего обращения к серверу
        - $id – идентификатор последнего сообщения, принятого клиентом
    */
    private function isDatabaseCleared($id)
    {
        if($id>0)
        {
            // подсчитав кол-во записей, в которых значение id меньше
            // чем $id мы узнаем, проводилась ли операция очистки
            // с момента последнего обращения клиента
            $check_clear =
                'SELECT count(*) old FROM chat where chat_id<=' . $id;
            $result = $this->mMysqli->query($check_clear);
            $row = $result->fetch_array(MYSQLI_ASSOC);
           // если очистка проводилась, необходимо очистить область сообщений
            if($row['old']==0)
                 return '<clear>true</clear>';
         }
         return '<clear>false</clear>';
     }
}
?>
```

8. **Создайте файл с именем** `get_color.php` **и добавьте в него следующий код:**

```
<?php
// имя файла с изображением
$imgfile='palette.png';
// загрузить файл с изображением
$img=imagecreatefrompng($imgfile);
// получить координаты точки, в которой пользователь щелкнул мышью
$offsetx=$_GET['offsetx'];
$offsety=$_GET['offsety'];
// получить цвет по заданным координатам
$rgb = ImageColorAt($img, $offsetx, $offsety);
$r = ($rgb >> 16) & 0xFF;
$g = ($rgb >> 8) & 0xFF;
$b = $rgb & 0xFF;
// вернуть код цвета
printf('#%02s%02s%02s', dechex($r), dechex($g), dechex($b));
?>
```

9. **Теперь перейдем на сторону клиента. Начнем с файла** `chat.css` **и добавим в него следующий код:**

```
body
{
    font-family: Tahoma, Helvetica, sans-serif;
    margin: 1px;
    font-size: 12px;
    text-align: left
}

#content
{
    border: DarkGreen 1px solid;
    margin-bottom: 10px
}

input
{
    border: #999 1px solid;
    font-size: 10px
}

#scroll
{
    position: relative;
    width: 340px;
    height: 270px;
    overflow: auto
}

.item
{
    margin-bottom: 6px
```

```
}

#colorpicker
{
    text-align:center
}
```

10. **Создайте новый файл с именем** `index.html` **и добавьте в него следующий код:**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>Чат AJAX</title>
       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"
/>
        <link href="chat.css" rel="stylesheet" type="text/css" />
        <script type="text/javascript" src="chat.js" ></script>
    </head>
    <body onload="init();">
        <noscript>
            Ваш броузер не поддерживает JavaScript!!
        </noscript>
        <table id="content">
            <tr>
                <td>
                    <div id="scroll">
                    </div>
                </td>
                <td id="colorpicker">
                    <img src="palette.png" id="palette" alt="Color
                        Palette" border="1" onclick="getColor(event);"/>
                    <br />
                    <input id="color" type="hidden" readonly="true"
                                        value="#000000" />
                    <span id="sampleText">
                         (текст будет выводиться таким цветом)
                    </span>
                </td>
            </tr>
        </table>
        <div>
            <input type="text" id="userName" maxlength="10" size="10"
                   onblur="checkUsername();"/>
            <input type="text" id="messageBox" maxlength="2000" size="50"
                   onkeydown="handleKey(event)"/>
            <input type="button" value="Отправить"
                                 onclick="sendMessage();" />
            <input type="button" value="Очистить"
                                 onclick="deleteMessages();" />
        </div>
```

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

11. Создайте файл `chat.js` **и добавьте в него следующий код:**

```
/* chatURL - URL страницы обновления сообщений */
var chatURL = "chat.php";
/* getColorURL - URL страницы, которая возвращает код выбранного цвета */
var getColorURL = "get_color.php";
/* создать объекты XMLHttpRequest, которые будут использоваться
   для получения сообщений и цвета */
var xmlHttpGetMessages = createXmlHttpRequestObject();
var xmlHttpGetColor = createXmlHttpRequestObject();
/* переменные, определяющие частоту обращений к серверу */
var updateInterval = 1000; // кол-во миллисекунд перед
                           // обращением за новым сообщением
// если true - выводить подробные сообщения об ошибках
var debugMode = true;
/* инициализировать кэш сообщений */
var cache = new Array();
/* lastMessageID - идентификатор самого последнего сообщения */
var lastMessageID = -1;
/* mouseX, mouseY - координаты указателя мыши */
var mouseX,mouseY;
/* создает экземпляр объекта XMLHttpRequest */
function createXmlHttpRequestObject()
{
    // ссылка на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
```

```
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

/* функция инициализации чата; исполняется,
   когда будет загружена страница */
function init()
{
    // получить ссылку на элемент ввода, где пользователь вводит текст
    var oMessageBox = document.getElementById("messageBox");
    // предотвратить запуск функции автодополнения
    oMessageBox.setAttribute("autocomplete", "off");
    // ссылка на текст «текст будет выводиться таким цветом»
    var oSampleText = document.getElementById("sampleText");
    // цвет по умолчанию - черный
    oSampleText.style.color = "black";
    // назначить пользователю случайное имя по умолчанию
    checkUsername();
    // инициировать процесс обновления окна чата
    requestNewMessages();
}

// эта функция назначает имя пользователя по умолчанию,
// если таковое еще не задано
function checkUsername()
{
    // обеспечивает назначение случайного имени пользователя
    // при загрузке формы
    var oUser=document.getElementById("userName");
    if(oUser.value == "")
        oUser.value = "Guest" + Math.floor(Math.random() * 1000);
}

/* эта функция вызывается по нажатию на кнопку «Отправить» */
function sendMessage()
{
    // сохранить текст сообщения в переменной и очистить поле ввода
    var oCurrentMessage = document.getElementById("messageBox");
    var currentUser = document.getElementById("userName").value;
    var currentColor = document.getElementById("color").value;
    // не передавать пустое сообщение
    if (trim(oCurrentMessage.value) != "" &&
      trim(currentUser) != "" && trim (currentColor) != "")
```

```
    {
        // нам необходимо передать сообщение и получить новые сообщения
        params = "mode=SendAndRetrieveNew" +
                 "&id=" + encodeURIComponent(lastMessageID) +
                 "&color=" + encodeURIComponent(currentColor) +
                 "&name=" + encodeURIComponent(currentUser) +
                 "&message=" + encodeURIComponent(oCurrentMessage.value);
        // добавить сообщение в очередь
        cache.push(params);
        // очистить поле ввода
        oCurrentMessage.value = "";
    }
}

/* эта функция вызывается по нажатию на кнопку «Очистить» */
function deleteMessages()
{
    // установить тип требуемой операции
    params = "mode=DeleteAndRetrieveNew";
    // добавить сообщение в очередь
    cache.push(params);
}

/* производит асинхронные обращения к серверу за получением новых
сообщений, отправляет сообщения из очереди и очищает сообщения */
function requestNewMessages()
{
    // получить имя пользователя и цвет
    var currentUser = document.getElementById("userName").value;
    var currentColor = document.getElementById("color").value;
    // продолжать только если xmlHttpGetMessages не содержит пустую ссылку
    if(xmlHttpGetMessages)
    {
        try
        {
            // не выполнять никаких операций, если еще
            // не завершилась предыдущая
            if (xmlHttpGetMessages.readyState == 4 ||
              xmlHttpGetMessages.readyState == 0)
            {
                // параметры запроса, отправляемого серверу
                var params = "";
                // если в очереди есть запросы, извлечь самый старый
                if (cache.length>0)
                    params = cache.shift();
                // если кэш пуст – просто получить новые сообщения
                else
                    params = "mode=RetrieveNew" +
                             "&id=" +lastMessageID;
                // послать серверу запрос на выполнение операции
                xmlHttpGetMessages.open("POST", chatURL, true);
                xmlHttpGetMessages.setRequestHeader("Content-Type",
```

```
                    "application/x-www-form-urlencoded");
                xmlHttpGetMessages.onreadystatechange =
                                                handleReceivingMessages;
                xmlHttpGetMessages.send(params);
            }
            else
            {
                // запланировать очередную проверку наличия новых сообщений
                setTimeout("requestNewMessages();", updateInterval);
            }
        }
        catch(e)
        {
            displayError(e.toString());
        }
    }
}

/* эта функция обрабатывает ответы http при получении новых сообщений */
function handleReceivingMessages()
{
    // продолжить только если прием завершен
    if (xmlHttpGetMessages.readyState == 4)
    {
        // продолжать только если статус HTTP = «OK»
        if (xmlHttpGetMessages.status == 200)
        {
            try
            {
                // обработать ответ сервера
                readMessages();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                displayError(e.toString());
            }
        }
        else
        {
            // вывести сообщение об ошибке
            displayError(xmlHttpGetMessages.statusText);
        }
    }
}

/* эта функция обрабатывает ответы сервера при получении новых сообщений */
function readMessages()
{
    // получить ответ сервера
    var response = xmlHttpGetMessages.responseText;
    // ошибка сервера?
```

```
    if (response.indexOf("ERRNO") >= 0
      || response.indexOf("error:") >= 0
      || response.length == 0)
        throw(response.length == 0 ? "Void server response." : response);
    // получить ссылку на корневой элемент
    response = xmlHttpGetMessages.responseXML.documentElement;
    // получить признак необходимости очистки области сообщений
    clearChat =
        response.getElementsByTagName("clear").item(0).firstChild.data;
    // если очистку необходимо провести
    if(clearChat == "true")
    {
        // очистить область сообщений и сбросить id
        document.getElementById("scroll").innerHTML = "";
        lastMessageID = -1;
    }
    // извлечь массивы из ответа сервера
    idArray = response.getElementsByTagName("id");
    colorArray = response.getElementsByTagName("color");
    nameArray = response.getElementsByTagName("name");
    timeArray = response.getElementsByTagName("time");
    messageArray = response.getElementsByTagName("message");
    // вывести новые сообщения в окно сообщений
    displayMessages(idArray, colorArray, nameArray, timeArray,
                    messageArray);
    // сохранить идентификатор последнего принятого сообщения
    if(idArray.length>0)
        lastMessageID = idArray.item(idArray.length - 1).firstChild.data;
    // перезапустить последовательность действий
    setTimeout("requestNewMessages();", updateInterval);
}

/* эта функция добавляет новые сообщения в окно сообщений */
function displayMessages(idArray, colorArray, nameArray,
                         timeArray, messageArray)
{
    // на каждом проходе цикла добавляется одно сообщение
    for(var i=0; i<idArray.length; i++)
    {
        // получить информацию о сообщении
        var color = colorArray.item(i).firstChild.data.toString();
        var time = timeArray.item(i).firstChild.data.toString();
        var name = nameArray.item(i).firstChild.data.toString();
        var message = messageArray.item(i).firstChild.data.toString();
        // собрать код HTML для отображения сообщения
        var htmlMessage = "";
        htmlMessage += "<div class=\"item\" style=\"color:" + color + "\">";
        htmlMessage += "[" + time + "] " + name + " said: <br/>";
        htmlMessage += message.toString();
        htmlMessage += "</div>";
        // вывести сообщение
        displayMessage (htmlMessage);
```

```
    }
}

// выводит сообщение
function displayMessage(message)
{
    // получить ссылку на элемент
    var oScroll = document.getElementById("scroll");
    // проверить – необходимо ли будет прокручивать окно
    // после добавления сообщения
    var scrollDown = (oScroll.scrollHeight - oScroll.scrollTop <=
                      oScroll.offsetHeight );
    // отобразить сообщение
    oScroll.innerHTML += message;
    // прокрутить окно вниз, если необходимо
    oScroll.scrollTop = scrollDown ? oScroll.scrollHeight :
    oScroll.scrollTop;
}

// эта функция выводит сообщение об ошибке
function displayError(message)
{
    // вывести подробное сообщение, если debugMode = true
    displayMessage("Ошибка доступа к серверу! "+
                   (debugMode ? "<br/>" + message : ""));
}

/* функция обработки нажатия клавиш, обнаруживает нажатие на клавишу Enter */
function handleKey(e)
{
    // получить событие
    e = (!e) ? window.event : e;
    // получить код символа нажатой клавиши
    code = (e.charCode) ? e.charCode :
            ((e.keyCode) ? e.keyCode :
            ((e.which) ? e.which : 0));
    // обработать событие keydown
    if (e.type == "keydown")
    {
        // если нажата клавиша Enter (код 13)
        if(code == 13)
        {
            // отправить текущее сообщение
            sendMessage();
        }
    }
}

/* удаляет ведущие и завершающие пробелы из строки */
function trim(s)
{
    return s.replace(/(^\s+)|(\s+$)/g, "")
}
```

```
/* вычисляет координаты указателя мыши */
function getMouseXY(e)
{
    // в зависимости от типа броузера
    if(window.ActiveXObject)
    {
        mouseX = window.event.x + document.body.scrollLeft;
        mouseY = window.event.y + document.body.scrollTop;
    }
    else
    {
        mouseX = e.pageX;
        mouseY = e.pageY;
    }
}

/* выполняет обращение к серверу за получением кода выбранного цвета */
function getColor(e)
{
    getMouseXY(e);
    // ничего не делать, если ссылка на объект XMLHttpRequest содержит null
    if(xmlHttpGetColor)
    {
        // инициализировать смещения по осям координат
        var offsetX = mouseX;
        var offsetY = mouseY;
        // получить ссылки
        var oPalette = document.getElementById("palette");
        var oTd = document.getElementById("colorpicker");
        // вычислить координаты в окне
        if(window.ActiveXObject)
        {
            offsetX = window.event.offsetX;
            offsetY = window.event.offsetY;
        }
        else
        {
            offsetX -= oPalette.offsetLeft + oTd.offsetLeft;
            offsetY -= oPalette.offsetTop + oTd.offsetTop;
        }
        // послать серверу запрос, чтобы получить цвет
        try
        {
            if (xmlHttpGetColor.readyState == 4 ||
              xmlHttpGetColor.readyState == 0)
            {
                params = "?offsetx=" + offsetX + "&offsety=" + offsetY;
                xmlHttpGetColor.open("GET", getColorURL+params, true);
                xmlHttpGetColor.onreadystatechange = handleGettingColor;
                xmlHttpGetColor.send(null);
            }
        }
```

```
        catch(e)
        {
            // вывести сообщение об ошибке
            displayError(xmlHttp.statusText);
        }
    }
}

/* эта функция обслуживает ответ http */
function handleGettingColor()
{
    // если сообщение получено, решить, что с ним делать
    if (xmlHttpGetColor.readyState == 4)
    {
        // только если статус HTTP = «OK»
        if (xmlHttpGetColor.status == 200)
        {
            try
            {
                //изменить цвет
                changeColor();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                displayError(xmlHttpGetColor.statusText);
            }
        }
        else
        {
            // вывести сообщение об ошибке
            displayError(xmlHttpGetColor.statusText);
        }
    }
}

/* эта функция изменяет цвет, которым будет отображаться текст сообщений */
function changeColor()
{
    response=xmlHttpGetColor.responseText;
    // ошибка сервера?
    if (response.indexOf("ERRNO") >= 0
      || response.indexOf("error:") >= 0
      || response.length == 0)
       throw(response.length == 0 ? "Невозможно изменить цвет!" : response);
    // изменить цвет
    var oColor = document.getElementById("color");
    var oSampleText = document.getElementById("sampleText");
    oColor.value = response;
    oSampleText.style.color = response;
 }
```

12. Теперь посмотрим, как работает наше приложение. Для начала взглянем на основное окно. Откройте в броузере страницу *http://localhost/ajax/chat.index.html*.

Как видите, по умолчанию сообщения выводятся черным цветом (код RGB: #000000). Кроме того, можно заметить (рис. 5.3), что по умолчанию пользователю было присвоено имя `Guest91`. При начальной загрузке страницы в ней отображаются все ранее отправленные сообщения. Цвет сообщений можно изменить щелчком мыши по изображению палитры.

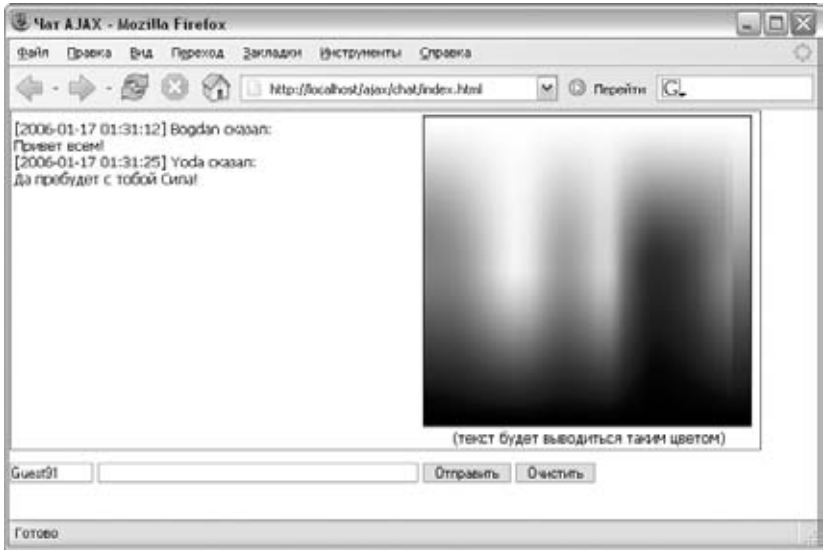

Рис. 5.3. *Внешний вид приложения Chat*

Что происходит внутри?

С технической точки зрения приложение состоит из двух приложений меньшего размера, которые и составляют наше решение:

- Приложение-чат
- Приложение выбора цвета

Приложение-чат реализует базовые функции отправления и приема сообщений. Любой пользователь может выбирать себе псевдоним и отправлять сообщения. Окно чата содержит список сообщений, обновляемый асинхронно.

Пользователю предоставляется возможность выбрать цвет сообщений из палитры `palette` содержащую широкий спектр цветов. После щелчка мышью по палитре координаты указателя отправляются серверу, который возвращает код выбранного цвета.

Уделив некоторое время анализу исходного кода, можно многое понять. Начнем с файла `index.html`. В нем нас интересует только область

прокрутки, которая может быть реализована на DHTML. Области прокрутки описаны по адресу *http://www.dyn-web.com/dhtml/scroll/*. Основная идея заключается в том, что прокручиваемая область конструируется из двух слоев, при этом один находится в другом. Мы добились этого, поместив внутренний слой в элемент `div scroll`.

Внешний слой представлен объектом `scroll`. Он имеет фиксированную высоту и ширину и очень полезное для нас свойство `overflow`. Вообще говоря, содержимое текстового поля ограничено его рамками. В некоторых случаях поле может переполняться (overflow), в результате часть содержимого оказывается за его пределами. В CSS свойство `overflow` определяет поведение элемента при переполнении. За дополнительной информацией по этой теме обращайтесь по адресу *http://www.w3.org/TR/REC-CSS2/visufx.html*.

Итак, мы определились с текстовым полем и с его поведением в случае переполнения, и теперь нетрудно догадаться, что внутри текстового поля находится элемент, который рано или поздно выйдет за границы объемлющего поля. Этот внутренний слой как раз и содержит текстовые сообщения.

Перейдем к файлу `chat.js`, который содержит код нашего приложения.

Весь файл делится на две части: первая отвечает за выбор цвета, а вторая – за обмен мгновенными сообщениями.

Начнем с выбора цвета. Эта часть, расположенная в самом начале, может показаться вам достаточно сложной, но на самом деле все гораздо проще. Посмотрим на картину в целом. Прежде всего у нас имеется изображение палитры с целым спектром цветов. В PHP есть две функции, которые помогут нам отыскать код цвета в палитре RGB для точки с заданными координатами, – `imagecreatefrompng` и `imagecolorat`. Когда мы будем обсуждать сценарий `get_color.php` мы подробнее поговорим об этих функциях. А пока нам достаточно знать, что эти две функции позволяют получить код цвета в формате RGB для точки с координатами x и y. Координаты точки мы получим с помощью функции `getMouseXY`.

Функция `getColor` возвращает код RGB цвета, выбранного пользователем щелчком мыши по изображению палитры. В первую очередь функция определяет координаты указателя мыши. Затем она определяет координаты указателя мыши относительно начала координат изображения. Делается это путем вычитания координат изображения внутри элемента `<td>` и координат самого элемента `td` в окне из координат указателя мыши, которые были получены функцией `getMouseXY`. Вычислив относительные координаты как `offsetx` и `offsety`, мы посылаем запрос сценарию сервера, который должен вернуть код RGB выбранного цвета. Изменение состояния соответствующего объекта `XMLHttpRequest` обрабатывается функцией `handleGettingColor`.

Функция `handleGettingColor` проверяет, завершился ли предыдущий запрос к серверу, и если ошибок обнаружено не было, она вызывает

функцию changeColor. Эта функция заносит в скрытое текстовое поле код RGB и окрашивает текстовое поле примера выбранным цветом.

Теперь посмотрим, как работает сам чат.

По умолчанию, когда происходит инициализация страницы, возникает событие onblur, по которому вызывается функция checkUsername. Эта функция гарантирует, что поле с именем пользователя не останется пустым, генерируя произвольное имя.

При нажатии кнопки Отправить вызывается функция sendMessage. Эта функция добавляет текущее сообщение в очередь, предварительно удалив из него ведущие и завершающие пробелы с помощью функции trim и преобразовав сообщение в форму, в которой оно безопасно может быть отправлено серверу, с помощью функции encodeURIComponent.

При нажатии какой-либо клавиши возникает событие keydown, по которому вызывается функция handleKey. Если была нажата клавиша Enter, вызывается функция sendMessage. Таким образом, нажатие кнопки Отправить и нажатие клавиши Enter, когда фокус ввода находится в элементе messageBox, приводят к выполнению одного и того же действия.

Функция deleteMessages добавляет запрос серверу на удаление сообщений.

За отправку сообщений отвечает функция requestNewMessages. Она извлекает сообщение из очереди и отправляет его серверу. Изменение состояния объекта XMLHttpRequest обслуживается функцией handleReceivingMessages.

Функция handleReceivingMessages проверяет, завершился ли предыдущий запрос к серверу, и если ошибок обнаружено не было, она вызывает функцию readMessages.

Функция readMessages проверяет, не стер ли кто-нибудь все сообщения в чате, и если это происходило, она очищает область сообщений в окне клиента. Для добавления новых сообщений в окно клиента вызывается функция displayMessages. Эта функция принимает в качестве аргументов массивы с новыми сообщениями, из которых она строит код HTML и добавляет его к уже существующему коду, вызывая функцию displayMessage. В самом начале функция displayMessage проверяет – находится ли полоса прокрутки в самом низу списка сообщений. Это необходимо для того, чтобы прокрутить область сообщений вниз в конце функции.

И последняя функция – функция init. Она запрашивает последние сообщения в чате, подставляет имя пользователя, устанавливает цвет по умолчанию и выключает функцию автодополнения.

Обработка ошибок возложена на функцию displayError. Она вызывается из функции displayMessage, которая передает ей сообщение об ошибке в качестве параметра.

Теперь переместимся на сторону сервера и начнем с рассмотрения файла chat.php. Сервер обслуживает запросы клиентов по следующей схеме:

- Получает параметры запроса.
- Идентифицирует затребованную операцию.
- Выполняет необходимые действия.
- Возвращает клиенту полученный результат.

В состав запроса входит параметр mode, который определяет, какая из следующих операций должна быть выполнена сервером:

- SendAndRetrieve. Сначала новые сообщения записываются в базу данных, после чего все они отправляются обратно клиенту.
- DeleteAndRetrieve. Все сообщения стираются, а все новые сообщения, появившиеся к этому моменту, извлекаются из базы данных и отправляются клиенту.
- Retrieve. Новые сообщения извлекаются из базы данных и отправляются клиенту.

Вся бизнес-логика, лежащая в основе chat.php, реализуется сценарием chat.class.php, который содержит определение класса Chat.

Метод deleteMessages удаляет из таблицы всю информацию, содержащуюся в ней.

Метод postMessages записывает новые сообщения в базу данных.

Метод isDatabaseCleared проверяет, производилось ли удаление сообщений. Основываясь на идентификаторе последнего принятого клиентом сообщения, она определяет, были ли в базе данных стерты все более старые сообщения.

Метод retrieveNewMessages извлекает все новые сообщения, начиная с последнего (идентифицируется по входному аргументу id), полученного с сервера в результате последнего запроса (если таковой вообще был, в противном случае высылаются все сообщения). Он также проверяет, производилась ли очистка базы данных, обращаясь к методу isDatabaseCleared. Затем функция собирает ответ и отправляет его клиенту.

Файл config.php содержит параметры доступа к базе данных. Файл error_handler.php содержит код модуля обработки ошибок.

Теперь рассмотрим реализацию возможности выбора цвета на стороне сервера, которая находится в файле get_color.php.

Мы уже упоминали две функции PHP, предназначенные для получения кода цвета точки в изображении. Теперь посмотрим, как они работают:

- imagecreatefrompng(string filename) возвращает идентификатор изображения в формате PNG, полученного из файла filename.
- int imagecolorat(resource image, int x, int y) возвращает индекс цвета точки с заданными координатами в изображении image. Если

PHP собран с поддержкой библиотеки GD 2.0 или выше, а само изображение содержит естественные цвета (true-color – 24 или 32 бита на пиксел), то функция возвращает целое число в формате RGB.

Первые 8 бит результата определяют интенсивность голубого цвета, следующие 8 – зеленого и следующие 8 бит – красного. С помощью операций сдвига и маскирования мы получаем отдельные значения красной, зеленой и голубой составляющей цвета в виде целочисленных значений. После этого остается лишь преобразовать полученные числа в шестнадцатеричное представление, объединить их в одну строку и отправить клиенту.

Давайте еще раз коротко вспомним все, о чем говорили. Мы начали с интерфейса пользователя, который предоставляется *клиентской частью приложения* и реализуется в файлах HTML, CSS и JavaScript – `index.html`, `chat.css` и `chat.js`. Рассмотрев, как выглядит интерфейс приложения и как обрабатываются данные, получаемые от сервера, мы перешли к серверной части приложения.

Мы рассмотрели сценарии, которые вызываются со стороны клиента: `chat.php` и `get_color.php`. В заключение мы рассмотрели файл, содержащий параметры соединения с базой данных (`config.php`), модуль обработки ошибок (`error_handler.php`) и сценарий, реализующий основные функциональные возможности (`chat.class.php`).

Подведение итогов

В начале этой главы мы увидели, какие трудности могут возникнуть при организации общения через Интернет. Мы показали, как технология AJAX помогает справиться с этими трудностями. Рассмотрев некоторые варианты реализации решений на базе AJAX, мы приступили к созданию собственного решения. Шаг за шагом мы создали свой чат на базе AJAX, сделав его простым, модульным и легко расширяемым.

Прочитав эту главу, вы можете попытаться улучшить наше решение, добавив в него следующие возможности:

- Комнаты в чате.
- Простую командную строку (вход/выход из комнаты чата, переход из комнаты в комнату).
- Передача частных сообщений.

6

Подсказки и функция автодополнения в AJAX

Подсказки и функция автодополнения – это весьма популярные функциональные возможности, реализуемые в большинстве современных веб-броузеров, в программах чтения электронной почты, текстовых редакторах и процессорах и в операционных системах. Подсказки и автодополнение – это две стороны одной медали, они всегда идут рядом, рука об руку. Обычно между ними не делается различий, но термин «автодополнение» употребляется гораздо чаще.

Функция автодополнения заключается в том, чтобы предсказать, какое слово или фразу пользователь желает ввести с клавиатуры. Эта возможность повышает скорость общения и делает пользовательский интерфейс более дружественным. Она помогает правильно строить выражения и снижает вероятность опечаток.

Функцию автодополнения в действии можно увидеть в броузерах, вводя новый адрес в адресной строке или заполняя какие-либо формы. В программах чтения электронной почты механизм автодополнения помогает выбрать адреса получателя после ввода одного-двух первых символов.

Трудно переоценить возможность автодополнения в специализированных текстовых редакторах, ориентированных на работу с исходными текстами программ. Длинные имена переменных упрощают понимание исходного кода, но их намного сложнее вводить с клавиатуры, если в редакторе отсутствует функция автодополнения. В некоторых редакторах, набрав имя объекта и точку, можно увидеть прокручиваемый список со всеми общедоступными членами объекта. Это очень напоминает справочную документацию, которая всегда находится под рукой. Microsoft реализовала эту функцию в **Visual Studio Integrated Development Environment** и запатентовала ее под названием **IntelliSense**. Однако в редактор **GNU Emacs** функция автодополнения была реализована на много лет раньше, чем она появилась в продуктах Microsoft.

Командные оболочки в UNIX, такие как bash и sh, или командная строка в Windows предоставляют возможность автоматического завершения имен команд, файлов и каталогов, которое обычно выполняется по нажатию клавиши табуляции после ввода нескольких первых символов слова. Я уверен, что вы найдете эту функцию очень удобной и полезной, если вам приходится много печатать![1]

Введение в подсказки и функцию автодополнения на базе AJAX

Функция автодополнения – это еще один прекрасный пример функциональных возможностей, которые традиционно использовались только в локальных приложениях. Популярные реализации этой функции в веб-приложениях появились совсем недавно. (Обратите внимание: функции автодополнения или возможность запоминания паролей, которые вы можете наблюдать в броузерах, реализуются локально, самими броузерами, а не сайтами.)

Все это обогащает пользовательский интерфейс веб-приложений новыми функциональными возможностями, которые ранее были присущи только локальным приложениям. Замечательный пример реализации функции автодополнения на базе AJAX можно найти по адресу *http://demo.script.aculo.us/ajax/autocompleter*.

Как самый известный пример реализации этой функциональной возможности назовем Google Suggest.

Google Suggest

Почему Google Suggest? Потому что это самая известная реализация подсказок и автодополнения, основанная на технологии AJAX. Хотите верьте, хотите – нет, но пальма первенства реализации этой технологии принадлежит вовсе не Google. Кристиан Стокер (Christian Stocker) использовал ее в своем блоге Bitflux Blog (*http://blog.bitflux.ch/archive/2004/07/13/livesearch_roundup.html*) еще в апреле 2004 года – за семь месяцев до того, как она была реализована Google. По адресу *http://www.sitepoint.com/article/life-autocomplete-textboxes* вы найдете статью, которая подробно описывает, как с помощью JavaScript реализовать функцию автодополнения в текстовых полях на веб-странице. Эта статья датирована сентябрем 2003 года! Объект `XMLHttpRequest` к этому времени был известен уже несколько лет. Таким образом,

[1] Примером одного из самых, пожалуй, удачных применений этой техники в программных продуктах последнего времени является текстовый процессор «свободного» ПО OpenOffice с его эффективной и «ненавязчивой» (а это и есть «оборотная сторона медали» описываемой технологии) техникой автодополнения. – *Примеч. науч. ред.*

в Google не изобрели ничего нового – они просто объединили известные компоненты (рис. 6.1).

Доступ к Google Suggest можно получить по адресу *http://www.google.com/webhp?complete=1&hl=en*.

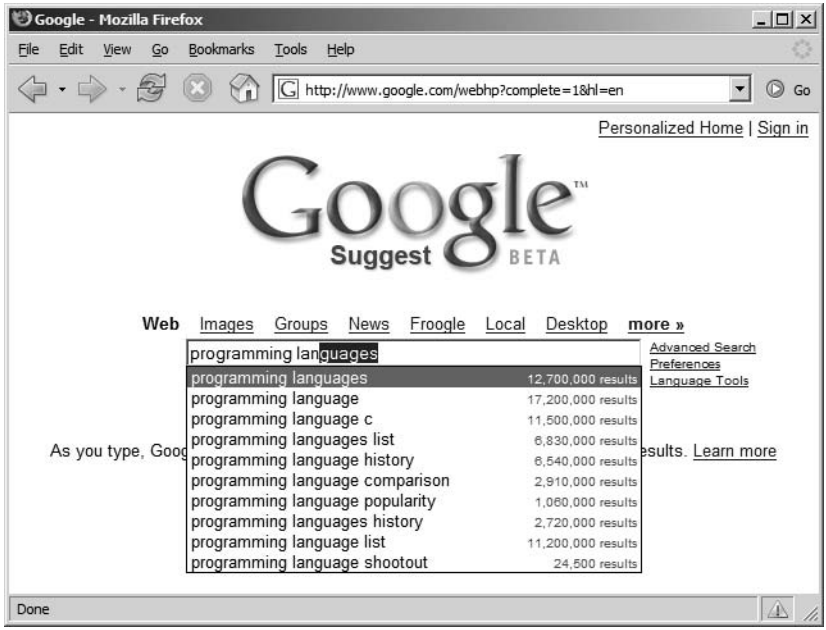

Рис. 6.1. *Демонстрация возможностей Google Suggest*

Самая примечательная часть решения, реализованного в сценарии JavaScript приложения Google Suggest, заключается в том, что оно запоминает таблицу подсказок, предлагавшихся ранее для определенных ключевых слов. То есть если ввести ключевое слово, а затем стереть несколько последних символов, источником подсказок будет служить внутренний буфер клиентского приложения, и приложению не придется повторно загружать их с сервера.

Аналогичная функциональность реализована в Gmail (*www.gmail.com*) и Google Maps (*http://maps.google.com*).

Реализация подсказок и функции автодополнения средствами AJAX

В этой главе мы реализуем функции подсказок и автодополнения, которые помогут вашим пользователям отыскивать описания функций PHP на официальном сайте *http://www.php.net*. База данных с названиями функций PHP, которая потребуется нашему приложению, находится по адресу *http://www.php.net/quickref.php*.

В нашем приложении мы реализуем следующие функциональные возможности:

- Поиск вариантов имен функций по мере ввода символов с клавиатуры и отображение их в виде выпадающего списка.
- Текущее слово в поле ввода будет автоматически дополняться недостающими символами, которые будут извлекаться из первой предлагаемой подсказки. Символы, добавленные функцией автодополнения, будут отображаться как выделенные.
- Первые символы названий функций в выпадающем списке, совпадающие с введенными символами, будут выделяться жирным шрифтом.
- Выпадающий список будет прокручиваемым, но полоса прокрутки будет появляться только в том случае, если количество подсказок в списке превысит некоторое предопределенное число (рис. 6.2).

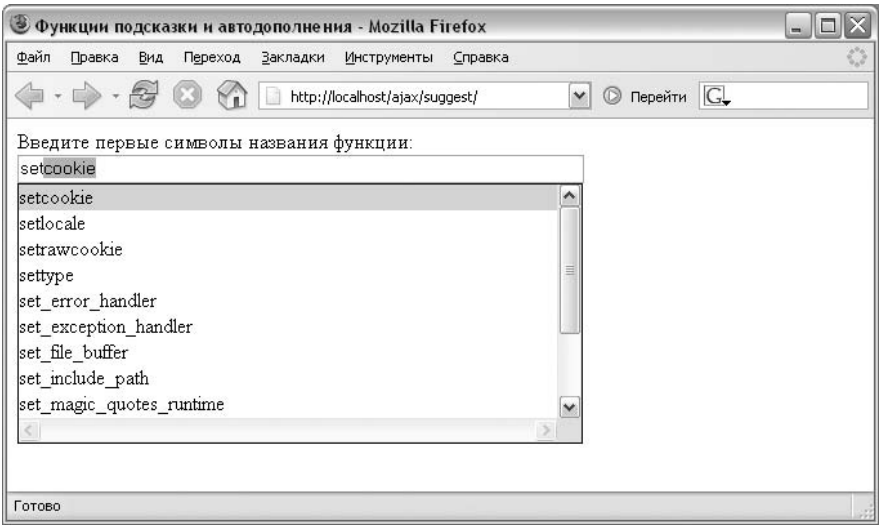

Рис. 6.2. *Результат поиска по нескольким первым символам*

Время действовать – подсказки и функция автодополнения на базе AJAX

1. Как обычно начнем с создания базы данных необходимой структуры. Создайте в базе данных `ajax` новую таблицу с именем `suggest`. Эта таблица будет содержать единственное поле, которое одновременно будет играть роль первичного ключа:

```
CREATE TABLE suggest
(
    name VARCHAR(100) NOT NULL DEFAULT '',
    PRIMARY KEY (name)
);
```

2. Таблица `suggest` должна быть заполнена списком функций PHP, имена которых мы взяли по адресу *http://www.php.net/quickref.php*. Так как таблица содержит более 4000 записей, здесь мы перечислим лишь первые десять. Полную версию этого сценария можно скачать с сайта книги:[1]

```
INSERT INTO suggest (name) VALUES
     ('abs'),
     ('acos'),
     ('acosh'),
     ('addcslashes'),
     ('addslashes'),
     ('aggregate'),
     ('aggregate_info'),
     ('aggregate_methods'),
     ('aggregate_methods_by_list'),
     ('aggregate_methods_by_regexp');
```

3. В каталоге `ajax` создайте новый каталог с именем `suggest`.

4. В первую очередь мы представим сценарии серверной части приложения. В каталоге `suggest` создайте новый файл с именем `config.php` и добавьте в него код с конфигурацией (измените эти значения так, чтобы они соответствовали вашей конфигурации):

```
<?php
// сведения, необходимые для соединения с базой данных
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

5. Затем добавьте стандартный сценарий обработки ошибок `error_handler.php`:

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
```

[1] Найти файл, содержащий этот объемный сценарий, на сайте книги не удалось, но он может быть воссоздан и более простым путем: копированием содержимого полной страницы *http://www.php.net/quickref.php* через буфер обмена в текстовый файл, содержащий редактируемую команду INSERT, с последующим текстовым редактированием этого списка по образу и подобию показанного автором фрагмента. – *Примеч. науч. ред.*

```
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

6. **Создайте файл** `suggest.php` **и добавьте в него следующий код:**

```
<?php
// загрузить файл с определением класса Suggest
require_once('suggest.class.php');
// создать экземпляр класса Suggest
$suggest = new Suggest();
// извлечь ключевое слово, переданное сценарию в качестве параметра
$keyword = $_GET['keyword'];
// очистить выходной буфер
if(ob_get_length()) ob_clean();
// не дать возможность клиентскому броузеру помещать результаты в кэш
header('Expires: Mon, 26 Jul 1997 05:00:00 GMT' );
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/xml');
// отправить результаты клиенту
echo $suggest->getSuggestions($keyword);
?>
```

7. **Создайте файл с именем** `suggest.class.php` **и добавьте в него следующий код:**

```
<?php
// загрузить модуль обработки ошибок
require_once('error_handler.php');
// загрузить файл с конфигурацией
require_once('config.php');
// определение класса Suggest, реализующего функции
// подсказок и автодополнения
class Suggest
{
    // соединение с базой данных
    private $mMysqli;
    // конструктор, открывает соединение с базой данных
    function __construct()
    {
        // соединиться с базой данных
        $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                    DB_DATABASE);
    }
```

```
    // деструктор, закрывает соединение с базой данных
    function __destruct()
    {
        $this->mMysqli->close();
    }

    // возвращает список имен функций PHP, имена
    // которых начинаются с $keyword
    public function getSuggestions($keyword)
    {
        // экранировать служебные символы
        $patterns = array('/\s+/', '/"+/', '/%+/');
        $replace = array('');
        $keyword = preg_replace($patterns, $replace, $keyword);
       // создать запрос SQL, который извлечет имена функций из базы данных
        if($keyword != '')
            $query = 'SELECT name ' .
                     'FROM suggest ' .
                     'WHERE name LIKE "' . $keyword . '%"';
        // если ключевое слово пустое – создать запрос SQL,
        // который вернет пустой набор данных
        else
            $query = 'SELECT name ' .
                     'FROM suggest ' .
                     'WHERE name=""';
        // выполнить запрос
        $result = $this->mMysqli->query($query);
        // создать ответ в формате XML
        $output = '<?xml version="1.0" encoding="UTF-8" s
                                        tandalone="yes"?>';
        $output .= '<response>';
        // если был получен не пустой результат запроса,
        // обойти в цикле полученный набор данных и добавить
        // имена функций в строку результата
        if($result->num_rows)
            while ($row = $result->fetch_array(MYSQLI_ASSOC))
                $output .= '<name>' . $row['name'] . '</name>';
        // закрыть входной поток
        $result->close();
        // добавить завершающий тег
        $output .= '</response>';
        // вернуть результаты
        return $output;
    }
    //конец определения класса Suggest
}
?>
```

8. **Создайте файл с именем** `index.html` **и добавьте в него следующий код:**

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

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>Функции подсказки и автодополнения</title>
        <meta http-equiv="Content-Type" content="text/html;
                                        charset=UTF-8" />
        <link href="suggest.css" rel="stylesheet" type="text/css" />
        <script type="text/javascript" src="suggest.js"></script>
    </head>
    <body>
        <noscript>
            Ваш броузер не имеет поддержки JavaScript!!
        </noscript>
        <div id="content" onclick="hideSuggestions();">
            <div id="message">Введите первые буквы
                              имени функции:</div>
            <input type="text" name="keyword" id="keyword" maxlength="70"
                   size="69" onkeyup = "handleKeyUp(event)" value="" />
            <div id="scroll">
                <div id="suggest">
                </div>
            </div>
        </div>
    </body>
</html>
```

9. Создайте файл с именем `suggest.css` **и добавьте в него следующий код:**

```
body
{
    font-family: helvetica, sans-serif;
    margin: 0px;
    padding: 0px;
    font-size: 12px
}

#content
{
    height: 100%;
    width: 100%;
    text-align:center
}

#message
{
    font-weight: bold;
    text-align: center;
    margin-left: 10px;
    margin-bottom: 10px;
    margin-top: 10px
}

a
{
```

```
    text-decoration: none;
    margin: 0px;
    color: #173f5f
}

input
{
    border: #999 1px solid;
    font-family: helvetica, sans-serif;
    font-weight: normal;
    font-size: 10px
}

#scroll
{
    position: relative;
    margin: 0 auto;
    visibility: hidden;
    background-color: white;
    z-index: 1;
    width: 300px;
    height: 180px;
    border-top-style: solid;
    border-right-style: solid;
    border-left-style: solid;
    border-collapse: collapse;
    border-bottom-style: solid;
    border-color: #000000;
    border-width: 1px;
    overflow: auto
}

#scroll div
{
    margin: 0 auto;
    text-align:left
}

#suggest table
{
    width: 270px;
    font-size: 11px;
    font-weight: normal;
    color: #676767;
    text-decoration: none;
    border: 0px;
    padding: 0px;
    text-align:left;
    margin: 0px
}

.highlightrow
{
```

```
    background-color: #999999;
    cursor: pointer
}
```

10. Создайте файл с именем `suggest.js` **и добавьте в него следующий код:**

```
/* URL сценария PHP, который возвращает подсказки для ключевого слова */
var getFunctionsUrl = "suggest.php?keyword=";
/* URL страницы, где можно посмотреть описание найденной функции */
var phpHelpUrl="http://www.php.net/manual/en/function.";
/* ключевое слово, которым был инициирован запрос HTTP */
var httpRequestKeyword = "";
/* последнее ключевое слово, для которого были получены подсказки */
var userKeyword = "";
/* количество принятых подсказок для ключевого слова */
var suggestions = 0;
/* максимальное число отображаемых символов в подсказке */
var suggestionMaxLength = 30;
/* признак того, что нажимались курсорные клавиши «вверх» или «вниз»,
   когда последний раз возникало событие keyup */
var isKeyUpDownPressed = false;
/* последняя подсказка, которая была использована для автодополнения */
var autocompletedKeyword = "";
/* признак того, что для заданного ключевого был получен не пустой список
   функций */
var hasResults = false;
/* идентификатор, использовавшийся для операции отмены с помощью метода
   clearTimeout. */
var timeoutId = -1;
/* текущая выбранная подсказка (курсорными клавишами или мышкой)*/
var position = -1;
/* буферный объект, который содержит принятые подсказки для различных
   ключевых слов */
var oCache = new Object();
/* минимальная и максимальная позиция видимых подсказок */
var minVisiblePosition = 0;
var maxVisiblePosition = 9;
// если имеет значение true – выводятся подробные сообщения об ошибках
var debugMode = true;
/* объект XMLHttp для взаимодействия с сервером */
var xmlHttpGetSuggestions = createXmlHttpRequestObject();
/* событие onload обрабатывается нашей функцией init */
window.onload = init;

// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // ссылка на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
```

```
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

/* функция инициализации страницы */
function init()
{
    // получить ссылку на элемент ввода ключевого слова
    var oKeyword = document.getElementById("keyword");
    // отключить функцию автодополнения броузера
    oKeyword.setAttribute("autocomplete", "off");
    // очистить поле ввода ключевого слова и установить в него фокус ввода
    oKeyword.value = "";
    oKeyword.focus();
    // установить время ожидания, которое должно пройти с момента
    // последнего нажатия на клавишу, перед тем как начать поиск
    setTimeout("checkForChanges()", 500);
}

/* добавляет в кэш массив значений с ключевым словом */
function addToCache(keyword, values)
{
    // создать новую запись в буфере
```

```
    oCache[keyword] = new Array();
    // добавить все значения для ключевого слова в буфер
    for(i=0; i<values.length; i++)
        oCache[keyword][i] = values[i];
}

/*
    Определяет наличие в буфере ключевого слова, совпадающего с аргументом
    keyword, если такого слова в буфере нет – пытается отыскать в буфере
    самые длинные префиксы, совпадающие с аргументом keyword
    и добавляет их в буфер для заданного ключевого слова
*/
function checkCache(keyword)
{
    // проверить наличие keyword в буфере
    if(oCache[keyword])
        return true;
    // попытаться отыскать наибольшие префиксы
    for(i=keyword.length-2; i>=0; i--)
    {
        // принять в качестве префикса текущее количество
        // первых символов в keyword
        var currentKeyword = keyword.substring(0, i+1);
        // если текущий префикс имеется в буфере
        if(oCache[currentKeyword])
        {
            // текущий префикс имеется в буфере
            var cacheResults = oCache[currentKeyword];
            // переместить совпадения в текущий буфер результатов
            var keywordResults = new Array();
            var keywordResultsSize = 0;
            // попробовать отыскать все совпадения,
            // начинающиеся с текущего префикса
            for(j=0;j<cacheResults.length;j++)
            {
                if(cacheResults[j].indexOf(keyword) == 0)
                  keywordResults[keywordResultsSize++] = cacheResults[j];
            }
            // добавить все найденные результаты в буфер
            addToCache(keyword, keywordResults);
            return true;
        }
    }
    // совпадений не найдено
    return false;
}

/* инициирует запрос HTTP для получения подсказок к текущему
   ключевому слову */
function getSuggestions(keyword)
{
    /* продолжать только если ключевое слово не пустое и последняя
```

```
   нажатая клавиша не была курсорной клавишей «вверх» или «вниз» */
if(keyword != "" && !isKeyUpDownPressed)
{
    // проверить наличие ключевого слова в буфере
    isInCache = checkCache(keyword);
    // если ключевое слово имеется в буфере...
    if(isInCache == true)
    {
        // извлечь совпадения из буфера
        httpRequestKeyword=keyword;
        userKeyword=keyword;
        // отобразить результаты из кэша
      displayResults(keyword, oCache[keyword]);
    }
    // если слово в буфере найдено не было, послать запрос HTTP
    else
    {
        if(xmlHttpGetSuggestions)
        {
            try
            {
                /* если объект XMLHttpRequest не занят обработкой
                   предыдущего запроса... */
                if (xmlHttpGetSuggestions.readyState == 4 ||
                    xmlHttpGetSuggestions.readyState == 0)
                {
                    httpRequestKeyword = keyword;
                    userKeyword = keyword;
                    xmlHttpGetSuggestions.open("GET",
                              getFunctionsUrl + encode(keyword), true);
                    xmlHttpGetSuggestions.onreadystatechange =
                                 handleGettingSuggestions;
                    xmlHttpGetSuggestions.send(null);
                }
                // если объект XMLHttpRequest занят...
                else
                {
                   // сохранить ключевое слово, введенное пользователем
                    userKeyword = keyword;
                    // сбросить все ранее установленные таймауты
                    if(timeoutId != -1)
                        clearTimeout(timeoutId);
                    // повторить попытку через 0.5 секунды
                timeoutId = setTimeout("getSuggestions(userKeyword);",
                                       500);
                }
            }
            catch(e)
            {
               displayError("Невозможно соединиться с сервером:\n" +
                            e.toString());
```

```
                }
            }
        }
    }
}

/* преобразовать все дочерние узлы документа XML в массив */
function xmlToArray(resultsXml)
{
    // инициализировать массив resultsArray
    var resultsArray= new Array();
    // обойти в цикле все узлы xml и извлечь их содержимое
    for(i=0;i<resultsXml.length;i++)
        resultsArray[i]=resultsXml.item(i).firstChild.data;
    // вернуть содержимое узлов в виде массива
    return resultsArray;
}

/* обрабатывает HTTP ответ сервера, который содержит подсказки
   к заданному ключевому слову  */
function handleGettingSuggestions()
{
    // если запрос выполнен - решить, что делать с данными
    if (xmlHttpGetSuggestions.readyState == 4)
    {
        // продолжать, только если получен статус HTTP «OK»
        if (xmlHttpGetSuggestions.status == 200)
        {
            try
            {
                // обработать ответ сервера
                updateSuggestions();
            }
            catch(e)
            {
                // вывести сообщение об ошибке
                displayError(e.toString());
            }
        }
        else
        {
            displayError("Возникли проблемы при приеме данных:\n" +
                         xmlHttpGetSuggestions.statusText);
        }
    }
}

/* обрабатывает ответ сервера */
function updateSuggestions()
{
    // извлечь ответ сервера
    var response = xmlHttpGetSuggestions.responseText;
```

```
    // ошибка сервера?
    if (response.indexOf("ERRNO") >= 0
          || response.indexOf("error:") >= 0
          || response.length == 0)
        throw(response.length == 0 ? "Void server response." : response);
    // получить ссылку на корневой элемент
    response = xmlHttpGetSuggestions.responseXML.documentElement;
    // инициализировать новый массив имен функций
    nameArray = new Array();
    // проверить – найдена ли хоть одна функция по нашему запросу
    if(response.childNodes.length)
    {
        /* извлечь имена функций из документа в виде массива */
       nameArray= xmlToArray(response.getElementsByTagName("name"));
    }
    // определить – это ли слово ищет пользователь
    if(httpRequestKeyword == userKeyword)
    {
        // отобразить массив результатов
        displayResults(httpRequestKeyword, nameArray);
    }
    else
    {
        // добавить результаты в буфер
        // мы не должны отображать полученные результаты,
        // если они более не нужны
        addToCache(httpRequestKeyword, nameArray);
    }
}

/* заполнить список подсказками */
function displayResults(keyword, results_array)
{
    // начало сборки таблицы HTML, содержащей результаты
    var div = "<table>";
    // если ключевое слово отсутствует в буфере, то добавить его туда
    if(!oCache[keyword] && keyword)
        addToCache(keyword, results_array);
    // если массив результатов пуст – вывести сообщение
    if(results_array.length == 0)
    {
        div += "<tr><td>По ключевому слову <strong>" + keyword +
           " ничего не найдено</strong></td></tr>";
        // установить признак того, что ничего не было найдено
        // и сбросить счетчик результатов
        hasResults = false;
        suggestions = 0;
    }
    // вывести результаты
    else
    {
```

```
// сбросить индекс выбранной подсказки
position = -1;
// сбросить признак нажатия на клавишу «вверх» или «вниз»
isKeyUpDownPressed = false;
/* установить признак того, что для данного ключевого
   слова имеются результаты */
hasResults = true;
// получить количество результатов в буфере
suggestions = oCache[keyword].length;
// обойти результаты в цикле и сгенерировать список результатов
// в формате HTML
for (var i=0; i<oCache[keyword].length; i++)
{
    // получить имя функции
    crtFunction = oCache[keyword][i];
    // сгенерировать ссылку на описание функции
    crtFunctionLink = crtFunction;
    // заменить в строке символ _ символом -
    while(crtFunctionLink.indexOf("_") !=-1)
        crtFunctionLink = crtFunctionLink.replace("_","-");
    // создать строку таблицы, которая будет содержать ссылку
    // на справочную страницу для данной функции PHP
    div += "<tr id='tr" + i +
        "' onclick='location.href=document.getElementById(\"a" + i +
          "\").href;' onmouseover='handleOnMouseOver(this);' " +
          "onmouseout='handleOnMouseOut(this);'>" +
          "<td align='left'><a id='a" + i +
          "' href='" + phpHelpUrl + crtFunctionLink + ".php";
   // поверить, не превышает ли текущее имя функции максимального
    // количества символов, которые могут быть отображены
    if(crtFunction.length <= suggestionMaxLength)
    {
        // выделить жирным шрифтом часть функции, которая
        // совпадает с ключевым словом
        div += "'><b>" +
             crtFunction.substring(0, httpRequestKeyword.length) +
               "</b>"
        div += crtFunction.substring(httpRequestKeyword.length,
                                     crtFunction.length) +
                                     "</a></td></tr>";
    }
    else
    {
        // поверить, не превышает ли длина ключевого слова
        // максимального количества символов,
        // которые могут быть отображены
        if(httpRequestKeyword.length < suggestionMaxLength)
        {
            // выделить жирным шрифтом часть функции, которая
            // совпадает с ключевым словом
            div += "'><b>" +
```

```
                        crtFunction.substring(0, httpRequestKeyword.length) +
                                                    "</b>"
                        div += crtFunction.substring(httpRequestKeyword.length,
                                                suggestionMaxLength) +
                                                "</a></td></tr>";
                    }
                    else
                    {
                        // выделить жирным шрифтом все имя функции
                        div += "'><b>" +
                              crtFunction.substring(0,suggestionMaxLength) +
                               "</b></td></tr>"
                    }
                }
            }
        }
        // завершить создание таблицы HTML
        div += "</table>";
        // получить ссылку на объекты suggest и scroll
        var oSuggest = document.getElementById("suggest");
        var oScroll = document.getElementById("scroll");
        // прокрутить в начало списка
        oScroll.scrollTop = 0;
        // обновить список подсказок и сделать его видимым
        oSuggest.innerHTML = div;
        oScroll.style.visibility = "visible";
        // если результаты были получены, дополняем ключевое слово
        if(results_array.length > 0)
            autocompleteKeyword();
    }

    /* периодически проверяет, изменилось ли ключевое слово */
    function checkForChanges()
    {
        // получить ссылку на объект keyword
        var keyword = document.getElementById("keyword").value;
        // проверить – есть ли что-нибудь в поле ввода
        if(keyword == "")
        {
            // скрыть список с подсказками
            hideSuggestions();
            // сбросить соответствующие переменные
            userKeyword="";
            httpRequestKeyword="";
        }
        // взвести таймер для выполнения очередной проверки
        setTimeout("checkForChanges()", 500);
        // проверить – было ли изменено ключевое слово
        if((userKeyword != keyword) &&
             (autocompletedKeyword != keyword) &&
             (!isKeyUpDownPressed))
```

```
      // обновить список подсказок
      getSuggestions(keyword);
}

/* обрабатывает событие нажатия на клавиши */
function handleKeyUp(e)
{
    // получить ссылку на событие
    e = (!e) ? window.event : e;
    // получить ссылку на приемник события
    target = (!e.target) ? e.srcElement : e.target;
    if (target.nodeType == 3)
        target = target.parentNode;
    // получить код символа нажатой клавиши
    code = (e.charCode) ? e.charCode :
           ((e.keyCode) ? e.keyCode :
           ((e.which) ? e.which : 0));
    // убедиться, что получено событие keyup
    if (e.type == "keyup")
    {
        isKeyUpDownPressed =false;
        // проверить, нажата ли клавиша, которая нас интересует
        if ((code < 13 && code != 8) ||
            (code >=14 && code < 32) ||
            (code >= 33 && code <= 46 && code != 38 && code != 40) ||
            (code >= 112 && code <= 123))
        {
            // просто игнорировать клавиши, которые нас не интересуют
        }
        else
        /* если нажата клавиша Enter, перейти на страницу с описанием
           текущей функции */
        if(code == 13)
        {
            // проверить, выбрана ли какая-нибудь функция
            if(position>=0)
            {
                location.href = document.getElementById("a" +
                                                        position).href;
            }
        }
        else
        // если нажата клавиша «вниз», перейти к следующей подсказке
        if(code == 40)
        {
            newTR=document.getElementById("tr"+(++position));
            oldTR=document.getElementById("tr"+(--position));
            // снять выделение с прежней подсказки
            if(position>=0 && position<suggestions-1)
                oldTR.className = "";
            // выделить новую подсказку и обновить ключевое слово
```

```
    if(position < suggestions - 1)
    {
        newTR.className = "highlightrow";
        updateKeywordValue(newTR);
        position++;
    }
    e.cancelBubble = true;
    e.returnValue = false;
    isKeyUpDownPressed = true;
    // прокрутить вниз, если подсказка за пределами окна
    if(position > maxVisiblePosition)
    {
        oScroll = document.getElementById("scroll");
        oScroll.scrollTop += 18;
        maxVisiblePosition += 1;
        minVisiblePosition += 1;
    }
}
else
// если была нажата клавиша «вверх», перейти
// к предыдущей подсказке
if(code == 38)
{
    newTR=document.getElementById("tr"+(--position));
    oldTR=document.getElementById("tr"+(++position));
    // снять выделение с прежней подсказки
    if(position>=0 && position <= suggestions - 1)
    {
        oldTR.className = "";
    }
    // выделить новую подсказку и обновить ключевое слово
    if(position > 0)
    {
        newTR.className = "highlightrow";
        updateKeywordValue(newTR);
        position--;
       // прокрутить вверх, если подсказка за пределами окна
      if(position<minVisiblePosition)
      {
          oScroll = document.getElementById("scroll");
          oScroll.scrollTop -= 18;
          maxVisiblePosition -= 1;
          minVisiblePosition -= 1;
      }
  }
else
if(position == 0)
    position--;
e.cancelBubble = true;
e.returnValue = false;
isKeyUpDownPressed = true;
```

```
    }
}

/* обновляет отображение ключевого слова в соответствии
   с выделенной подсказкой */
function updateKeywordValue(oTr)
{
    // получить ссылку на объект keyword
    var oKeyword = document.getElementById("keyword");
    // получить ссылку на справочную страницу для текущей функции
    var crtLink = document.getElementById("a" +
                          oTr.id.substring(2,oTr.id.length)).toString();
    // заменить символ - на символ _ и отбросить расширение.php
    crtLink = crtLink.replace("-", "_");
    crtLink = crtLink.substring(0, crtLink.length - 4);
    // обновить ключевое слово
    oKeyword.value = unescape(crtLink.substring(phpHelpUrl.length,
                              crtLink.length));
}

/* снимает выделение со всех подсказок */
function deselectAll()
{
    for(i=0; i<suggestions; i++)
    {
        var oCrtTr = document.getElementById("tr" + i);
        oCrtTr.className = "";
    }
}

/* обрабатывает перемещение указателя мыши по списку подсказок */
function handleOnMouseOver(oTr)
{
    deselectAll();
    oTr.className = "highlightrow";
    position = oTr.id.substring(2, oTr.id.length);
}

/* обрабатывает выход указателя мыши за пределы списка с подсказками */
function handleOnMouseOut(oTr)
{
    oTr.className = "";
    position = -1;
}

/* экранирует служебные символы в строке */
function encode(uri)
{
    if (encodeURIComponent)
    {
        return encodeURIComponent(uri);
    }

    if (escape)
```

```
    {
        return escape(uri);
    }
}

/* скрывает слой, содержащий подсказки */
function hideSuggestions()
{
    var oScroll = document.getElementById("scroll");
    oScroll.style.visibility = "hidden";
}

/* выделяет диапазон в текстовом элементе, ссылка на который передается
   в качестве аргумента */
function selectRange(oText, start, length)
{
    // проверить тип броузера - IE или FF
    if (oText.createTextRange)
    {
        // IE
        var oRange = oText.createTextRange();
        oRange.moveStart("character", start);
        oRange.moveEnd("character", length - oText.value.length);
        oRange.select();
    }
    else
    // FF
    if (oText.setSelectionRange)
    {
        oText.setSelectionRange(start, length);
    }
    oText.focus();
}

/* дополняет введенное слово */
function autocompleteKeyword()
{
    //получить ссылку на объект keyword
    var oKeyword = document.getElementById("keyword");
    // сбросить позицию выделенной подсказки
    position=0;
    // снять выделение со всех подсказок
    deselectAll();
    // подсветить выбранную подсказку
    document.getElementById("tr0").className="highlightrow";
    // обновить ключевое слово в соответствии с подсказкой
    updateKeywordValue(document.getElementById("tr0"));
    // выделить часть, которая была дополнена
   selectRange(oKeyword,httpRequestKeyword.length,oKeyword.value.length);
    // запомнить полное ключевое слово вместе с дополненной частью
    autocompletedKeyword=oKeyword.value;
}
```

```
/* отображает сообщение об ошибке */
function displayError(message)
{
    // вывести подробное сообщение об ошибке, если debugMode = true
    alert("Ошибка обращения к серверу! "+
          (debugMode ? "\n" + message : ""));
}
```

11. Код готов к проверке! Откройте в броузере страницу по адресу *http://localhost/ajax/suggest/*. Предположим, что вы ищете описание функции `strstr`. После ввода символа `s` вы увидите список функций, имена которых начинаются с этого символа (рис. 6.3).

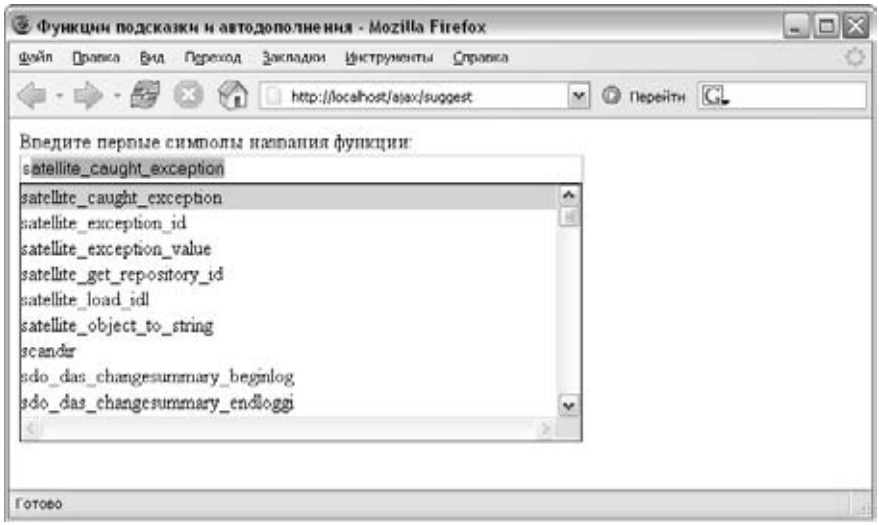

***Рис. 6.3.** В PHP много функций, имена которых начинаются с символа «s»*

12. Отлично, в PHP, оказывается, довольно много функций, имена которых начинаются с символа `s`. Обратите внимание, что имя первой функции дополнило содержимое поля ввода и что мы получили очень длинный список, который можно прокрутить. Теперь введем второй символ слова `strstr`: `t`.
13. Как мы и ожидали, список функций сократился. Найдите в списке имя функции, которую мы ищем, продолжая ввод символов или прокручивая список с помощью клавиш «вверх» и «вниз» на клавиатуре или с помощью мыши. Найдя требуемую функцию, нажмите клавишу ENTER или щелкните по имени функции мышью (рис. 6.4).

Что происходит внутри?

Начнем с файла `index.html`. В этом сценарии интересно то, что область прокрутки может быть реализована средствами DHTML. Области прокрутки описаны здесь: *http://www.dyn-web.com/dhtml/scroll/*. Основ-

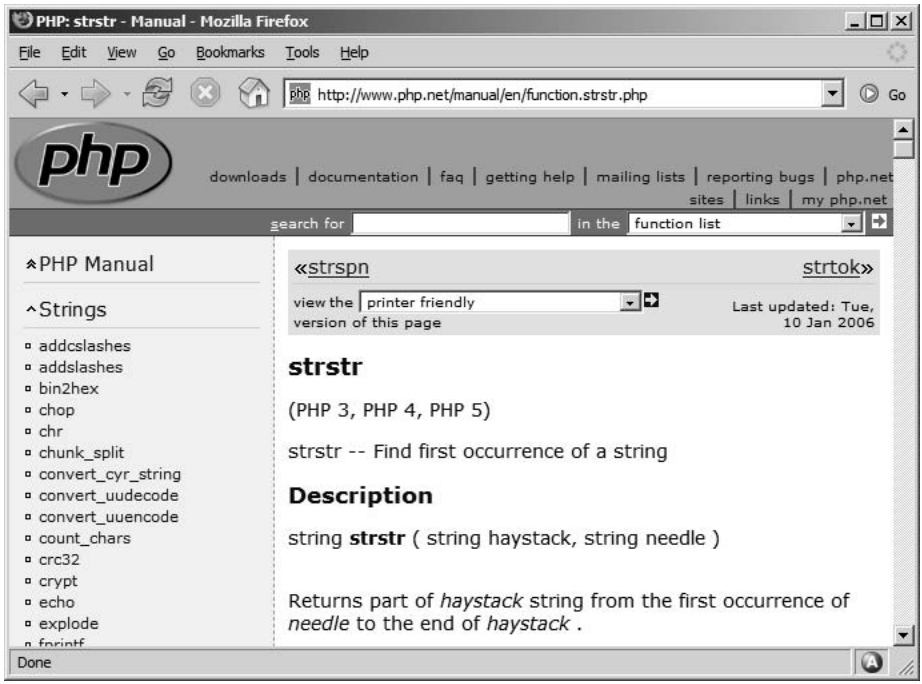

Рис. 6.4. *Описание функции strstr*

ная идея заключается в том, что прокручиваемая область конструируется из двух слоев, один из которых помещается в другой. В нашем примере внутренний слой помещен в элемент `div scroll`.

Внешний слой представлен объектом `scroll`. Он имеет фиксированную высоту и ширину и очень полезное для нас свойство `overflow`. Содержимое текстового поля ограничено его рамками. В некоторых случаях поле может переполняться (`overflow`), в результате чего часть содержимого оказывается за его пределами. В CSS свойство `overflow` определяет поведение элемента при переполнении. Дополнительную информацию по этой теме можно найти по адресу *http://www.w3.org/TR/REC-CSS2/visufx.html*.

Еще один интересный момент заключается в том, как мы центрируем объект по горизонтали. Классический атрибут `align=center` отсутствует в XHTML 1.0, и потому нам нужен какой-нибудь способ обойти это ограничение. Решение было найдено в применении атрибута `margin`. Для того чтобы отцентрировать объект, атрибуту `margin` необходимо присвоить значение `auto`. Если задано корректное значение для `doctype`, Internet Explorer 6 отцентрирует по горизонтали элемент, имеющий значение `auto` в атрибуте `margin`. В противном случае, как и в предыдущих версиях Internet Explorer, этот атрибут будет проигнорирован. Для ранних версий Internet Explorer следует устанавливать в значение

`center` атрибут `text-align` того элемента, который объемлет элемент, нуждающийся в центрировании. Дело в том, что Internet Explorer ошибочно реагирует на атрибут `text-align` не только в строчных, но и во всех блочных элементах.

Событие `onkeyup`, возникающее в элементе ввода, фактически запускает процесс поиска и отображения списка подсказок для текущего ключевого слова. Элемент `div content` отслеживает событие `onclick`, и когда пользователь щелкает мышью за пределами области вывода подсказок, она делается невидимой и появляется только тогда, когда пользователь изменяет ключевое слово.

Функциональность этого приложения практически полностью сосредоточена в сценарии JavaScript, DOM и CSS. Серверная часть чрезвычайно проста и выполняет незначительный объем работ, а вот клиентская, заключенная в сценарии `suggest.js`, намного сложнее. Перечислим функциональные возможности клиента, которые мы реализовали:

1. Когда пользователь начинает ввод ключевого слова, перед ним появляется выпадающий список, который обновляется по мере ввода дополнительных символов или стирания некоторых из них.
2. Первые символы в подсказках, совпадающие с ключевым словом, выводятся жирным шрифтом.
3. Содержимое первой подсказки автоматически дополняет ключевое слово в поле ввода.
4. При перемещении по списку подсказок с помощью клавиш «вверх» и «вниз» ключевое слово дополняется текстом текущей выбранной подсказки.
5. Перемещение указателя мыши по подсказкам не вызывает никаких действий.
6. Нажатие клавиши ENTER или щелчок мышью по подсказке отправляет броузер на страницу, содержащую описание выбранной функции PHP, на сайте *php.net*.
7. Аналогичным образом броузер отправляется на страницу сайта php.net, если пользователь нажмет клавишу ENTER в поле ввода.
8. Когда пользователь щелкает мышью за пределами области вывода подсказок и за пределами поля ввода ключевого слова, список с подсказками делается невидимым.
9. Подсказки буферизуются на стороне клиента.

У нас имеется функция, периодически проверяющая факт изменения ключевого слова. Изменение ключевого слова инициирует запрос HTTP к серверу, в котором передается ключевое слово. В ответ сервер возвращает список подсказок, в котором содержатся имена функций, соответствующих заданному ключевому слову. Броузер клиента отображает подсказки в виде выпадающего списка. Пользователь имеет возможность перемещаться по списку как с помощью курсорных клавиш

«вверх» и «вниз», так и с помощью мыши. При вводе нового или стирании существующего символа происходит обновление списка подсказок. Посмотрев на рисунки в предыдущем разделе и коротко описав весь процесс, перейдем к подробному описанию реализации.

Функция `createXmlHttpRequestObject` применяется нами для создания объекта `XMLHttpRequest`.

Функция `init` не делает ничего существенного, она лишь отключает атрибут `autocomplete` для поля ввода ключевого слова. Делается это с целью предотвратить инициацию механизма автодополнения самого броузера. Поскольку атрибут `"autocomplete"="off"` не соответствует спецификациям XHTML, код HTML считается ошибочным. Этот атрибут был введен Microsoft и позднее заимствован большинством броузеров.

Функция, определяющая изменение ключевого слова, называется `checkForUpdates`. Обнаружив изменения, она запустит процесс обновления списка подсказок. Функция `handleKeyUp` служит для организации перемещения по списку. Более подробно об этой функции мы поговорим чуть позже в этой же главе.

Мы уже говорили о буферизации результатов. Этот прием дает прекрасные результаты в плане оптимизации приложений подобного рода. У нас имеются две функции, которые заняты обслуживанием буфера, – `checkCache` и `addToCache`.

Функция `checkCache` проверяет наличие ключевого слова в буфере. Если его в буфере нет, она пытается отыскать в буфере значений самые длинные префиксы, совпадающие с ключевым словом. Полученные в процессе поиска результаты добавляются в буфер функцией `addToCache`.

Функция `addToCache` добавляет в буфер заданное ключевое слово вместе со списком соответствующих ему подсказок.

Для получения новых подсказок вызывается функция `getSuggestions`. Если текущее ключевое слово уже находится в буфере (функция `checkCache`), мы заполняем список подсказок значениями из буфера. Если предыдущий запрос HTTP еще не завершился, мы сохраняем ключевое слово, чтобы его можно было использовать в следующем обращении к серверу, и устанавливаем таймаут для запуска этой функции. Таким способом мы гарантируем сохранность последнего введенного ключевого слова, по которому мы не можем немедленно послать запрос серверу. Как только текущий запрос будет выполнен, немедленно будет инициирован новый запрос с последним ключевым словом.

Функция `handleGettingSuggestions` контролирует ход исполнения запроса и по его завершении вызывает функцию `updateSuggestions`.

Функция `updateSuggestions` убеждается в необходимости обновления списка подсказок. Если выясняется, что во время исполнения запроса к серверу производилась попытка послать новый запрос, значит, пользователь изменил ключевое слово, и поэтому мы не должны отобра-

жать найденные подсказки, т. к. они более не интересны пользователю. Тем не менее полученные подсказки помещаются в буфер.

Функция `xmlToArray` преобразует коллекцию узлов XML в массив.

Функция, фактически строящая список подсказок, называется `displayResults`. Она принимает в качестве входных аргументов ключевое слово и список функций в виде массива. Прежде всего она помещает полученные аргументы в буфер, таким образом, если нам надо будет повторно отыскать это же ключевое слово, то повторное обращение к веб-серверу не потребуется. Затем функция в цикле обходит массив с подсказками и строит таблицу HTML, заполняя ее гиперссылками. Если массив не содержит подсказок, выводится сообщение о том, что по заданному ключевому слову ничего не было найдено.

Функция `updateKeywordValue` отвечает за дополнение текущего ключевого слова выбранной подсказкой, которая передается в функцию как ссылка на элемент `<tr>`.

Функция `hideSuggections` делает невидимым элемент `div`, содержащий подсказки для текущего ключевого слова.

Функция `deselectAll` снимает выделение с выделенных подсказок.

Функции `handleOnMouseOver` и `handleOnMouseOut` запускаются, когда указатель мыши входит или выходит за пределы области `tr` вывода каждой подсказки. Они влияют на стиль отображения подсказки в соответствии с возникающим событием.

Функция `encode` экранирует специальные символы в строке, которую она получает в качестве входного аргумента. Эта функция вызывается из `getSuggestions` перед обращением к серверу.

Теперь поговорим о функции `handleKeyUp`. Она предназначена для организации перемещения по списку подсказок и перехода на страницу с описанием выбранной функции. Нас интересуют нажатия лишь нескольких клавиш, поэтому все остальные мы попросту игнорируем. Чтобы добиться поставленной цели, нам необходимо гарантировать, что код будет одинаково хорошо работать во всех типах броузеров. А для этого нам придется написать несколько строк, чтобы вы сами могли все увидеть.

Для того чтобы узнать, какие клавиши нажимались, мы должны знать их коды. Объект `event`, передаваемый функции в качестве входного аргумента, имеет свойство `keyCode`, в котором находится код нажатой клавиши. В табл. 6.1 приведен список большинства специальных клавиш с их кодами.

По нажатию клавиши Enter (код 13) выполняется переход на страницу с описанием функции, которой соответствует текущая выбранная подсказка, если таковая существует. По нажатию клавиши «вверх» или «вниз» выделяется подсказка выше или ниже текущей, если это возможно. Кроме того, ключевое слово дополняется вновь выбранной подсказкой. Мы не обрабатываем нажатия остальных клавиш, поскольку

они могут нажиматься для изменения ключевого слова, а эта часть действий с клавиатурой уже обрабатывается функцией `checkForChanges`.

Таблица 6.1. Коды клавиш

Клавиша	Код	Клавиша	Код
Backspace	8	Стрелка «вниз»	40
Tab	9	Print Screen	44
Enter	13	Delete	46
Shift	16	F1	112
Ctrl	17	F2	113
Alt	18	F3	114
Pause/Break	19	F4	115
Caps Lock	20	F5	116
Esc	27	F6	117
Page Up	33	F7	118
Page Down	34	F8	119
End	35	F9	120
Home	36	F10	121
Стрелка «влево»	37	F11	122
Стрелка «вверх»	38	F12	123
Стрелка «вправо»	39		

Еще одна проблема возникает, когда количество подсказок превышает десять. Она связана с прокручиваемой областью `div`. Мы хотим дать пользователю возможность перемещаться по списку подсказок с помощью клавиш «вверх» и «вниз». Если в результате нажатия на клавишу пользователь перемещается на подсказку, которая в настоящий момент невидима, нам необходимо выполнить прокрутку области, чтобы сделать подсказку видимой. Для этого мы сохраняем позиции первой и последней видимых подсказок. У нас получается как бы окно, которое перемещается по результатам в соответствии с заданным направлением движения и текущим выбранным результатом.

Автоматическое дополнение ключевого слова текстом текущей выбранной подсказки реализуется с помощью функций `selectRange` и `autocompleteKeyword`. Добавленная часть отображается как выделенный текст в поле ввода ключевого слова. Метод `select()` выделяет весь текст и потому не может служить для выделения части слова. Для решения этой задачи Internet Explorer предлагает одно решение, а Mozilla/Firefox – другое. Мы уже не в первый раз сталкиваемся с тем, что одни и те же проблемы в разных броузерах решаются по-разному, поэтому надо

с каждым из них разбираться отдельно. В Firefox все довольно просто, поскольку в нем есть функция, которая решает эту задачу, – `setSelectionRange`. Она принимает два параметра: начало выделенной области и ее протяженность. В Internet Explorer то же самое достигается при помощи объекта `TextRange`. Рассмотрим его поближе, потому что он может оказаться для нас полезным.

Объект `TextRange` может решать такие задачи, как поиск текста или его выделение. Текстовые диапазоны позволяют выделять символы, слова и целые предложения в тексте. Каждая из этих трех вариаций представляет собой логический элемент объекта. Для того чтобы использовать такой объект, необходимо выполнить следующие действия:

- Создать текстовый диапазон.
- Применить метод к выделенному тексту.

Можно скопировать текст, выполнить поиск по тексту и выбрать часть текста, что нам и требуется.

Для создания объекта можно вызвать метод `createTextRange` таких элементов, как `body`, `textarea` или `button`.

Каждый объект имеет начало и конец области видимости текста. При создании объекта начало и конец области по умолчанию совпадают с началом и концом всего содержимого элемента. Для изменения области видимости текстового диапазона применяются функции `move`, `moveStart` и `moveEnd`. Каждая из них принимает два параметра, при этом первый определяет логический элемент, а второй – количество элементов. Возвращаемое значение функций – количество элементов, включенных в диапазон. Метод `select` выделяет весь текущий объект. Чтобы получить полное представление о возможностях объекта, посетите страничку MSDN по адресу *http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dyncontent/textrange.asp*.

После приема списка подсказок и добавления их на страницу необходимо дополнить ключевое слово значением первой подсказки. Делается это с помощью функции `selectRange`, описанной выше.

Для обработки ошибок служит функция `displayError`, которая выводит предупреждения, получая текст сообщения в качестве входного параметра.

Итак, мы получили представление о работе клиентской части приложения. Теперь перейдем на сторону сервера.

Серверная часть приложения чрезвычайно проста. Сценарий `suggest.php` принимает от клиента параметр, который представляет собой ключевое слово. Затем вызывается метод `getSuggestions` класса `Suggest`, определение которого находится в файле `suggest.class.php`. Этот метод выполняет поиск подсказок для заданного ключевого слова. После чего сервер возвращает клиенту документ XML, который содержит список функций PHP с именами, соответствующими ключевому слову. Как

видите, основная работа выполняется на стороне клиента, а на стороне сервера практически ничего не происходит.

Справочная система по функциям PHP, реализованная в виде подсказок и функции автодополнения на основе технологии AJAX, определенно оказалась более востребованной, чем мы ожидали. В процессе разработки приложения мы столкнулись с большим количеством трудностей, преодолевая которые, мы нашли решения, которые, будем надеяться, пригодятся нам при разработке других приложений.

Подведение итогов

В начале главы мы дали определение подсказок и функции автодополнения. Мы увидели, насколько широка область их применения – от текстовых редакторов до операционных систем.

Приложение, разработанное в этой главе, предоставляет возможность получения справочной информации по функциям PHP с официального сайта *www.php.net*.

Функциональность нашего приложения напоминает функциональность Google Suggest с многих точек зрения, однако предлагает и некоторые дополнительные возможности.

7

Построение диаграмм в реальном времени средствами SVG и AJAX

Масштабируемая векторная графика (Scalable Vector Graphic – SVG) появилась сравнительно недавно и имеет прекрасный шанс стать одной из самых значимых технологий создания графических изображений для Интернета, как это в свое время произошло с Flash. SVG – это язык описания двухмерных графических изображений. Развитие SVG не связано с разработкой веб-приложений, но эта технология прекрасно вписывается в общую картину, потому что она отлично дополняет возможности, предоставляемые веб-броузерами. Реализаций SVG уже довольно много, стандарты немного запаздывают, но в скором будущем это наверняка будет исправлено.

Технология SVG вошла в список рекомендаций **консорциума World Wide Web (W3C)** в январе 2003 г. Среди громких имен компаний, которые приняли участие в разработке SVG, можно назвать Sun Microsystems, Adobe, Apple, IBM, Kodak, и этот список далеко не полон. Текущей является спецификация SVG 1.2. Среди других рекомендаций W3C, которые скорее всего получат поддержку в большинстве веб-броузеров и операционных систем, можно упомянуть Mobile SVG, SVG Print и sXBL.

Технология SVG обладает следующими основными характеристиками:

- Формат SVG базируется на формате XML, что позволяет легко изменять файлы с помощью обычных текстовых или специализированных редакторов.
- Изображения SVG масштабируемы, т. е. их можно увеличивать, уменьшать и поворачивать, не ухудшая их качества.
- SVG включает в себя и шрифтовые элементы, что позволяет описывать не только графические элементы, но и текстовые надписи.
- SVG включает в себя элементы описания анимации.

- SVG допускает приложения с использованием множества пространств имен XML.
- SVG дает возможность производить манипуляции с деревом документа из сценариев с помощью интерфейса XML DOM и SVG uDOM.

Приведем перечень ссылок, где можно найти материалы по SVG:

- Страница SVG W3C по адресу *http://www.w3.org/Graphics/SVG/*.
- Введение в SVG по адресу *http://www.w3schools.com/svg/svg_intro.asp*.
- Очень полезный список ссылок по тематике SVG по адресу *http://www.svgi.org/*.
- Описание структуры документа SVG по адресу *http://www.w3.org/TR/SVG/struct.html*.
- Справочное руководство по SVG по адресу *http://www.w3schools.com/svg/svg_reference.asp*.
- Примеры создания изображений SVG по адресу *http://www.carto.net/papers/svg/samples/* и *http://svg-whiz.com/samples.html*.

Реализация построения диаграмм в реальном времени[1]

Прежде чем продолжать, убедитесь, что ваш броузер поддерживает SVG. Программный код этого примера проверялся в Firefox 1.5, в Internet Explorer с установленным модулем Adobe SVG Viewer и с программой просмотра SVG-графики Apache Batik. Поддержку броузером SVG можно проверить, пройдя по ссылке, которую можно найти на сайте книги *http://ajaxphp.packtpub.com*.[2]

Броузер Firefox имеет встроенную поддержку SVG. Однако это первая версия SVG, поэтому ее реализация еще далека от совершенства и имеет недостаточно высокую производительность, и это обстоятельство надо учитывать при создании программ.

Чтобы добавить поддержку SVG в Internet Explorer, потребуется установить внешний модуль SVG. В наших тестах мы использовали модуль от Adobe, который можно скачать по адресу *http://www.ado-*

[1] Здесь и в других местах автор достаточно вольно интерпретирует термин «реальное время», но его вполне можно понять: в контексте веб-технологий он означает «в темпе поступления реакций пользователя», что существенно отличается от строгого термина realtime, например в промышленных системах управления. – *Примеч. науч. ред.*

[2] Можно также зайти на страницу *http://svg-whiz.com/svg/animation/TheDominoEffect.svg* (или попытаться просмотреть любой образец на странице *http://svg-whiz.com/samples.html*) и убедиться, что количество броузеров, поддерживающих SVG, сейчас невелико (IE6). – *Примеч. науч. ред.*

be.com/svg/viewer/install.main.html. Для установки надо загрузить небольшой файл с именем `SVGView.exe` и запустить его на исполнение. При первой загрузке страницы, содержащей графику в формате SVG, надо подтвердить согласие с условиями лицензионного соглашения.

Наконец, мы проверяли работоспособность приложения с помощью программы просмотра SVG-графики Apache Batik.[1] В этом случае файлы SVG надо загружать напрямую, поскольку эта программа не поддерживает загрузку файлов формата HTML, которые в свою очередь должны загружать сценарии SVG. (Кроме того, Batik представляет собой отличное средство просмотра структуры DOM; он в удобной форме отображает узлы SVG в виде иерархического дерева.)

В упражнении из этой главы мы создадим простое приложение построения диаграмм, которое будет асинхронно получать данные от сценария PHP. Вообще данные могут иметь любую природу, но мы взяли простой алгоритм, который генерирует случайные числа. На рис. 7.1 показан внешний вид нашего приложения.

Диаграмма на рис. 7.1 хранится в статическом SVG-файле `temp.svg`. Этот файл был сгенерирован работающей программой; это не снимок экрана с работающим приложением. Сценарий с именем `temp.svg` хранится вместе с исходными текстами примеров к этой главе, и его можно скачать с сайта книги. Его можно загрузить прямо в броузер (необязательно через веб-сервер), предварительно убедившись, что он понимает формат SVG.

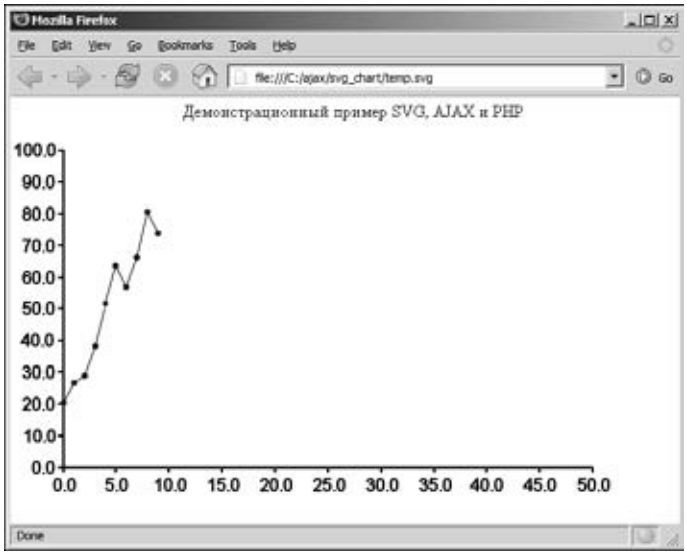

Рис. 7.1. *Диаграмма SVG*

[1] *http://xmlgraphics.apache.org/batik/. – Примеч. науч. ред.*

Для начала заглянем внутрь файла temp.svg и посмотрим, что нам придется генерировать из сценария JavaScript. Обратите внимание: сценарий SVG может быть сгенерирован как на стороне клиента, так и на стороне сервера. В нашем приложении сервер будет генерировать только пары координат, а код JavaScript на стороне клиента будет по этим координатам строить и выводить SVG.

Взгляните на урезанную версию файла temp.svg:

```
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" onload="init(evt)">
  <a xlink:href="http://ajaxphp.packtpub.com">
    <text x="200" y="20">
      Демонстрационный пример SVG, AJAX и PHP
    </text>
  </a>
  <!-- Все элементы диаграммы приведены к началу координат в точке 50, 50 -->
  <g transform="translate(50, 50)">
    <!-- Группируем все элементы осей (линии и текстовые узлы) -->
    <g>
      <!-- Path чертит оси и ставит на них метки -->
      <path stroke="black" stroke-width="2" d=" ... определение path ..."/>
      <!-- текстовые узлы, описывающие метки на осях координат -->
      <text x="-10" y="322" stroke="black">0.0</text>
      ...
      ... остальные текстовые узлы, описывающие метки на осях координат
      ...
    </g>
    <!-- Рисование линий между точками - узлами диаграммы -->
    <path stroke="black" stroke-width="1" fill="none" d="...определение ..."/>
    <!-- Рисование точек - узлов диаграммы в виде кружков синего цвета -->
    <circle cx="00" cy="239.143" r="3" fill="blue" />
    ...
    ... остальные узлы с точками, в виде кружков синего цвета
    ...
  </g>
</svg>
```

Рассмотрим этот фрагмент поближе и опишем все элементы диаграммы. Формат SVG поддерживает понятие групп элементов. Границы группы обозначаются тегами <g> и </g>. В файле temp.svg мы имеем две группы: первая приведена к координатам (50, 50) и включает в себя все элементы диаграммы. Вторая группа вложена в первую и содержит описание осей координат и текстовых меток на осях.

Формат SVG поддерживает большое количество типов элементов, среди которых есть элементы описания анимации (да, SVG необычайно мощная технология). В нашем примере задействованы лишь самые основные: path (рисует оси координат и линии на диаграмме), text (рисует текстовые метки на осях координат и динамически выводит координаты узлов диаграммы, когда указатель мыши находится над ними; эта особенность не показана во фрагменте кода) и circle (рисует синие точки, которые представляют узлы диаграммы).

Описания этих элементов можно найти на сайте *www.w3schools.com*:

- *http://www.w3schools.com/svg/svg_path.asp*
- *http://www.w3schools.com/svg/svg_circle.asp*
- *http://www.w3schools.com/svg/svg_text.asp*

Порядок вычерчивания линии описывается определением элемента `path`. Полный код элемента `path`, который рисует линии, соединяющие узлы диаграммы, выглядит следующим образом:

```
<!-- Draw the lines between chart nodes -->
<path stroke="black" stroke-width="1" fill="none"
  d="M0,239.143 L10,220.286 L20,213.429 L30,185.571 L40,145.714
    L50,108.857 L60,129 L70,101.143 L80,58.2857 L90,78.4286"/>
```

В представленный выше фрагмент кода не попала часть сценария, описывающая события `mouseover` и `mouseout`, которые возникают в узлах диаграммы. В нашем примере событие `mouseover` (возникает, когда вы помещаете указатель мыши над узлом) вызывает функцию JavaScript, которая выводит значения координат узла. По событию `mouseout` текст исчезает. Вы можете увидеть эту особенность в действии на рис. 7.2, где показано работающее приложение.

Примечание

Для того чтобы в броузере Firefox получить содержимое диаграммы в формате SVG, сгенерированной динамически, в любой заданный момент времени, щелкните правой кнопкой мыши по диаграмме, выберите пункт Select All (Выбрать все), затем снова щелкните по диаграмме правой кнопкой мыши и выберите пункт меню View Selection Source (Просмотр исходного кода выделенного фрагмента).

***Рис. 7.2.** Приложение построения диаграмм SVG в действии*

Время действовать – создание диаграмм в реальном времени

1. Начнем с создания нового подкаталога с именем `svg_chart` в каталоге `ajax`.
2. В каталоге `svg_chart` создайте файл с именем `index.html` и добавьте в него следующий код:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html>
  <head>
    <title>
      Рисование диаграмм в реальном времени средствами SVG и AJAX
    </title>
  </head>
  <body>
    <embed src="chart.svg" width="600" height="450" type="image/svg+xml"
/>
  </body>
</html>
```

3. Создайте файл с именем `chart.svg` и добавьте в него следующий код:

```
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" onload="init(evt)">
  <script type="text/ecmascript" xlink:href="ajaxRequest.js"/>
  <script type="text/ecmascript" xlink:href="realTimeChart.js"/>
  <a xlink:href="http://ajaxphp.packtpub.com">
    <text x="200" y="20">
      Демонстрационный пример SVG, AJAX и PHP
    </text>
  </a>
</svg>
```

4. Создайте файл с именем `ajaxRequest.js` и добавьте в него следующий код:

```
// ссылка на объект XMLHttpRequest
var xmlHttp = null;
// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // ссылка на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
```

```
    {
        // подразумевается броузер IE6 или его более старая версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// инициирует запрос AJAX
function ajaxRequest(url, callback)
{
    // сохранить ссылку на функцию обратного вызова, которая будет
    // вызываться по прибытии ответа сервера
    var innerCallback = callback;
    // создать объект XMLHttpRequest при первом обращении к методу
    if (!xmlHttp) xmlHttp = createXmlHttpRequestObject();
    // если соединение не занято, инициировать новый запрос
    if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
    {
        xmlHttp.onreadystatechange = handleGettingResults;
        xmlHttp.open("GET", url, true);
        xmlHttp.send(null);
    }
    else
        // если соединение занято, повторить попытку через 1 секунду
        setTimeout("ajaxRequest(url,callback)", 1000);

    // вызывается при изменении состояния запроса
    function handleGettingResults()
    {
        // вперед можно двигаться только если транзакция завершена
        if (xmlHttp.readyState == 4)
        {
            // статус HTTP = 200 говорит о том, что транзакция
```

```
                // успешно завершена
                if (xmlHttp.status == 200)
                {
                    // вызвать функцию обратного вызова
                    // и передать ей ответ сервера
                    innerCallback(xmlHttp.responseText)
                }
                else
                {
                    // вывести сообщение об ошибке
                    alert("Невозможно соединиться с сервером ");
                }
            }
        }
    }
```

5. **Основной объем работ на стороне клиента выполняется сценарием** `realTimeChart.js`**:**

```
// пространство имен SVG
var svgNS = "http://www.w3.org/2000/svg";
// ссылка на документ SVG
var documentSVG = null;
// ссылка на корневой элемент <g>, в котором сгруппированы
// все элементы диаграммы
var chartGroup = null;
// как часто запрашивать новые данные у сервера?
var updateInterval = 1000;
// координаты (в пикселах), используемые для приведения
// начала координат диаграммы
var x = 50, y = 50;
// размеры диаграммы (в пикселах)
var height = 300, width = 500;
// начало координат диаграммы
var xt1 = 0, yt1 = 0;
// максимальные значения по осям
var xt2 = 50, yt2 = 100;
// количество рисок на горизонтальной и вертикальной осях
var xDivisions = 10, yDivisions = 10;
// первоначальная ширина и высота текста (впоследствии пересчитывается)
var defaultTextWidth = 30, defaultTextHeight = 20;
// ссылки на элементы диаграммы для пересчета местоположения
var xIndexes = new Array(xDivisions + 1);
var yIndexes = new Array(yDivisions + 1);
// текстовый элемент, который отображает координаты
// выбранного узла диаграммы
var currentNodeInfo;
// последние значения, сгенерированные сервером
var lastX = -1, lastY = -1;
// разделяемые элементы svg
var chartGroup, dataGroup, dataPath;

// инициализирует диаграмму
```

```
function init(evt)
{
    /**** Подготовить группу, которая будет содержать всю диаграмму ****/
    // получить ссылку на документ SVG
    documentSVG = evt.target.ownerDocument;
    // создать элемент <g>, в котором будут
    // сгруппированы все элементы диаграммы
    chartGroup = documentSVG.createElementNS(svgNS, "g");
    chartGroup.setAttribute("transform", "translate(" + x + " " + y + ")");

    /**** Подготовить группу, в которой будут находиться оси X и Y ****/
    axisGroup = documentSVG.createElementNS(svgNS, "g");
    // создать ось X как элемент <path>
    axisPath = documentSVG.createElementNS(svgNS, "path");
    // линия оси будет черной и толщиной в 2 пиксела
    axisPath.setAttribute("stroke", "black");
    axisPath.setAttribute("stroke-width", "2");

    /**** Создать риски для осей X и Y ****/
    // создать текст определения рисок для оси X
    pathText = "M 0 " + height;
    // добавить риски на ось X (последняя риска добавляется отдельно)
    for (var i = 0; i <= xDivisions; i++)
        pathText += "l 0 5 l 0 -5 " +
                ((i == xDivisions) ? "" : ("l " + width/xDivisions + " 0"));
    // создать текст определения рисок для оси Y
    pathText += "M 0 " + height;
    // добавить риски на ось Y (последняя риска добавляется отдельно)
    for (var i = 0; i <= yDivisions; i++)
        pathText += "l -5 0 l 5 0 " +
                 ((i == yDivisions) ? "" : ("l 0 -" + height / yDivisions));
    // добавить текст определения (атрибут <d>) в элемент path
    axisPath.setAttribute("d", pathText);
    // добавить элемент path в группу осей
    axisGroup.appendChild(axisPath);

    /**** Создать текстовые метки для осей X и Y ****/
    // добавить текстовые метки к оси X
    for (var i = 0; i <= xDivisions; i++)
    {
        // создать элемент <text> для риски
        t = documentSVG.createElementNS(svgNS, "text");
        // запомнить ссылку на элемент
        xIndexes[i] = t;
        // добавить текст в элемент <text>
        t.appendChild(documentSVG.createTextNode(
             (xt1 + i * ((xt2 - xt1) / xDivisions)).toFixed(1)));
        // установить атрибуты X и Y элемента <text>
        t.setAttribute("x", i * width / xDivisions - defaultTextWidth / 2);
        t.setAttribute("y", height + 30 + defaultTextHeight);
        // при первой загрузке диаграммы, текстовые надписи
        // должны быть невидимы
        t.setAttribute("stroke", "white");
```

```
            // добавить элемент <text> в группу с осями координат
            axisGroup.appendChild(t);
        }
        // добавить текстовые метки к оси Y
        for (var i = 0; i <= yDivisions; i++)
        {
            // создать элемент <text> для риски
            t = documentSVG.createElementNS(svgNS, "text");
            // запомнить ссылку на элемент
            yIndexes[i] = t;
            // добавить текст в элемент <text>
            t.appendChild(documentSVG.createTextNode(
                     (yt1 + i * ((yt2 - yt1) / yDivisions)).toFixed(1)));
            // установить атрибуты X и Y элемента <text>
            t.setAttribute("x", -30 -defaultTextWidth);
            t.setAttribute("y", height - i * height / yDivisions
                            + defaultTextHeight / 2);
            // при первой загрузке диаграммы,
            // текстовые надписи должны быть невидимы
            t.setAttribute("stroke", "white");
            // добавить элемент <text> в группу с осями координат
            axisGroup.appendChild(t);
        }
        // добавить группу с осями в диаграмму
        chartGroup.appendChild(axisGroup);

        /**** Подготовка элемента <path>, который будет отображать данные ****/
        dataPath = documentSVG.createElementNS(svgNS, "path");
        dataPath.setAttribute("stroke", "black");
        dataPath.setAttribute("stroke-width", "1");
        dataPath.setAttribute("fill", "none");
        // добавить элемент path в группу диаграммы
        chartGroup.appendChild(dataPath);

        /**** Заключительные шаги инициализации ****/
        // добавить группу с диаграммой в документ SVG
        documentSVG.documentElement.appendChild(chartGroup);
        // это необходимо для корректного отображения текстовых узлов в Firefox
        setTimeout("refreshXYIndexes()", 500);
        // инициировать отправку повторяющихся запросов серверу
        setTimeout("updateChart()", updateInterval);
    }

    // эта функция перерисовывает текстовые метки на осях и делает их видимыми
    // (это необходимо для корректного позиционирования текста в Firefox)
    function refreshXYIndexes()
    {
        // перерисовать текстовые метки оси X
        for (var i = 0; i <= xDivisions; i++)
            if (typeof xIndexes[i].getBBox != "undefined")
                try
                {
                    textWidth = xIndexes[i].getBBox().width;
```

```
                textHeight = xIndexes[i].getBBox().height;
                xIndexes[i].setAttribute("x", i*width/xDivisions
                                         - textWidth/2);
                xIndexes[i].setAttribute("y", height + 10 + textHeight);
                xIndexes[i].setAttribute("stroke", "black");
            }
            catch(e) {}
    // перерисовать текстовые метки оси Y
    for (var i = 0; i <= yDivisions; i++)
        if (typeof yIndexes[i].getBBox != "undefined")
            try
            {
                twidth = yIndexes[i].getBBox().width;
                theight = yIndexes[i].getBBox().height;
                yIndexes[i].setAttribute("y", height-i*height/yDivisions
                                         + theight/2);
                yIndexes[i].setAttribute("x", -10 -twidth);
                yIndexes[i].setAttribute("stroke", "black");
            }
            catch(e) {}
}

// вызывается, когда указатель мыши оказывается над узлом диаграммы
// и отображает его координаты
function createPointInfo(x, y, whereX, whereY)
{
    // не отображать одновременно координаты более чем одного узла
    if (currentNodeInfo) removePointInfo();
        // создать текстовый элемент
        currentNodeInfo = documentSVG.createElementNS(svgNS, "text");
        currentNodeInfo.appendChild(documentSVG.createTextNode("("+x+",
                                                               "+y+")"));
    // установить координаты
    currentNodeInfo.setAttribute("x", whereX.toFixed(1));
    currentNodeInfo.setAttribute("y", whereY - 10);
    // добавить элемент в группу
    chartGroup.appendChild(currentNodeInfo);
}

// удаляет текстовый элемент, в котором выводятся координаты узла диаграммы
function removePointInfo()
{
    chartGroup.removeChild(currentNodeInfo);
    currentNodeInfo = null;
}

// рисует новую точку на диаграмме
function addPoint(X, Y)
{
    // сохранить эти значения для использования в последующем
    lastX = X;
    lastY = Y;
    // начать заново (перезагрузить страницу) после того, как было получено
```

```
    // последнее значение
    if (X == xt2)
        window.location.reload(false);
    // рассчитать координаты нового узла
    coordX = (X - xt1) * (width / (xt2 - xt1));
    coordY = height - (Y - yt1) * (height / (yt2 - yt1));
    // нарисовать узел на диаграмме как синий кружок
    var circle = documentSVG.createElementNS(svgNS, "circle");
    circle.setAttribute("cx", coordX);   // координата X
    circle.setAttribute("cy", coordY);   // координата Y
    circle.setAttribute("r", 3);         // радиус
    circle.setAttribute("fill", "blue"); // цвет
    circle.setAttribute("onmouseover",
                        "createPointInfo(" + X + "," +
                        Y + "," + coordX + "," + coordY + ")");
    circle.setAttribute("onmouseout", "removePointInfo()");
    chartGroup.appendChild(circle);
    // добавить в диаграмму новый отрезок,
    // соединяющий новый узел с предыдущим
    current = dataPath.getAttribute("d"); // текущее определение path
    // обновить определение path
    if (!current || current == "")
        dataPath.setAttribute("d", " M " + coordX + " " + coordY);
    else
       dataPath.setAttribute("d", current + " L " + coordX + " " + coordY);
}

// инициирует асинхронный запрос на получение новых данных
function updateChart()
{
    // собрать строку запроса
    param = "?lastX=" + lastX + ((lastY != -1) ? "&lastY=" + lastY : "");
    // послать запрос каким-либо способом
    if (window.getURL)
        // поддерживается Adobe SVG Viewer и Apache Batik
        getURL("svg_chart.php" + param, handleResults);
    else
        // поддерживается Mozilla, реализован в ajaxRequest.js
        ajaxRequest("svg_chart.php" + param, handleResults);
}

// функция обратного вызова, которая читает данные, принятые от сервера
function handleResults(data)
{
    // получить ответ сервера
    if (window.getURL)
        responseText = data.content;
    else
        responseText = data;
    // разделить на пару координат X и Y
    var newCoords = responseText.split(",");
    // нарисовать в этих координатах новый узел
    addPoint(newCoords[0], newCoords[1]);
```

```
        // повторить последовательность действий
        setTimeout("updateChart()", updateInterval)
    }
```

6. **В завершение создайте файл серверного сценария с именем** `svg_chart.php`**:**

```
<?php
// инициализация переменных
$maxX = 50;  // максимум по оси X
$maxY = 100; // максимум по оси Y
$maxVariation = $maxY / 7; // максимальное отклонение по оси Y за один шаг
// клиент сообщает последнее полученное значение X (по умолчанию -1)
if (isset($_GET['lastX']))
    $lastX = $_GET['lastX'];
else
    $lastX = -1;
// клиент сообщает последнее полученное значение Y
// (по умолчанию – случайное число)
if (isset($_GET['lastY']))
    $lastY = $_GET['lastY'];
else
    $lastY = rand(0, $maxY);
// найти новое случайное число
$randomY = (int) ($lastY + $maxVariation - rand(0, $maxVariation*2));
// убедиться, что новое значение Y находится в диапазоне от 0 до $maxY
while ($randomY < 0) $randomY += $maxVariation;
while ($randomY > $maxY) $randomY -= $maxVariation;
// записать пару значений в строку
$output = $lastX + 1 . ',' . $randomY;
// очистить выходной буфер
if(ob_get_length()) ob_clean();
// предотвратить возможность кэширования страницы броузером
header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); // время в прошлом
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
// отправить результаты клиенту
echo $output;
?>
```

7. **Откройте страницу** *http://localhost/ajax/svg_chart/* **и полюбуйтесь на вашу собственную, совершенно неповторимую диаграмму!**

Что происходит внутри?

Рассмотрим коротко наиболее важные участки кода, начиная с серверной части. Сценарий `svg_chart.php` **вызывается асинхронно и генерирует новую пару координат (**`X`**,** `Y`**) для клиента. Предполагается, что клиент сообщит серверу значения координат, сгенерированные перед этим, и на их основе будет сгенерирована новая пара. Этот алгоритм достаточно хорошо моделирует реальную ситуацию. Ранее сгенерированные координаты передаются серверу методом** `GET` **в виде двух параметров –**

lastX и lastY. Для того чтобы проверить правильность функционирования серверного сценария, не привлекая остальную часть приложения, можно загрузить страницу *http://localhost/ajax/svg_chart/svg_chart.php?lastX=5&lastY=44* (рис. 7.3).

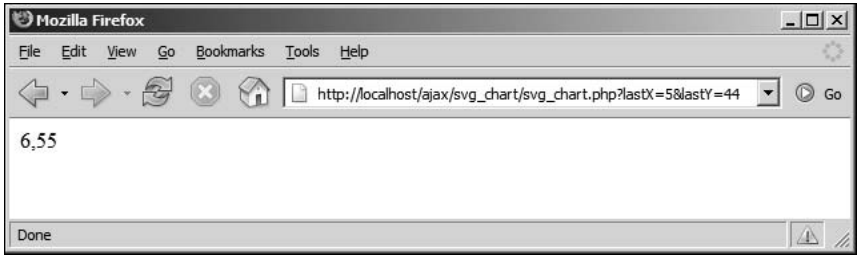

Рис. 7.3. *Сервер сгенерировал новую пару координат для клиента*

На стороне клиента все начинается с файла index.html, который очень прост, поскольку его основное предназначение состоит в том, чтобы загрузить файл SVG. Лучший на текущий момент способ сделать это – воспользоваться элементом <embed>, который не поддерживается W3C (можно взять и элементы <object> и <iframe>, но они обладают некоторыми недостатками; см. *http://www.w3schools.com/svg/svg_inhtml.asp*):

```
<body>
    <embed src="chart.svg" width="600" height="450" type="image/svg+xml" />
</body>
```

Теперь перейдем к файлу chart.svg. Он не так велик, потому что использует функциональные возможности, предоставляемые файлами JavaScript (обратите внимание на событие onload), но он включает в себя заголовок диаграммы:

```
<svg width="100%" height="100%" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" onload="init(evt)">
  <script type="text/ecmascript" xlink:href="ajaxRequest.js"/>
  <script type="text/ecmascript" xlink:href="realTimeChart.js"/>
  <a xlink:href="http://ajaxphp.packtpub.com">
    <text x="200" y="20">
      Демонстрационный пример SVG, AJAX и PHP
    </text>
  </a>
</svg>
```

Файл chart.svg содержит ссылки на два файла с кодом JavaScript:

Сценарий ajaxRequest.js содержит реализацию механизма асинхронных запросов HTTP на основе объекта XMLHttpRequest. Этот механизм позволяет сценарию realTimeChart.js получать новые данные от сервера, когда работа выполняется под управлением броузера Firefox. В случае внешних модулей Adobe SVG Viewer и Apache Batik мы используем реализацию, предоставляемую этими модулями, – метод

getURL. Более подробные сведения можно почерпнуть из исходного кода функции updateChart в файле realTimeChart.js.

Код в ajaxRequest.js выдержан в несколько ином стиле по сравнению с тем, что мы видели до сих пор в этой книге. Вы должны уяснить главное:

- Вместо того чтобы создавать новые объекты XMLHttpRequest для каждого запроса, как это было сделано в некоторых примерах, все запросы HTTP отправляются через единственный экземпляр объекта XMLHttpRequest. Тем самым мы гарантируем, что ответы сервера будут приниматься в том же порядке, в каком отправлялись запросы, что очень важно для данного приложения (как и для любого другого приложения, где важен порядок следования ответов). Если объект занят обработкой предыдущего запроса, наш сценарий ждет одну секунду и повторяет попытку. Эта методика, кроме всего прочего, позволяет экономно расходовать ресурсы сервера.
- Метод ajaxRequest() принимает в качестве параметров URL сервера и ссылку на функцию обратного вызова, которая должна быть вызвана при получении ответа от сервера. Такой способ реализации очень удобен, когда необходимо организовать гибкий доступ к одной и той же функциональности из различных источников.
- Определение метода handleGettingResults() находится внутри метода ajaxRequest. Это одна из особенностей JavaScript, которая позволяет эмулировать возможности ООП. До сих пор мы не использовали эти особенности, потому что, на наш взгляд, они могут принести реальную выгоду только при создании больших приложений, а кроме того, было бы преждевременно демонстрировать их программистам, которые стоят в начале длинного и извилистого пути познания и не знакомы с приемами ООП. По мере обретения опыта вы обнаружите, что эта особенность в состоянии облегчить разработку приложений.

Сценарий realTimeChart.js содержит всю функциональность, необходимую для создания диаграмм в формате SVG. Код снабжен подробными комментариями по всем специфическим приемам, которые в нем применялись. Здесь же мы дадим краткое описание каждого из методов:

- Метод init() вызывается при загрузке страницы и выполняет действия по инициализации диаграммы. Он генерирует код SVG, который вычерчивает оси координат и строит начальную структуру всей диаграммы. Изначально цифровые метки на осях координат имеют белый цвет, чтобы их не было видно. Это связано с недостатками реализации механизма SVG в броузере Firefox, не позволяющими точно рассчитать размер и положение символов текста до того, как они будут отображены. Вариант с предопределенными значениями нам не подходит, поскольку диаграмма может настраиваться в процессе работы и в окончательном варианте оси могут выводиться в ином масштабе. Для преодоления этих недостатков

метод `init()` запускает метод `refreshXYIndexes()` через полсекунды с помощью функции `setTimeout()`.

- Метод `refreshXYIndexes()` рассчитывает корректное положение текстовых меток на осях координат даже несмотря на недостатки, имеющиеся в Firefox 1.5. После этого он устанавливает новые координаты и изменяет цвет шрифта с белого на черный, чтобы сделать метки видимыми.
- Метод `createPointInfo()` вызывается из функции `onmouseover` для отображения координат узла диаграммы, находящегося под указателем мыши.
- Метод `removePointInfo()` вызывается из функции `onmouseout`, чтобы стереть выведенные предыдущим методом координаты узла.
- Метод `updateChart()` инициирует асинхронный запрос. Если имеется возможность вызвать метод `getURL` (доступен при использовании внешних модулей Adobe SVG и Apache Batik), то для отправки запроса применяется этот метод. В противном случае вызывается метод `ajaxRequest` (из сценария `ajaxRequest.js`). В теле запроса методом `GET` серверу передается последняя полученная клиентом пара координат, на основе которой сервер вычисляет новые значения.
- Метод `handleResults()` представляет собой функцию обратного вызова, которая вызывается при получении ответа от сервера. Из ответа извлекаются (опять же, методом, зависящим от реализации SVG) новые координаты, сгенерированные сервером, и передаются методу `addPoint()`.
- Метод `addPoint()` принимает новую пару координат и на их основе создает новый узел диаграммы. Сами координаты сохраняются, поскольку их необходимо будет передать серверу в следующем запросе. Сервер (опять на основе этих координат) рассчитает новые значения для клиента, для чего применяется достаточно простой алгоритм: с каждым новым запросом координата X увеличивается на 1, а в качестве значения Y выбирается случайное число, но с учетом предыдущего значения координаты Y.

Подведение итогов

Независимо от того, нравится вам SVG или нет (особенно в свете недавних войн SVG и Flash), вы должны признать, что эта технология позволяет реализовать достаточно мощную функциональность в веб-страницах. Почувствовав на практике ее возможности, вы уже будете знать, в каких проектах ее можно использовать. Если вы всерьез заинтересовались SVG, поищите визуальные редакторы, которые могут значительно облегчить создание SVG. Кроме того, можно подумать о покупке одной из многочисленных книг, посвященных SVG.

8

Таблицы в AJAX

Представление данных в табличной форме всегда было одной из областей, где веб-приложения серьезно проигрывали обычным программам. Фактически при переходе от одной части таблицы к другой или при изменении отдельных ее частей приходилось полностью загружать новую страницу, что, безусловно, снижало качество приложения с точки зрения простоты и удобства использования. С технической точки зрения полная перезагрузка страницы также нехороша, поскольку неоправданно увеличивает трафик.

Но теперь вы уже знаете, что это затруднение преодолимо. Содержимое страницы можно обновлять с помощью технологии AJAX. Она позволяет создавать страницы с красивым дизайном, которые не будут даже «моргать», поскольку обновляться будут только изменяемые данные, а не вся страница целиком.

Впервые в этой книге мы будем применять **расширяемый язык таблиц стилей для преобразований (Extensible Stylesheet Language Transformation – XSLT)** и **язык адресации частей документа XML (XML Path Language – XPath)** для генерации данных, передаваемых клиенту. Языки XSLT и XPath входят в семейство расширяемого языка стилей (XSL). XSLT позволяет описать правила преобразования документа XML в какой-либо другой формат, а XPath представляет собой очень мощный язык построения запросов, который дает возможность производить поиск и извлечение данных из документов XML. XSLT может применяться для построения графического интерфейса в веб-приложениях, в этом случае он позволяет реализовать очень гибкую архитектуру, в которой сервер отдает данные в формате XML, а затем эти данные преобразуются в формат HTML с помощью преобразования XSL. Введение в XSL можно найти в приложении C по адресу *http://ajax-php.packtpub.com*, а отличное описание языка – по адресу *http://en.wikipedia.org/wiki/Extensible_Stylesheet_Language*.

Примечание

Обратите внимание, что преобразование XSL может выполняться как на стороне клиента, так и на стороне сервера. Реализация преобразований, описываемая в данной главе, размещается на стороне клиента. Она не требует наличия какого-либо специального программного обеспечения на стороне сервера, но накладывает некоторые ограничения на клиента. В главе 9 мы посмотрим, как выполняются преобразования на стороне сервера с помощью PHP. В этом случае вам потребуется добавить в PHP возможность выполнения преобразований, зато такое решение будет способно работать с любым клиентом, т. к. клиент будет получать код HTML, уже готовый к отображению.

В этой главе мы будем применять:

- XSL – для создания таблиц данных в представлении HTML на основе данных, принимаемых с сервера в формате XML.
- AJAX – для реализации редактируемых таблиц данных. Пользователь получит возможность переключаться между страницами таблицы со списком продуктов без необходимости перезагрузки всей страницы.

Реализация таблиц данных на стороне клиента средствами AJAX и XSLT

В этом примере мы создадим таблицу данных с возможностью редактирования. Данные для заполнения таблиц любезно предоставлены интернет-магазином *http://www.the-joke-shop.com*.

На рис. 8.1 показана вторая страница со списком продуктов, а на рис. 8.2 – как эта таблица выглядит после щелчка по ссылке Edit (Редактировать), где видно, что строка с названием продукта перешла в режим редактирования.

Здесь имеется значительный объем данных, генерируемых динамически, и это даст нам хорошую возможность познакомиться с XSLT.

Сначала напишем код, чтобы у вас на руках имелось работающее приложение, а затем прокомментируем его. Программа будет состоять из следующих файлов:

- `grid.php`
- `grid.class.php`
- `error_handler.php`
- `config.php`
- `grid.css`
- `index.html`
- `grid.xsl`
- `grid.js`

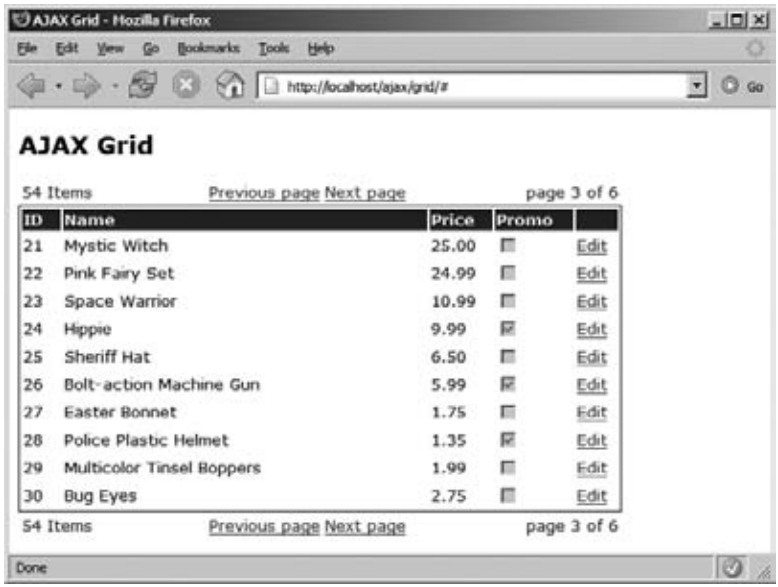

Рис. 8.1. *Таблица с данными, реализованная на базе технологии AJAX*

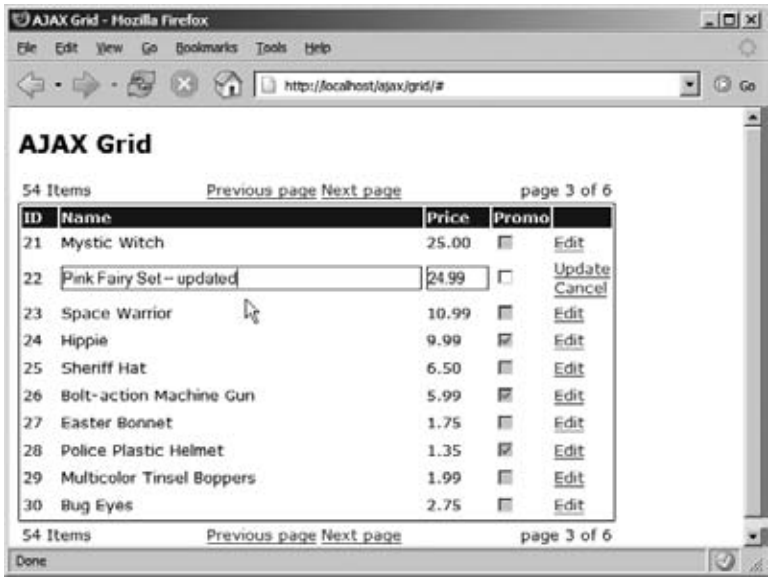

Рис. 8.2. *Таблица с данными в режиме редактирования*

Время действовать – таблицы в AJAX

1. Начнем с подготовки базы данных для этого упражнения. Для работы приложения нам потребуется таблица со списком продуктов.

Вы можете воспользоваться сценарием `product.sql`, загрузив его с сайта книги, или ввести данные вручную (фрагмент кода, приведенный ниже, содержит названия только первых десяти наименований продуктов, а для создания таблицы с полным списком надо загрузить сценарий).

```
CREATE TABLE product
(
    product_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL DEFAULT '',
    price DECIMAL(10,2) NOT NULL DEFAULT '0.00',
    on_promotion TINYINT NOT NULL DEFAULT '0',
    PRIMARY KEY (product_id)
);

INSERT INTO product(name, price, on_promotion)
            VALUES('Santa Costume', 14.99, 0);
INSERT INTO product(name, price, on_promotion)
            VALUES('Medieval Lady', 49.99, 1);
INSERT INTO product(name, price, on_promotion)
            VALUES('Caveman', 12.99, 0);
INSERT INTO product(name, price, on_promotion)
            VALUES('Costume Ghoul', 18.99, 0);
INSERT INTO product(name, price, on_promotion)
            VALUES('Ninja', 15.99, 0);
INSERT INTO product(name, price, on_promotion)
            VALUES('Monk', 13.99, 0);
INSERT INTO product(name, price, on_promotion)
            VALUES('Elvis Black Costume', 35.99, 0);
INSERT INTO product(name, price, on_promotion)
            VALUES('Robin Hood', 18.99, 0);
INSERT INTO product(name, price, on_promotion)
            VALUES('Pierot Clown', 22.99, 1);
INSERT INTO product(name, price, on_promotion)
            VALUES('Austin Powers', 49.99, 0);
```

2. Создайте в каталоге `ajax` подкаталог с именем `grid`.
3. В первую очередь мы представим сценарии серверной части приложения. В каталоге `grid` создайте файл с именем `grid.php`, который будет отвечать на асинхронные запросы клиента:

```
<?php
// загрузить модуль обработки ошибок и класс Grid
require_once('error_handler.php');
require_once('grid.class.php');
// в качестве параметра 'action' может быть только
// либо FEED_GRID_PAGE, либо UPDATE_ROW
if (!isset($_GET['action']))
{
    echo 'Server error: клиент не передал команду.';
    exit;
}
```

```
else
{
    // сохранить выполняемую команду в переменной $action
    $action = $_GET['action'];
}
// создать экземпляр класса Grid
$grid = new Grid($action);
// допустимыми командами являются FEED_GRID_PAGE и UPDATE_ROW
if ($action == 'FEED_GRID_PAGE')
{
    // извлечь номер страницы
    $page = $_GET['page'];
    // прочитать список продуктов на странице
    $grid->readPage($page);
}
else if ($action == 'UPDATE_ROW')
{
    // извлечь параметры
    $id = $_GET['id'];
    $on_promotion = $_GET['on_promotion'];
    $price = $_GET['price'];
    $name = $_GET['name'];
    // обновить запись
    $grid->updateRecord($id, $on_promotion, $price, $name);
}
else
    echo 'Server error: от клиента поступила неверная команда.';
// очистить выходной буфер
if(ob_get_length()) ob_clean();
// исключить возможность кэширования страницы броузерами
header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); // время в прошлом
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/xml');
// выходные данные будут иметь формат XML
header('Content-type: text/xml');
echo '<?xml version="1.0" encoding="ISO-8859-1"?>';
echo '<data>';
echo '<action>' . $action . '</action>';
echo $grid->getParamsXML();
echo $grid->getGridXML();
echo '</data>';
?>
```

4. **Создайте файл с именем** `grid.class.php` **и добавьте в него следующий код:**

```
<?php
// загрузить конфигурационный файл
require_once('config.php');
// запустить сессию
```

```
session_start();

// класс, обеспечивающий реализацию действий со списком продуктов
class Grid
{
    // счетчик страниц таблицы
    public $mTotalPages;
    // счетчик элементов
    public $mItemsCount;
    // индекс возвращаемой таблицы
    public $mReturnedPage;
    // дескриптор соединения с базой данных
    private $mMysqli;
    // дескриптор соединения с базой данных
    private $grid;
    // конструктор класса
    function __construct()
    {
        // открыть соединение с базой данных MySQL
        $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                    DB_DATABASE);
        // с помощью countAllRecords получить количество записей в таблице
        $this->mItemsCount = $this->countAllRecords();
    }

    // деструктор класса, закрывает соединение с базой данных
    function __destruct()
    {
        $this->mMysqli->close();
    }

    // прочитать список продуктов на странице и сохранить его в $this->grid
    public function readPage($page)
    {
        // создать SQL запрос, который вернет список продуктов на странице
        $queryString = $this->createSubpageQuery('SELECT * FROM product',
                                                  $page);
        // выполнить запрос
        if ($result = $this->mMysqli->query($queryString))
        {
            // извлечь данные в виде ассоциативного массива
            while ($row = $result->fetch_assoc())
            {
                // создать документ XML, содержащий список продуктов
                $this->grid .= '<row>';
                foreach($row as $name=>$val)
                    $this->grid .= '<' . $name . '>' .
                                   htmlentities($val) .
                                   '</' . $name . '>';
                $this->grid .= '</row>';
            }
            // закрыть полученный из базы набор данных
```

```
            $result->close();
        }
    }

    // обновить запись
    public function updateRecord($id, $on_promotion, $price, $name)
    {
        // экранировать специальные символы во входных данных, чтобы их
        // можно было безопасно использовать в составе SQL запросов
        $id = $this->mMysqli->real_escape_string($id);
       $on_promotion = $this->mMysqli->real_escape_string($on_promotion);
        $price = $this->mMysqli->real_escape_string($price);
        $name = $this->mMysqli->real_escape_string($name);
        // создать SQL запрос, который изменит запись
        $queryString = 'UPDATE product SET name="' . $name . '", ' .
                       'price=' . $price . ',' .
                       'on_promotion=' . $on_promotion .
                       ' WHERE product_id=' . $id;
        // выполнить команду SQL
        $this->mMysqli->query($queryString);
    }

    // возвращает сведения о текущем запросе (кол-во страниц и т. д.)
    public function getParamsXML()
    {
        // вычислить номер предыдущей страницы
        $previous_page =
             ($this->mReturnedPage == 1) ? '' : $this->mReturnedPage-1;
        // вычислить номер следующей страницы
        $next_page = ($this->mTotalPages == $this->mReturnedPage) ?
                     '' : $this->mReturnedPage + 1;
        // вернуть параметры
        return '<params>' .
               '<returned_page>' . $this->mReturnedPage .
                                   '</returned_page>'.
               '<total_pages>' . $this->mTotalPages . '</total_pages>'.
               '<items_count>' . $this->mItemsCount . '</items_count>'.
               '<previous_page>' . $previous_page . '</previous_page>'.
               '<next_page>' . $next_page . '</next_page>' .
               '</params>';
    }

    // возвращает текущую страницу таблицы в формате XML
    public function getGridXML()
    {
        return '<grid>' . $this->grid . '</grid>';
    }

    // возвращает общее число записей в таблице
    private function countAllRecords()
    {
        /* если количество записей в этой сессии еще не определялось,
           прочитать значение из базы данных */
```

```
            if (!isset($_SESSION['record_count']))
            {
                // запрос, который вернет количество записей
                $count_query = 'SELECT COUNT(*) FROM product';
                // выполнить запрос и забрать результат
                if ($result = $this->mMysqli->query($count_query))
                {
                    // извлечь первую строку
                    $row = $result->fetch_row();
                    /* извлечь значение первого поля в первой строке
                      (в нем находится кол-во записей) и сохранить
                      его в переменной сессии */
                    $_SESSION['record_count'] = $row[0];
                    // закрыть набор данных
                    $result->close();
                }
            }
            // прочитать значение из переменной сессии и вернуть его
            return $_SESSION['record_count'];
        }

        // получает запрос SELECT, который вернет весь список продуктов
        // и модифицирует его так, чтобы запрос возвращал список продуктов
        // только для заданной страницы
        private function createSubpageQuery($queryString, $pageNo)
        {
            // если количество продуктов невелико, то мы не производим
            // разбивку списка на страницы
            if ($this->mItemsCount <= ROWS_PER_VIEW)
            {
                $pageNo = 1;
                $this->mTotalPages = 1;
            }
           // иначе – вычисляем количество страниц и строим новый запрос SELECT
            else
            {
               $this->mTotalPages = ceil($this->mItemsCount / ROWS_PER_VIEW);
                $start_page = ($pageNo - 1) * ROWS_PER_VIEW;
               $queryString .= ' LIMIT ' . $start_page . ',' . ROWS_PER_VIEW;
            }
            // сохранить номер возвращаемой страницы
            $this->mReturnedPage = $pageNo;
            // вернуть новую строку запроса
            return $queryString;
        }
        // конец определения класса Grid
    }
    ?>
```

5. **Добавьте файл с параметрами соединения** `config.php`**:**

```
<?php
// сведения, необходимые для соединения с базой данных
```

```
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

6. **Создайте сценарий обработки ошибок** `error_handler.php` **и добавьте в него следующий код:**

```
<?php
// установить функцию error_handler как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

7. **Теперь переместимся на сторону клиента. Начнем с файла** `index.html`**:**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head>
        <title>Таблицы в AJAX</title>
        <script type="text/javascript" src="grid.js"></script>
        <link href="grid.css" type="text/css" rel="stylesheet"/>
    </head>
    <body onload="init();">
        <div id="gridDiv" />
    </body>
</html>
```

8. **Далее создайте файл XSLT с именем** `grid.xsl`**, он будет использоваться сценарием JavaScript:**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <h2>Таблицы в AJAX</h2>
    <xsl:call-template name="menu"/>
    <form id="grid_form_id">
```

```
    <table class="list">
      <tr>
        <th class="th1">№</th>
        <th class="th2">Название</th>
        <th class="th3">Цена</th>
        <th class="th4">Реклама</th>
        <th class="th5"></th>
      </tr>
      <xsl:for-each select="data/grid/row">
        <xsl:element name="tr">
          <xsl:attribute name="id">
            <xsl:value-of select="product_id" />
          </xsl:attribute>
          <td><xsl:value-of select="product_id" /></td>
          <td><xsl:value-of select="name" /> </td>
          <td><xsl:value-of select="price" /></td>
          <td>
            <xsl:choose>
              <xsl:when test="on_promotion &gt; 0">
                <input type="checkbox" name="on_promotion"
                       disabled="disabled" checked="checked"/>
              </xsl:when>
              <xsl:otherwise>
                <input type="checkbox" name="on_promotion"
                       disabled="disabled"/>
              </xsl:otherwise>
            </xsl:choose>
          </td>
          <td>
            <xsl:element name="a">
              <xsl:attribute name = "href">#</xsl:attribute>
              <xsl:attribute name = "onclick">
                  editId(<xsl:value-of select="product_id" />, true)
              </xsl:attribute>
              Редактировать
            </xsl:element>
          </td>
        </xsl:element>
      </xsl:for-each>
    </table>
  </form>
  <xsl:call-template name="menu" />
</xsl:template>
<xsl:template name="menu">
  <xsl:for-each select="data/params">
    <table>
      <tr>
        <td class="left">
          <xsl:value-of select="items_count" /> Items
        </td>
        <td class="right">
```

```
            <xsl:choose>
              <xsl:when test="previous_page>0">
                <xsl:element name="a" >
                  <xsl:attribute name="href" >#</xsl:attribute>
                  <xsl:attribute name="onclick">
                    loadGridPage(<xsl:value-of select="previous_page"/>)
                  </xsl:attribute>
                  Предыдущая страница
                </xsl:element>
              </xsl:when>
            </xsl:choose>
          </td>
          <td class="left">
            <xsl:choose>
              <xsl:when test="next_page>0">
                <xsl:element name="a">
                  <xsl:attribute name = "href" >#</xsl:attribute>
                  <xsl:attribute name = "onclick">
                    loadGridPage(<xsl:value-of select="next_page"/>)
                  </xsl:attribute>
                  Следующая страница
                </xsl:element>
              </xsl:when>
            </xsl:choose>
          </td>
          <td class="right">
            страница <xsl:value-of select="returned_page" />
            из <xsl:value-of select="total_pages" />
          </td>
        </tr>
      </table>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

9. **Создайте файл** `grid.js`**:**

```
// ссылка на объект XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// имя файла XSLT
var xsltFileUrl = "grid.xsl";
// сценарий, возвращающий данные в формате XML
var feedGridUrl = "grid.php";
// идентификатор элемента div, где выводится таблица
var gridDivId = "gridDiv";
// строка состояния таблицы – элемент div
var statusDivId = "statusDiv";
// временная строка данных
var tempRow;
// ID продукта, который редактируется
var editableId = null;
// документ XSLT
```

```
var stylesheetDoc;

// все начинается здесь
function init()
{
    // проверить, поддерживает ли броузер возможность работы с XSLT
    if(window.XMLHttpRequest && window.XSLTProcessor && window.DOMParser)
    {
        // загрузить таблицу
        loadStylesheet();
        loadGridPage(1);
        return;
    }
    // проверить, корректно ли поддерживает XSLT
    // версия Internet Explorer, которая имеется у пользователя
    if (window.ActiveXObject && createMsxml2DOMDocumentObject())
    {
        // загрузить таблицу
        loadStylesheet();
        loadGridPage(1);
        // выйти из функции
        return;
    }
    // если проверка возможностей броузера потерпела неудачу,
    // известить пользователя
    alert("Ваш броузер не обладает необходимой функциональностью.");
}

function createMsxml2DOMDocumentObject()
{
    // ссылка на объект MSXML
    var msxml2DOM;
    // версии MSXML, которые могут использоваться
    // для работы с нашей таблицей
    var msxml2DOMDocumentVersions = new Array("Msxml2.DOMDocument.6.0",
                                              "Msxml2.DOMDocument.5.0",
                                              "Msxml2.DOMDocument.4.0");
    // попытаться отыскать подходящий объект MSXML
    for (var i=0; i<msxml2DOMDocumentVersions.length && !msxml2DOM; i++)
    {
        try
        {
            // попытаться создать объект
            msxml2DOM = new ActiveXObject(msxml2DOMDocumentVersions[i]);
        }
        catch (e) {}
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!msxml2DOM)
        alert("Пожалуйста, обновите версию MSXML с сайта \n" +
              "http://msdn.microsoft.com/XML/XMLDownloads/default.aspx");
    else
```

```
        return msxml2DOM;
}

// создает экземпляр объекта XMLHttpRequest
function createXmlHttpRequestObject()
{
    // ссылка на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {}
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// загружает таблицу стилей с сервера с помощью синхронного запроса
function loadStylesheet()
{
    // загрузить файл с сервера
    xmlHttp.open("GET", xsltFileUrl, false);
    xmlHttp.send(null);
    // попытаться загрузить документ XSLT
    if (this.DOMParser) // броузер со встроенной поддержкой XSLT
```

```
    {
        var dp = new DOMParser();
        stylesheetDoc = dp.parseFromString(xmlHttp.responseText,
                                                    "text/xml");
    }
    else if (window.ActiveXObject) // Internet Explorer?
    {
        stylesheetDoc = createMsxml2DOMDocumentObject();
        stylesheetDoc.async = false;
        stylesheetDoc.load(xmlHttp.responseXML);
    }
}

// выполняет асинхронный запрос на получение новой страницы таблицы
function loadGridPage(pageNo)
{
    // запретить режим редактирования на время загрузки
    editableId = false;
    // продолжать только если объект XMLHttpRequest не занят
    if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
    {
        var query = feedGridUrl + "?action=FEED_GRID_PAGE&page=" + pageNo;
        xmlHttp.open("GET", query, true);
        xmlHttp.onreadystatechange = handleGridPageLoad;
        xmlHttp.send(null);
    }
}

// обрабатывает ответ сервера, содержащий новую страницу таблицы
// со списком продуктов
function handleGridPageLoad()
{
    // когда readyState = 4, мы можем прочитать ответ сервера
    if (xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if (xmlHttp.status == 200)
        {
            // прочитать ответ
            response = xmlHttp.responseText;
            // ошибка сервера?
            if (response.indexOf("ERRNO") >= 0
                    || response.indexOf("error") >= 0
                    || response.length == 0)
            {
                // вывести сообщение об ошибке
                alert(response.length == 0 ? "Server serror." : response);
                // выйти из функции
                return;
            }
            // сервер посылает ответы в формате XML
            xmlResponse = xmlHttp.responseXML;
```

```
                // броузер имеет встроенную поддержку XSLT?
                if (window.XMLHttpRequest && window.XSLTProcessor &&
                    window.DOMParser)
                {
                    // загрузить документ XSLT
                    var xsltProcessor = new XSLTProcessor();
                    xsltProcessor.importStylesheet(stylesheetDoc);
                  // сгенерировать код HTML с новой страницей таблицы продуктов
                    page = xsltProcessor.transformToFragment(xmlResponse,
                                                             document);
                    // отобразить страницу
                    var gridDiv = document.getElementById(gridDivId);
                    gridDiv.innerHTML = "";
                    gridDiv.appendChild(page);
                }
                // эта часть кода предназначена для Internet Explorer
                else if (window.ActiveXObject)
                {
                    // загрузить документ XSLT
                    var theDocument = createMsxml2DOMDocumentObject();
                    theDocument.async = false;
                    theDocument.load(xmlResponse);
                    // отобразить страницу
                    var gridDiv = document.getElementById(gridDivId);
                    gridDiv.innerHTML =
                                    theDocument.transformNode(stylesheetDoc);
                }
            }
            else
            {
                alert("Ошибка чтения ответа от сервера.")
            }
        }
    }

    // переводит строку по заданному id продукта в режим редактирования,
    // если в переменной editMode содержится значение true, и отменяет режим
    // редактирования, если в переменной editMode содержится значение false
    function editId(id, editMode)
    {
        // получить ссылку на элемент <tr> таблицы, которая содержит список
        var productRow = document.getElementById(id).cells;
        // режим редактирования разрешен?
        if(editMode)
        {
            // одновременно в режиме редактирования может находиться
            // только одна строка
            if(editableId) editId(editableId, false);
            // сохранить текущие значения, на случай если пользователь решит
            // отменить свои изменения
            save(id);
```

```
            // создать поля редактирования
            productRow[1].innerHTML =
                    '<input class="editName" type="text" name="name" ' +
                    'value="' + productRow[1].innerHTML+'">';
            productRow[2].innerHTML =
                    '<input class="editPrice" type="text" name="price" ' +
                    'value="' + productRow[2].innerHTML+'">';
            productRow[3].getElementsByTagName("input")[0].disabled = false;
            productRow[4].innerHTML = '<a href="#" ' +
                    'onclick="updateRow(document.forms.grid_form_id,' + id +
                    ')">Изменить</a><br/><a href="#" onclick="editId(' + id +
                    ',false)">Отмена</a>';
            // запомнить id редактируемого продукта
            editableId = id;
        }
        // если режим редактирования запрещен...
        else
        {
            productRow[1].innerHTML = document.forms.grid_form_id.name.value;
            productRow[2].innerHTML = document.forms.grid_form_id.price.value;
            productRow[3].getElementsByTagName("input")[0].disabled = true;
            productRow[4].innerHTML = '<a href="#" onclick="editId(' + id +
                                      ',true)">Редактировать</a>';
            // нет редактируемых строк
            editableId = null;
        }
    }

    // сохраняет первоначальную информацию о продукте перед редактированием
    function save(id)
    {
        // получить ссылку на строку в таблице
        var tr = document.getElementById(id).cells;
        // сохранить данные
        tempRow = new Array(tr.length);
        for(var i=0; i<tr.length; i++)
        tempRow[i] = tr[i].innerHTML;
    }

    // отменяет внесенные изменения и восстанавливает первоначальные значения
    function undo(id)
    {
        // получить ссылку на строку в таблице
        var tr = document.getElementById(id).cells;
        // скопировать прежние значения
        for(var i=0; i<tempRow.length; i++)
            tr[i].innerHTML = tempRow[i];
        // нет редактируемых строк
        editableId = null;
    }

    // отправляет на сервер изменения, сделанные пользователем,
```

```
// если это возможно
function updateRow(grid, productId)
{
    // продолжать только если объект XMLHttpRequest не занят
    if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
    {
        var query = feedGridUrl + "?action=UPDATE_ROW&id=" + productId +
                    "&" + createUpdateUrl(grid);
        xmlHttp.open("GET", query, true);
        xmlHttp.onreadystatechange = handleUpdatingRow;
        xmlHttp.send(null);
    }
}

// обрабатывает принятый от сервера ответ,
// соответствующий запросу на изменение сведений о продукте
function handleUpdatingRow()
{
    // если readyState = 4, мы можем прочитать ответ сервера
    if(xmlHttp.readyState == 4)
    {
        // продолжать, только если статус HTTP равен «OK»
        if(xmlHttp.status == 200)
        {
            // прочитать ответ
            response = xmlHttp.responseText;
            // ошибка сервера?
            if (response.indexOf("ERRNO") >= 0
                    || response.indexOf("error") >= 0
                    || response.length == 0)
               alert(response.length == 0 ? "Server serror." : response);
            // если ошибок обнаружено не было, выйти
            // из режима редактирования
            else
                editId(editableId, false);
        }
        else
        {
            // отменить внесенные изменения в случае ошибки
            undo(editableId);
            alert("Ошибка сервера.");
        }
    }
}

// создает строку параметров запроса на изменение записи
function createUpdateUrl(grid)
{
    // инициализировать строку запроса
    var str = "";
    // собрать строку запроса со значениями из элементов таблицы
    for(var i=0; i<grid.elements.length; i++)
```

```
            switch(grid.elements[i].type)
            {
                case "text":
                case "textarea":
                    str += grid.elements[i].name + "=" +
                           escape(grid.elements[i].value) + "&";
                    break;
                case "checkbox":
                    if (!grid.elements[i].disabled)
                        str += grid.elements[i].name + "=" +
                               (grid.elements[i].checked ? 1 : 0) + "&";
                    break;
            }
        // вернуть строку запроса
        return str;
    }
```

10. И наконец, создайте файл `grid.css`**:**

```
body
{
    font-family: Verdana, Arial;
    font-size: 10pt
}

table
{
    width: 500px;
}

td.right
{
    color: darkblue;
    text-align: right;
    width: 125px
}

td.left
{
    color: darkblue;
    text-align: left;
    width: 125px
}

table.list
{
    border: black 1px solid;
}

th
{
    text-align: left;
    background-color: navy;
    color: white
```

```
}

th.th1
{
    width: 30px
}

th.th2
{
    width: 300px
}

input.editName
{
    border: black 1px solid;
    width: 300px
}

input.editPrice
{
    border: black 1px solid;
    width: 50px
}
```

11. Откройте страницу *http://localhost/ajax/grid* в броузере, проверьте ее и убедитесь, что она работает так, как мы и ожидали (для справки см. рис. 8.1 и 8.2).

Что происходит внутри?

Начнем рассмотрение программного кода со сценариев, работающих на стороне сервера. Основу серверной части составляет база данных. В нашем случае это таблица `product`, содержащая следующие поля:

- `product_id` – это первичный ключ таблицы. Содержит числовой идентификатор продукта.
- `name` – название продукта.
- `price` – цена продукта.
- `on_promotion` – битовое поле (должно принимать только два значения 0 или 1, хотя MySQL в зависимости от версии может разрешить большее количество возможных значений), которое содержит признак участия данного продукта в рекламной кампании. Мы добавили это поле только ради того, чтобы продемонстрировать, как использовать элементы-флажки (checkboxes) для отображения битовых значений.

Как обычно на стороне сервера работает сценарий PHP, который в данном случае назван `grid.php` и представляет собой главную точку входа для всех асинхронных запросов клиента.

Сценарий `grid.php` ожидает получить от клиента строку с параметром запроса (с именем `action`), который сообщит ему, что необходимо сделать. Возможные значения этого параметра:

- `FEED_GRID_PAGE`: Это значение используется для получения страницы со списком продуктов. Вместе с этим параметром серверу должен быть передан параметр `page`, который определяет номер запрошенной страницы.
- `UPDATE_ROW`: Это значение нужно для обновления сведений о продукте, измененных пользователем. Для выполнения этого действия серверу должны быть переданы новые значения полей записи в виде четырех параметров: `id`, `name`, `price` и `on_promotion`.

Чтобы увидеть, в каком виде данные поступают от сервера, выполните простой вызов *http://localhost/ajax/grid/grid.php?action=FEED_GRID_PAGE&page=1*. При использовании в базе данных значений по умолчанию, результат должен выглядеть примерно, как на рис. 8.3.

На стороне клиента эти данные будут проанализированы и преобразованы в таблицу HTML с помощью XSL-преобразования. Этот программный код был проверен в броузерах Mozilla и Internet Explorer, которые к моменту написания этой книги обладали всей необходимой функциональностью. Броузер Opera, как ожидается, будет поддерживать преобразования XSL, начиная с версии 9.

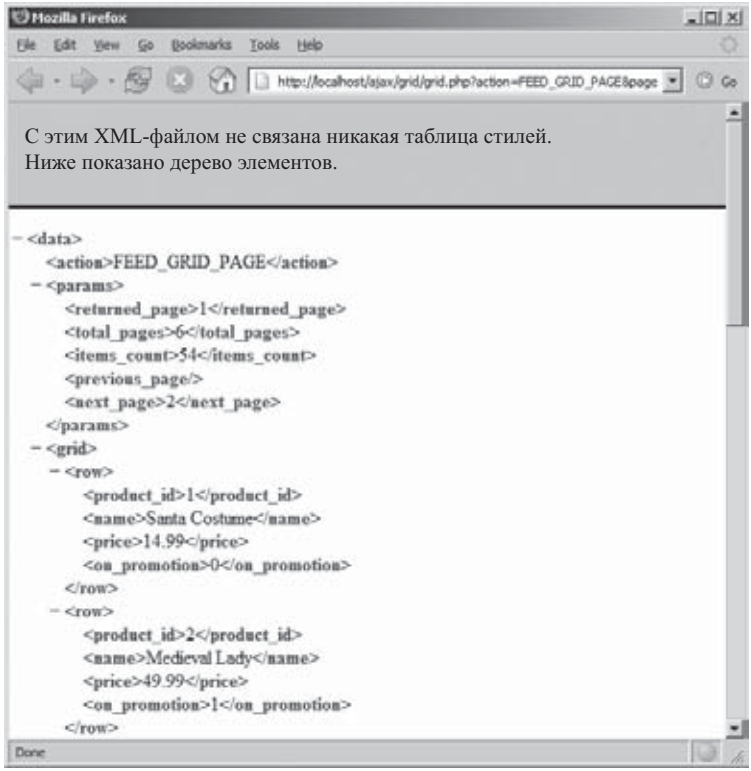

Рис. 8.3. *Сервер вернул первую страницу со списком продуктов*

Код XSL-преобразования находится в файле grid.xsl. Краткое введение в XSL вы найдете в приложении C на сайте книги *http://ajaxphp.packtpub.com*. За более подробной информацией обращайтесь к Интернету и многочисленным книгам. Тема XSL очень обширна, так что готовьтесь к длительному процессу изучения, если вы намерены овладеть ею.

Функция init() – это первая функция сценария grid.js, работающего на стороне клиента. Она проверяет, обладает ли броузер пользователя функциональностью, необходимой для выполнения XSL-преобразований:

```
// все начинается здесь
function init()
{
    // проверить, поддерживает ли броузер возможность работы с XSLT
    if(window.XMLHttpRequest && window.XSLTProcessor && window.DOMParser)
    {
        // загрузить таблицу
        loadStylesheet();
        loadGridPage(1);
        return;
    }
    // проверить, корректно ли поддерживает XSLT версия Internet Explorer,
    // которая имеется у пользователя
    if (window.ActiveXObject && createMsxml2DOMDocumentObject())
    {
        // загрузить таблицу
        loadStylesheet();
        loadGridPage(1);
        // выйти из функции
        return;
    }
    // если проверка возможностей броузера потерпела неудачу,
    // известить пользователя
    alert("Ваш броузер не обладает необходимой функциональностью.");
}
```

Эта функция дает возможность продолжить работу, только если обнаружен либо Internet Explorer (в этом случае у пользователя должна быть установлена достаточно свежая версия библиотеки MSXML), либо броузер, имеющий встроенную поддержку классов XMLHttpRequest, XSLTProcessor и DOMParser.

Следующая функция, с которой также следует разобраться, – это loadStylesheet(). Она вызывается всего один раз, во время загрузки страницы. Она запрашивает с сервера файл grid.xsl и загружает его. Файл grid.xsl загружается с помощью синхронного вызова и затем сохраняется локально способом, определяемым броузером клиента, который зависит от того, обладает ли броузер встроенной поддержкой необходимой функциональности или это Internet Explorer, который использует объект ActiveXObject:

```
// загружает таблицу стилей с сервера с помощью синхронного запроса
function loadStylesheet()
{
    // загрузить файл с сервера
    xmlHttp.open("GET", xsltFileUrl, false);
    xmlHttp.send(null);
    // попытаться загрузить документ XSLT
    if (this.DOMParser) // броузер с встроенной поддержкой XSLT
    {
        var dp = new DOMParser();
        stylesheetDoc = dp.parseFromString(xmlHttp.responseText, "text/xml");
    }
    else if (window.ActiveXObject) // Internet Explorer?
    {
        stylesheetDoc = createMsxml2DOMDocumentObject();
        stylesheetDoc.async = false;
        stylesheetDoc.load(xmlHttp.responseXML);
    }
}
```

Функция `loadGridPage` первый раз вызывается во время загрузки страницы, а затем всякий раз, когда пользователь щелкает по ссылкам Previous Page (Предыдущая страница) и Next Page (Следующая страница), чтобы загрузить новую порцию данных. Эта функция отправляет серверу асинхронные запросы, указывая номер требуемой страницы:

```
// выполняет асинхронный запрос на получение новой страницы таблицы
function loadGridPage(pageNo)
{
    // запретить режим редактирования на время загрузки
    editableId = false;
    // продолжать только если объект XMLHttpRequest не занят
    if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
    {
        var query = feedGridUrl + "?action=FEED_GRID_PAGE&page=" + pageNo;
        xmlHttp.open("GET", query, true);
        xmlHttp.onreadystatechange = handleGridPageLoad;
        xmlHttp.send(null);
    }
}
```

Функция обратного вызова `handleGridPageLoad` обрабатывает ответ сервера. После типичной последовательности действий по выявлению ошибок следует программный код, который эффективно выполняет преобразование документа XML, принятого от сервера, в код HTML, который должен отображаться клиентом. Код, выполняющий преобразование, опять же зависит от типа броузера. Последовательность действий для Internet Explorer и для броузеров, обладающих встроенной поддержкой XSL, различна:

```
// сервер посылает ответы в формате XML
xmlResponse = xmlHttp.responseXML;
```

```
// броузер имеет встроенную поддержку XSLT?
if (window.XMLHttpRequest && window.XSLTProcessor && window.DOMParser)
{
    // загрузить документ XSLT
    var xsltProcessor = new XSLTProcessor();
    xsltProcessor.importStylesheet(stylesheetDoc);
    // сгенерировать код HTML с новой страницей таблицы продуктов
    page = xsltProcessor.transformToFragment(xmlResponse, document);
    // отобразить страницу
    var gridDiv = document.getElementById(gridDivId);
    gridDiv.innerHTML = "";
    gridDiv.appendChild(page);
}
// эта часть кода предназначена для Internet Explorer
else if (window.ActiveXObject)
{
    // загрузить документ XSLT
    var theDocument = createMsxml2DOMDocumentObject();
    theDocument.async = false;
    theDocument.load(xmlResponse);
    // отобразить страницу
    var gridDiv = document.getElementById(gridDivId);
    gridDiv.innerHTML = theDocument.transformNode(stylesheetDoc);
}
```

Следующая функция `editId`. Она вызывается, когда пользователь щелкает по ссылке Edit (Редактировать) или Cancel (Отмена), чтобы войти или выйти из режима редактирования. При входе в режим редактирования поля с названием продукта, его ценой и признаком участия в рекламной кампании преобразуются в поля ввода. При выходе из режима редактирования те же самые поля возвращаются в состояние, не доступное для редактирования.

Вспомогательные функции `save()` и `undo()` вызываются при редактировании строк. Функция `save` сохраняет первоначальные значения полей записи, которые загружаются обратно в таблицу функцией `undo`, если пользователь изменит их и затем щелкнет по ссылке `Отмена`, передумав вносить изменения.

Запись изменений в базу данных на сервере производится функцией `updateRow`, которая вызывается щелчком по ссылке `Изменить`. Эта функция отправляет серверу асинхронный запрос, посылая в нем новые значения, которые собираются в одну строку запроса с помощью вспомогательной функции `createUpdateUrl`:

```
// отправляет на сервер изменения, сделанные пользователем, если это возможно
function updateRow(grid, productId)
{
    // продолжать только если объект XMLHttpRequest не занят
    if (xmlHttp && (xmlHttp.readyState == 4 || xmlHttp.readyState == 0))
    {
```

```
            var query = feedGridUrl + "?action=UPDATE_ROW&id=" + productId +
                        "&" + createUpdateUrl(grid);
            xmlHttp.open("GET", query, true);
            xmlHttp.onreadystatechange = handleUpdatingRow;
            xmlHttp.send(null);
        }
    }
```

Функция обратного вызова `handleUpdatingRow` отвечает за выход из режима редактирования, если изменения прошли успешно, и за вывод сообщения, если на стороне сервера произошла ошибка:

```
// продолжать, только если статус HTTP равен «OK»
if(xmlHttp.status == 200)
{
    // прочитать ответ
    response = xmlHttp.responseText;
    // ошибка сервера?
    if (response.indexOf("ERRNO") >= 0
            || response.indexOf("error") >= 0
            || response.length == 0)
        alert(response.length == 0 ? "Server serror." : response);
    // если ошибок обнаружено не было, выйти из режима редактирования
    else
        editId(editableId, false);
}
```

Способ отображения ошибок не изменился по сравнению с предыдущими упражнениями. Если сервер возвратил определенное сообщение об ошибке, оно отображается перед пользователем. Если конфигурация PHP не предполагает вывод сообщений об ошибках, мы получим пустой ответ от сервера, и в этом случае мы просто выводим обобщенное сообщение.

Подведение итогов

В этой главе для создания таблиц данных применялись уже знакомые вам приемы технологии AJAX. Вы встретились с XSL, позволяющим выстроить очень гибкую архитектуру, в которой серверная часть приложения не должна беспокоиться о способе отображения данных.

Применение XSL для форматирования данных, прежде чем представить их пользователю, – один из профессиональных приемов решения подобного рода задач. Если вы намерены всерьез заняться разработкой веб-приложений, рекомендуем как следует изучить XSL. Предупреждаем: это будет долгий и трудный путь, но в конце концов ваши усилия окупятся сторицей.

9

Чтение лент новостей в AJAX

За несколько последних лет жизнь во Всемирной сети стала намного более интенсивной. Мы наблюдаем взрывной рост количества новых источников информации, таких как новостные сайты (например, *http://www.newsvine.com* и *http://www.digg.com*), появляющихся ежедневно, и персональные сетевые дневники – блоги (создается ощущение, что сейчас каждый человек имеет свой блог).

Совершенно естественно, что в ответ на такой наплыв информации появилось множество систем, которые позволяют собирать, классифицировать и фильтровать эту информацию. На практике подобные системы реализуются на основе **централизованного распространения информации в Интернете (синдицирования)**, когда отдельные части сайта (раздел новостей, блоги, статьи и т. д.) делаются доступными для других сайтов или приложений.

Чтобы сделать информацию пригодной для использования другими сторонами, ее надо распространять в обобщенном формате, который может быть преобразован в любые другие форматы, отличные от оригинального. Самые популярные форматы представления информации: **RSS 2.0** (Really Simple Syndication, или очень простой способ распространения информации) и **Atom**.

Узнать подробнее об истории развития форматов RSS и Atom можно в веб-энциклопедии: *http://en.wikipedia.org/wiki/RSS_(protocol)*.

В этой главе мы рассмотрим формат RSS, затем заглянем на Google Reader (RSS-агрегатор Google), а потом создадим собственную веб-страницу RSS-агрегатора на основе технологии AJAX и PHP.

Работаем с RSS

RSS – это широко распространенный формат, основанный на XML и предназначенный для обмена информацией в Интернете между приложениями. Самое главное преимущество XML заключается в том,

что это самый обычный текстовый формат, который легко может быть прочитан каким угодно приложением. RSS-ленты могут просматриваться как обычные текстовые файлы, но в этом нет большого смысла, т. к. для их чтения необходимо специализированное ПО, которое генерирует веб-страницы на основе содержащихся в них данных.

RSS не единственный стандарт распространения рассылок в виде XML, но для следующего упражнения мы выбрали именно его, потому что он получил очень широкое распространение. Чтобы лучше понять RSS, мы должны узнать, что подразумевается под этим названием и какова структура документа RSS.

Структура документа RSS

Первая версия RSS появилась в 1999 г. Эта версия известна как 0.9. С тех пор этот формат дорос до текущей версии 2.0.1, которая была *заморожена* сообществом разработчиков, поскольку новая версия скорее всего выйдет уже под другим именем.

Типичная лента RSS выглядит следующим образом:

```
<rss version="2.0">
  <channel>
    <title>CNN.com</title>
    <link>http://www.example.org</link>
    <description>A short description of this feed</description>
    <language>en</language>
    <pubDate>Mon, 17 Oct 2005 07:56:23 EDT</pubDate>
    <item>
      <title>Catchy Title</title>
      <link>http://www.example.org/2005/11/catchy-title.html</link>
      <description>
        The description can hold any content you wish, including XHTML.
      </description>
      <pubDate>Mon, 17 Oct 2005 07:55:28 EDT</pubDate>
    </item>
    <item>
      <title>Another Catchy Title</title>
      <link>http://www.example.org/2005/11/another-catchy-title.html</link>
      <description>
        The description can hold any content you wish, including XHTML.
      </description>
      <pubDate>Mon, 17 Oct 2005 07:55:28 EDT</pubDate>
    </item>
  </chanel>
</rss>
```

Лента может содержать любое количество элементов `<item>`. Каждый из них содержит автономную новость или запись в блоге, вне зависимости от ее содержимого.

Вся информация хранится в виде обычного текста, но, как мы уже отмечали выше, для ее чтения требуется специализированное ПО, которое выполнит синтаксический разбор файла XML и вернет необходимую нам информацию. Программы, анализирующие формат RSS, называются агрегаторами, потому что основное их предназначение состоит в том, чтобы извлекать и собирать воедино (агрегировать) информацию из нескольких RSS-источников.

Примером такого приложения может служить Google Reader – интернет-услуга компании Google, запущенная в обращение осенью 2005 г. Одним из ветеранов по предоставлению веб-услуги чтения лент RSS является *http://www.bloglines.com*.

Google Reader

Google Reader (*http://reader.google.com*) реализует простой и интуитивно понятный интерфейс на основе технологии AJAX, который помогает пользователям следить за своими RSS-подписками и читать их. Эта услуга запущена не так давно (к моменту написания этой книги она еще находилась в состоянии бета-версии), но уже стала весьма популярной у пользователей. На рис. 9.1 показано приложение Google Reader, читающее новости с ленты RSS издательства Packt Publishing.

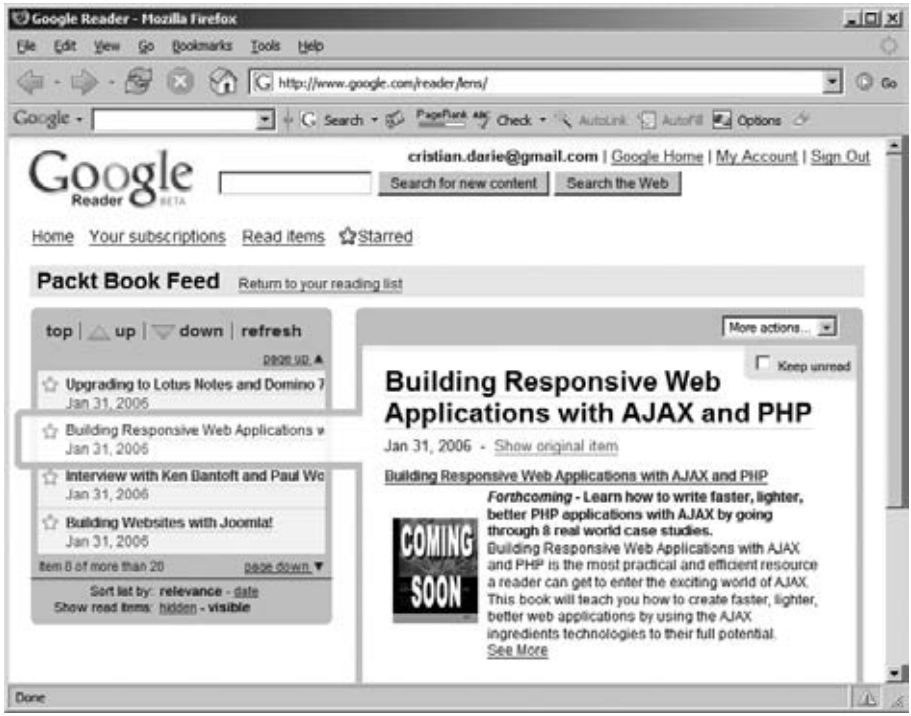

***Рис. 9.1.** Управление подписками (лентами) RSS в Google Reader*

Реализация чтения лент RSS с помощью технологии AJAX

Для нормальной работы этого упражнения необходимо, чтобы во время установки PHP была включена поддержка XSL. Инструкции по установке, содержащиеся в приложении А, включают в себя шаги по обеспечению поддержки XSL.

В следующем упражнении мы создадим приложение для чтения лент RSS на основе технологии AJAX. Ниже приводятся его основные характеристики:

1. Приложение будет достаточно простым. Список лент жестко «зашит» в код серверного сценария PHP.
2. Для преобразования RSS в некий другой формат, который мы сможем отображать перед посетителем, будет применяться XSLT. В этой главе XSL-преобразование будет выполняться на стороне сервера средствами PHP.
3. Читать ответ сервера новостей мы будем средствами библиотеки SimpleXML, которая впервые появилась в PHP 5. Официальную документацию к ней вы найдете по адресу *http://php.net/simplexml*. SimpleXML – это замечательная библиотека, которая позволяет читать содержимое документов XML намного проще, чем это делается в случае применения интерфейса DOM.
4. Внешний вид приложения приводится на рис. 9.2.

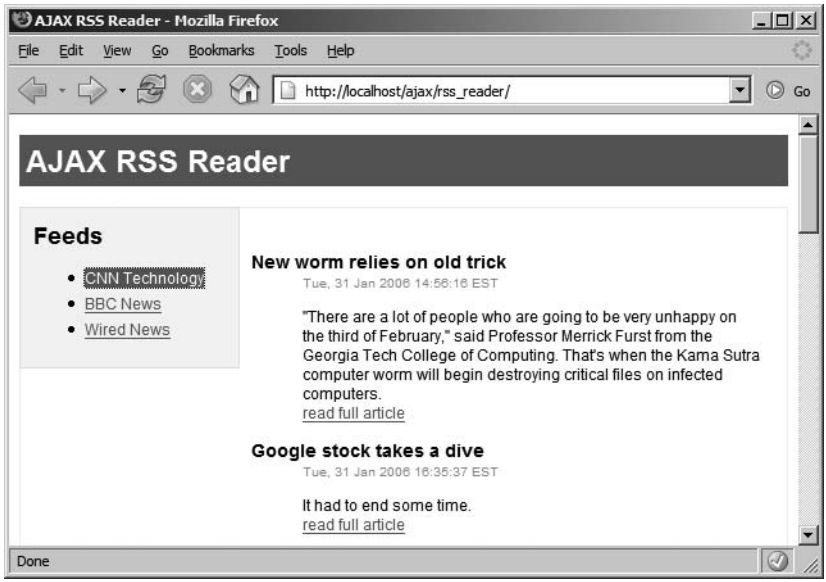

Рис. 9.2. *Начальная страница нашего приложения чтения лент RSS*

Ленты новостей загружаются динамически и отображаются в виде ссылок в левом столбце. Щелчок по ленте будет приводить к отправке запроса HTTP, в ответ на который серверный сценарий будет получать желаемую ленту.

После этого сервер выполнит XSL-преобразование ленты новостей и вернет ее в виде строки XML. Результаты будут в удобочитаемом виде отображаться пользователю.

Время действовать – программа чтения лент RSS

1. В каталоге `ajax` создайте новый каталог с именем `rss_reader`.
2. Сначала создадим серверную часть приложения. Создайте файл с именем `rss_reader.php` и добавьте в него следующий код:

```
<?php
// загрузить вспомогательные сценарии
require_once ('error_handler.php');
require_once ('rss_reader.class.php');
// создать экземпляр класса чтения RSS
$reader = new CRssReader(urldecode($_POST['feed']));
// очистить выходной буфер
if(ob_get_length()) ob_clean();
// предотвратить возможность кэширования страницы броузерами
header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); // время в прошлом
header('Last-Modified: ' . gmdate( 'D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/xml');
// вернуть новости клиенту
echo $reader->getFormattedXML();
?>
```

3. Создайте новый файл с именем `rss_reader.class.php` и добавьте в него следующий код:

```
<?php
// этот класс получает полосу RSS и выполняет преобразование XSLT
class CRssReader
{
    private $mXml;
    private $mXsl;

    // конструктор - создает объект XML для заданной полосы
    function __construct($szFeed)
    {
        // загрузить полосу в объект SimpleXML
        $this->mXml = simplexml_load_file(urldecode($szFeed));
        // загрузить содержимое файла XSL в объект SimpleXML
        $this->mXsl = simplexml_load_file('rss_reader.xsl');
    }

    // на основе полученной полосы создает документ XML
```

```
    public function getFormattedXML()
    {
        // создать объект XSLTProcessor
        $proc = new XSLTProcessor;
        // присоединить XSL
        $proc->importStyleSheet($this->mXsl);
        // выполнить преобразование и вернуть результат в виде строки XML
        return $proc->transformToXML($this->mXml);
    }
}
?>
```

4. **Создайте новый файл с именем** `rss_reader.xsl` **и добавьте в него следующий код:**

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <dl>
      <xsl:for-each select="rss/channel/item">
        <dt><h3><xsl:value-of select="title" /></h3></dt>
        <dd>
          <span class="date"><xsl:value-of select="pubDate" /></span>
          <p>
            <xsl:value-of select="description" />
            <br />
            <xsl:element name="a">
              <xsl:attribute name = "href">
              <xsl:value-of select="link" />
              </xsl:attribute>
              Читать всю статью
            </xsl:element>
          </p>
        </dd>
      </xsl:for-each>
    </dl>
  </xsl:template>
</xsl:stylesheet>
```

5. **Теперь добавьте стандартный модуль обработки ошибок** `error_handler.php`**. Его можно скопировать из каталога предыдущего примера. Вот этот модуль:**

```
<?php
// установить функцию error_handler, как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
```

```
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

6. **В каталоге** `rss_reader` **создайте файл с именем** `config.php`**, куда надо добавить список лент, которые будут собираться нашим приложением.**

```
<?php
// добавить некоторые ленты новостей
$feeds = array ('0' => array('title' => 'CNN Technology',
                'feed' => 'http://rss.cnn.com/rss/cnn_tech.rss'),
                '1' => array('title' => 'BBC News',
                'feed' =>
'http://news.bbc.co.uk/rss/newsonline_uk_edition/front_page/rss.xml'),
                '2' => array('title' => 'Wired News',
                'feed' =>
'http://wirednews.com/news/feeds/rss2/0,2610,3,00.xml'));
?>
```

7. **Создайте файл** `index.php` **и добавьте в него следующий код:**

```
<?php
// загрузить список лент
require_once ('config.php');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html;
                                          charset=utf-8" />
        <title> Чтение лент RSS средствами AJAX</title>
        <link rel="stylesheet" type="text/css" href="rss_reader.css"/>
        <script src="rss_reader.js" type="text/javascript"></script>
    </head>
    <body>
        <h1>Чтение лент RSS средствами AJAX</h1>
        <div id="feeds">
            <h2>Ленты</h2>
            <ul id="feedList">
            <?php
            // отобразить список лент
            for ($i = 0; $i < count($feeds); $i++)
            {
```

```
                echo '<li id="feed-' . $i . '"><a href="javascript:void(0);" ';
                echo 'onclick="getFeed(document.getElementById(\'feed-' . $i .
                      '\'), \'' . urlencode($feeds[$i]['feed']) . '\');">';
                 echo $feeds[$i]['title'] . '</a></li>';
               }
               ?>
            </ul>
            </div>
            <div id="content">
               <div id="loading" style="display:none">Загрузка ленты...</div>
                <div id="feedContainer" style="display:none"></div>
                <div id="home">
                    <h2>О программе чтения лент RSS</h2>
                    <p>
                    Программа чтения лент RSS – это всего лишь простое
                    приложение, которое предоставляет минимум
                    возможностей для чтения лент RSS.
                    </p>
                    <p>
                    Это приложение представлено в качестве
                    обучающего упражнения к книге
                   <a href="https://www.packtpub.com/ajax_php/book"> Building
                    Responsive Web Applications with AJAX and PHP</a>
                    (Packt Publishing, 2006).
                    </p>
                </div>
            </div>
        </body>
    </html>
```

8. **Создайте файл с именем** `rss_reader.js` **(для клиентской стороны) и добавьте в него следующий код:**

```
// ссылка на экземпляр XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// если содержит значение true, то выводятся подробные сообщения об ошибках
var showErrors = true;
// создает экземпляр XMLHttpRequest
function createXmlHttpRequestObject()
{
    // ссылка на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
```

```
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попробовать создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {} // игнорировать возможные ошибки
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// эта функция выводит сообщение об ошибке
function displayError($message)
{
    // игнорировать ошибку, если showErrors = false
    if (showErrors)
    {
        // отключить вывод ошибок
        showErrors = false;
        // вывести сообщение об ошибке
        alert("Обнаружена ошибка: \n" + $message);
    }
}

// извлекает заголовки из ленты новостей и отображает их
function getFeed(feedLink, feed)
{
    // продолжать, только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // попытаться соединиться с сервером
        try
        {
            if (xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
            {
```

```
                    /* получить число лент и обойти их в цикле, меняя имя
                       класса вмещающего элемента (<li>). */
                    var numberOfFeeds =
                       document.getElementById("feedList").childNodes.length;
                    for (i = 0; i < numberOfFeeds; i++)
                  document.getElementById("feedList").childNodes[i].className =
                        "";
                    // изменить имя класса для ленты, по которой был
                    // произведен щелчок, чтобы она оказалась подсвеченной
                    feedLink.className = "active";
                    // вывести сообщение «Загрузка...» на время загрузки лент
                    document.getElementById("loading").style.display = "block";
                    // послать серверу запрос на выполнение необходимых действий
                    params = "feed=" + feed;
                    xmlHttp.open("POST", "rss_reader.php", true);
                    xmlHttp.setRequestHeader("Content-Type",
                                        "application/x-www-form-urlencoded");
                    xmlHttp.onreadystatechange = handleHttpGetFeeds;
                    xmlHttp.send(params);
                }
                else
                {
                    // если соединение занять, повторить попытку через 1 секунду
                   setTimeout("getFeed('" + feedLink + "', '" + feed + "');",
                               1000);
                }
            }
            // вывести сообщение об ошибке в случае неудачи
            catch (e)
            {
                displayError(e.toString());
            }
        }
    }

    // эта функция принимает ответ HTTP
    function handleHttpGetFeeds()
    {
        // продолжать только если ответ пришел полностью
        if (xmlHttp.readyState == 4)
        {
            // продолжать только если статус HTTP = «OK»
            if (xmlHttp.status == 200)
            {
                try
                {
                    displayFeed();
                }
                catch(e)
                {
                    // вывести сообщение об ошибке
```

```
                displayError(e.toString());
            }
        }
        else
        {
            displayError(xmlHttp.statusText);
        }
    }
}

// обрабатывает ответ сервера
function displayFeed()
{
    // прочитать ответ сервера как простой текст, чтобы убедиться
    // в отсутствии ошибок
    var response = xmlHttp.responseText;
    // ошибка сервера?
    if (response.indexOf("ERRNO") >= 0
            || response.indexOf("error:") >= 0
            || response.length == 0)
        throw(response.length == 0 ? "Пустой ответ сервера." : response);
    // скрыть сообщение «Загрузка...» после получения ленты
    document.getElementById("loading").style.display = "none";
    // добавить преобразованное содержимое XML к существующей структуре DOM
    var titlesContainer = document.getElementById("feedContainer");
    titlesContainer.innerHTML = response;
    // сделать видимым элемент, вмещающий ленту
    document.getElementById("feedContainer").style.display = "block";
    // очистить текст домашней страницы
    document.getElementById("home").innerHTML = "";
}
```

9. **Создайте файл с именем** `rss_reader.css` **и добавьте в него следующий код:**

```
body
{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 12px;
}

h1
{
    color: #ffffff;
    background-color: #3366CC;
    padding: 5px;
}

h2
{
    margin-top: 0px;
}

h3
```

```
{
    margin-bottom: 0px;
}

li
{
    margin-bottom: 5px;
}

div
{
    padding: 10px;
}

a, a:visited
{
    color: #3366CC;
    text-decoration: underline;
}

a:hover
{
    color: #ffffff;
    background-color: #3366CC;
    text-decoration: none;
}

.active a
{
    color: #ffffff;
    background-color: #3366CC;
    text-decoration: none;
}

.active a:visited
{
    color: #ffffff;
    background-color:#3366CC;
    text-decoration:none;
}

.active a:hover
{
    color:#ffffff;
    background-color: #3366CC;
    text-decoration: none;
}

#feeds
{
    display: inline;
    float: left;
    width: 150px;
    background-color: #f4f4f4;
    border:1px solid #e6e6e6;
```

```
}

#content
{
    padding-left:170px;
    border:1px solid #f1f1f1;
}

#loading
{
    float: left;
    display: inline;
    width: 410px;
    background-color: #fffbb8;
    color: #FF9900;
    border: 1px solid #ffcc00;
    font-weight: bold;
}

.date
{
    font-size: 10px;
    color: #999999;
}
```

10. Откройте страницу *http://localhost/ajax/rss_reader* в броузере. Она должна выглядеть примерно так, как показано на рис. 9.3. Щелкнув мышью по одной из ссылок, вы должны получить нечто похожее на рис. 9.2.

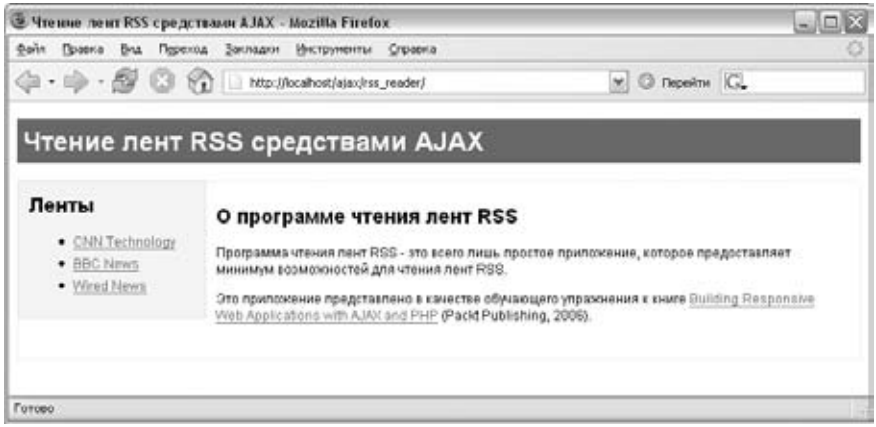

Рис. 9.3. *Первая страница приложения чтения лент RSS*

Что происходит внутри?

В таком исполнении приложение не может рассматриваться как профессиональное решение задачи, однако оно доказывает саму возможность чтения лент новостей. Чтобы достичь такого результата, потре-

бовалось не так много программного кода, а добавить любые функциональные возможности будет не так уж сложно.

Пользовательский интерфейс этого приложения достаточно прост, работа его начинается с файла index.php. Сначала мы должны подключить файл config.php, где описаны наши ленты новостей, которые будут отображаться в левой колонке страницы. Ленты описаны в виде ассоциативного массива массивов. Ключи основного массива представляют собой числа, начиная с 0, а значения – массивами, в которых ключами служат заголовки лент, а значениями – их URL. Массив лент $feeds описан следующим образом:

```
$feeds = array ('0' => array('title' => 'CNN Technology',
                'feed' => 'http://rss.cnn.com/rss/cnn_tech.rss'),
                '1' => array('title' => 'BBC News',
                'feed' =>
'http://news.bbc.co.uk/rss/newsonline_uk_edition/front_page/rss.xml'),
                '2' => array('title' => 'Wired News',
                'feed' =>
'http://wirednews.com/news/feeds/rss2/0,2610,3,00.xml'));
```

В более удобочитаемом виде это описание можно представить как:

Таблица 9.1. Содержимое массива $feeds

№	Заголовок полосы (title)	URL полосы (feed)
0	CNN Technology	*http://rss.cnn.com/rss/cnn_tech.rss*
1	BBC News	*http://news.bbc.co.uk/rss/newsonline_uk_edition/front_page/rss.xml*
2	Wired News	*http://wirednews.com/news/feeds/rss2/0,2610,3,00.xml*

Мы решили хранить информацию о лентах новостей в таком виде ради сохранения простоты приложения, но код нетрудно изменить так, чтобы те же сведения хранились в базе данных.

В файле index.php мы в цикле просматриваем все ленты и отображаем их в виде несортированного списка из ссылок в элементах <li>. Для каждой ссылки мы назначаем обработчик события onclick – функцию getFeed. Она принимает два параметра: идентификатор элемента <li> и URL ленты. Идентификатор нам необходим для того, чтобы подсветить выбранный заголовок ленты, а URL – чтобы передать его серверу в виде параметра в запросе HTTP. Функция urlencode обеспечивает преобразование URL в безопасное представление для передачи серверу, который в свою очередь выполнит обратное преобразование с помощью функции urldecode.

Два замечания по поводу файла index.php:

- Изначально невидимый элемент , с идентификатором id="loading", будет отображаться во время получения ленты, чтобы про-

информировать пользователя о том, что идет загрузка ленты новостей. Это бывает удобно при работе через медленное соединение или с медленными серверами, когда время загрузки достаточно велико.

```
<div id="loading" style="display:none">Загрузка ленты...</div>
```

- Элемент `<div>` с `id="feedContainer"` – это контейнер, куда будет загружена лента новостей. Лента будет динамически вставлена внутрь элемента `div`.

```
<div id="feedContainer" style="display:none"></div>
```

Файл `rss_reader.js` содержит стандартный программный код инициализации объекта `XMLHttpRequest`, посылки запроса и приема ответа. Передача запроса HTTP выполняется функцией `getFeed`. Она сначала обходит в цикле все ссылки на ленты и снимает подсветку, подменяя имя класса CSS пустой строкой. А затем подсвечивает активную ссылку:

```
var numberOfFeeds = document.getElementById("feedList").childNodes.length;
for (i = 0; i < numberOfFeeds; i++)
    document.getElementById("feedList").childNodes[i].className = "";
// изменить имя класса для ленты, по которой был произведен
// щелчок, чтобы она оказалась подсвеченной
feedLink.className = "active";
```

На следующем этапе выводится сообщение «Загрузка ленты...»:

```
// вывести сообщение «Загрузка...» на время загрузки лент
document.getElementById("loading").style.display = "block";
```

В заключение отправляется запрос HTTP с заголовком ленты в качестве параметра:

```
// послать серверу запрос на выполнение необходимых действий
params = "feed=" + feed;
xmlHttp.open("POST", "rss_reader.php", true);
xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xmlHttp.onreadystatechange = handleHttpGetFeeds;
xmlHttp.send(params);
```

Сценарий `rss_reader.php` создает экземпляр класса `CRssReader` и отображает документ XML, полученный в результате XSL-преобразования. Основная работа выполняется в следующих строках (код, который очищает выходной буфер и предотвращает возможность кэширования страницы, мы опустили):

```
$reader = new CRssReader(urldecode($_POST['feed']));
echo $reader->getFormattedXML();
```

Определение класса `CRssReader` находится в файле `rss_reader.class.php`. Этот класс принимает документ XML и форматирует его. Однако получить удаленный файл с помощью нового, появившегося в PHP 5 расширения SimpleXML – это только полдела. Мы также должны загрузить шаблон XSL и преобразовать принятый файл XML.

Конструктор класса загружает файл XML и сохраняет его в члене класса с именем `$mXml`, а файл XSL – в члене класса `$mXsl`:

```
// конструктор, создающий объект XML для заданной полосы
function __construct($szFeed)
{
    // загрузить полосу в объект SimpleXML
    $this->mXml = simplexml_load_file(urldecode($szFeed));
    // загрузить содержимое файла XSL в объект SimpleXML
    $this->mXsl = simplexml_load_file('rss_reader.xsl');
}
```

Функция `getFormattedXML()` создает новый объект `XSLTProcessor`, с помощью которого выполняет XSL-преобразование. Метод `transformToXML` просто возвращает отформатированный документ XML после применения XSL-преобразования:

```
// на основе полученной полосы создает документ XML
public function getFormattedXML()
{
    // создать объект XSLTProcessor
    $proc = new XSLTProcessor;
    // присоединить XSL
    $proc->importStyleSheet($this->mXsl);
    // выполнить преобразование и вернуть результат в виде строки XML
    return $proc->transformToXML($this->mXml);
}
```

Все, что нам требуется от XSL-преобразования, – это обойти в цикле все «записи» документа XML и вывести данные, содержащиеся в них. Каждая запись ограничена парой тегов `<item>` и `</item>`.

Цикл в файле `rss_reader.xsl` мы описали следующим образом:

```
<xsl:for-each select="rss/channel/item">
```

Например, чтобы вывести текущий заголовок, мы написали:

```
<h3><xsl:value-of select="title" /></h3>
```

Обратите внимание на то, как создается новый элемент `<a>` средствами XSLT:

```
<xsl:element name="a">
  <xsl:attribute name = "href">
    <xsl:value-of select="link" />
  </xsl:attribute>
  прочитать всю статью
</xsl:element>
```

Таким способом мы создаем ссылки на статьи, расположенные на вебсайтах.

В приложении также имеется небольшой объем кода CSS, который определяет форматирование страницы в соответствии с нашими желаниями. Если вы заглянете в файл `rss_reader.css`, вам все станет понятно.

Подведение итогов

Сегодня Всемирная паутина не такая, какой была вчера, а завтра, разумеется, она будет не такой, как сегодня. Вчерашняя сеть представляла собой набор страниц, связанных между собой. Все они были статическими и всю необходимую информацию содержали внутри себя. Главное, что сегодня характеризует Всемирную паутину, – это обмен информацией между веб-сайтами и/или приложениями.

Основываясь на знаниях, полученных из этой главы, вы сможете создать еще лучшую программу чтения новостей RSS, но зачем останавливаться на этом? Вы владеете мощной технологией, которая позволит вам строить превосходные приложения, которые способны оказать влияние на развитие Сети завтрашнего дня!

10

Технология drag-and-drop в AJAX

Когда впервые на веб-сайтах появилась[1] возможность перетаскивать объекты мышью, люди смотрели на нее с удивлением. На сайтах она смотрелась действительно необычно! С тех пор JavaScript превратился из языка сценариев «для создания визуальных эффектов» в мощный и стандартизованный язык «решения сложных задач».

Уже создано немало интегрированных систем и инструментальных пакетов для работы с JavaScript, но продолжают появляться все новые и новые. Одним из самых популярных пакетов библиотек JavaScript является **script.aculo.us**. Он позволяет реализовать поразительные эффекты на веб-страницах, в чем вам помогут убедиться примеры, представленные на официальной веб-странице по адресу *http://script.aculo.us*. Script.aculo.us – это пакет библиотек JavaScript с открытыми исходными текстами, распространяемый на основе лицензии MIT-like, которая допускает использование программного продукта сколь угодно долго и без каких-либо ограничений при условии, что вы включаете в ее состав примечание об авторских правах. Библиотеку script.aculo.us можно скачать со страницы *http://script.aculo.us/downloads*. Документация к ней расположена по адресу *http://wiki.script.aculo.us*.

В этой главе на примере создания AJAX-приложения с поддержкой баз данных и сортируемых списков показано, как можно использовать возможности пакета **script.aculo.us**.

Применение механизма перетаскивания во Всемирной паутине

Рассмотрев некоторые веб-приложения с поддержкой механизма drag-and-drop, мы обнаружили по крайней мере две ситуации, в которых

[1] К тому времени техника перетаскивания уже была много лет известна в локальных приложениях и стала естественной и привычной. – *Примеч. науч. ред.*

возможность перетаскивания объектов мышью делает пользовательский интерфейс более привлекательным и облегчает взаимодействие между машиной и человеком. Механизм перетаскивания с успехом может применяться для реализации:[1]

- Покупательской корзины
- Сортируемых списков

Покупательские корзины

Вы, вероятно, уже знакомы с интернет-магазинами. С началом бума AJAX на свет появилось новое поколение покупательских корзин, которые можно наполнять простым перетаскиванием товаров мышью, обходясь без нажатия кнопки «Добавить в корзину». Реальную ценность этой «особенности» можно поставить под сомнение (моя бабушка по-прежнему предпочитает пользоваться кнопкой – она просто не знает, как перетаскивать объекты мышью), однако зрительный эффект, создаваемый ею, весьма внушителен.

На некоторых веб-сайтах эта возможность уже реализована. Один из примеров – интернет-магазин Panic Goods, торгующий футболками! Адрес магазина: *http://www.panic.com/goods*.

Обратили внимание на голубоватую полосу внизу экрана? Это и есть покупательская корзина. Попробуйте перетащить несколько футболок из каталога в корзину и посмотрите, что будет происходить с ней. Футболки выстраиваются в корзине в ряд, и вы видите, что вы выбрали и сколько все это стоит. Перетащите выбранные товары за пределы голубоватой полосы, чтобы удалить их из корзины. Здорово, правда?

Сортируемые списки

С одним из таких списков мы сталкиваемся практически ежедневно; это списки заданий, которые надо выполнить (списки **to-do**). Обычно они составляются на разграфленных желтых листах бумаги, а иногда для создания таких списков даже применяется специальное ПО.

Но раз уж стало так много веб-приложений, то должна отыскаться хотя бы дюжина приложений, реализующих списки to-do! Здесь я упомяну лишь приложение Ta-da Lists (*http://www.tadalist.com*), разработанное компанией **37signals**. Эта компания фактически повторно изобрела всю концепцию веб-приложений и подняла ее на новый уровень. Ta-da Lists, один из первых ее продуктов, представляет собой инструмент создания различных списков to-do, каждый из которых состоит из отдельного набора пунктов (заданий, которые необходимо выполнить). Это очень удобный инструмент, и многие пользуются им несмотря на

[1] Эти две возможности демонстрируются на уже упоминавшемся сайте *http://demo.script.aculo.us/shop* и на *http://demo.script.aculo.us/ajax/sortable_elements*. – *Примеч. науч. ред.*

то, что имеются более совершенные продукты компании 37signals, такие как Basecamp (*http://www.basecamphq.com*) и Backpack (*http://www. backpackit.com*).

Ta-da Lists обладает интуитивно понятным пользовательским интерфейсом, в нем реализован удобный набор команд. Однако приложение страдает от отсутствия одной элементарной возможности, которая еще больше повысила бы удобство и простоту использования. Продукт не позволяет перетаскивать мышью элементы списка. Чтобы изменить порядок следования пунктов в списке Ta-da Lists, нужно щелкнуть по ссылке, после чего загрузится новая страница, на которой в каждом элементе списка появятся четыре кнопки (переместить в начало, переместить выше, переместить ниже, переместить в конец).

Приложение в такой его реализации неплохо справляется со своей задачей, но оно могло бы быть еще быстрее и еще удобнее. Компания 37signals реализовала улучшенные функциональные возможности в Basecamp, где списки to-do предоставляют возможность перемещения пунктов с помощью мыши, что лишний раз доказывает более высокие потребительские качества концепции drag-and-drop.

Создание приложения с поддержкой механизма перетаскивания

Итак, мы хотим на базе технологии AJAX создать приложение управления сортируемыми списками с поддержкой механизма перетаскивания. Единственное, что отличает это приложение от других, вошедших в состав книги, – это то, что здесь мы собираемся использовать две дополнительные интегрированные системы JavaScript – **Prototype** и **script.aculo.us**.

«Prototype – это интегрированная система JavaScript, цель которой заключается в том, чтобы облегчить разработку динамических веб-приложений.» Она была создана Сэмом Стефенсоном (Sam Stephenson) и быстро доросла до уровня интегрированной системы благодаря широте своих функциональных возможностей.

Примечание

Prototype распространяется на основе лицензии MIT-like и доступна по адресу *http://prototype.conio.net*.

Более подробно о Prototype вы сможете узнать из учебного руководства, размещенного по адресу *http://www.particletree.com/features/quick-guide-to-prototype*.

Интегрированная система Prototype обладает следующими характеристиками:

- Она полностью написана в объектно-ориентированном стиле.
- Включает в себя набор сервисных функций.

- Содержит вспомогательные функции для работы с формами.
- Поддерживает технологию AJAX.
- Имеет в своем составе объект PeriodicalExecuter, который позволяет вызывать заданную функцию через установленные интервалы времени.

Другой пионер в области разработки приложений на языке JavaScript, Томас Фукс (Thomas Fuchs), создал замечательную библиотеку script.aculo.us, с помощью которой можно реализовать захватывающие визуальные эффекты. Мы задействуем некоторые из ее функциональных возможностей в нашем приложении (если быть точнее, возможность перетаскивания объектов мышью). Библиотека script.aculo.us построена на базе интегрированной системы Prototype и наследует, таким образом, все ее возможности.

Библиотека script.aculo.us обладает следующими характеристиками:

- Она полностью написана в объектно-ориентированном стиле.
- Реализует визуальные эффекты (плавного появления, плавного исчезновения, увеличения, уменьшения, разворачивания вниз, сворачивания вверх, дрожания и прочие).
- Поддерживает механизм перетаскивания.
- Поддерживает механизм автодополнения.
- Поддерживает возможность редактирования по месту.
- Включает в себя элементы управления – бегунки.

Приложение, которое мы создадим, будет представлять собой небольшую программу управления заданиями, которая позволит нам создавать и удалять новые задания и переупорядочивать задания в списке. Эта программа будет обладать следующими характеристиками:

- В основе будет лежать база данных.
- Элементы списка можно будет перетаскивать мышью.
- Добавление новых заданий будет производиться средствами технологии AJAX.
- Немедленное изменение базы данных после перетаскивания элемента списка.
- Будет реализована возможность удаления элементов списка путем перетаскивания их в специальную область.

Посмотрим, как будет выглядеть наше приложение (рис. 10.1).

Перемещение одного элемента списка по экрану будет вынуждать другие элементы изменять свое положение в списке.

Перемещение задания в область DROP HERE TO DELETE (чтобы удалить, переместите задание сюда) будет вызывать диалог подтверждения операции удаления, прежде чем приложение выполнит эту операцию (рис. 10.2).

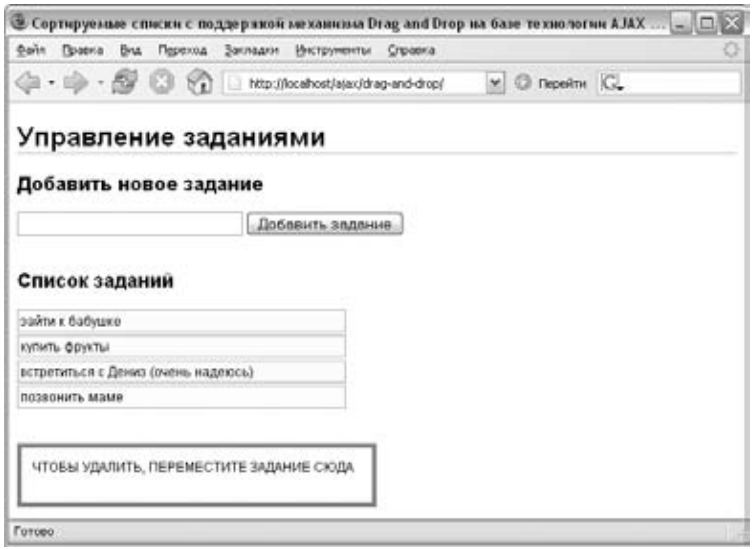

Рис. 10.1. *Операции добавления, удаления и переупорядочивания заданий с простым визуальным интерфейсом*

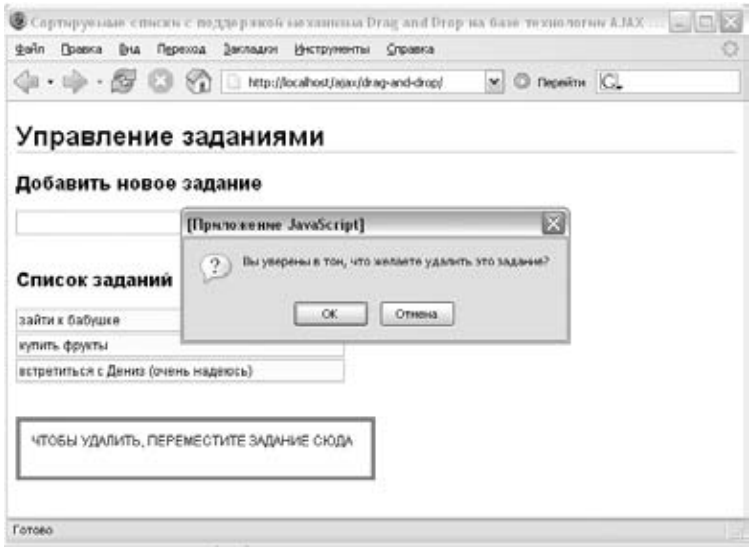

Рис. 10.2. *Прежде чем удалить задание, вы должны подтвердить свое желание*

Время действовать – приложение управления заданиями с привлечением технологии AJAX

1. Соединитесь с базой данных `ajax` и создайте таблицу с именем `tasks` с помощью следующего оператора:

```
CREATE TABLE tasks (
    id INT UNSIGNED NOT NULL AUTO_INCREMENT,
    order_no INT UNSIGNED NOT NULL default '0',
    description VARCHAR(100) NOT NULL default '',
    PRIMARY KEY (id)
);
```

2. **В каталоге** `ajax` **создайте подкаталог с именем** `drag-and-drop`**.**

3. **В каталоге** `drag-and-drop` **создайте файл с именем** `config.php` **и добавьте в него параметры соединения с базой данных:**

```
<?php
// сведения, необходимые для соединения с базой данных
define('DB_HOST', 'localhost');
define('DB_USER', 'ajaxuser');
define('DB_PASSWORD', 'practical');
define('DB_DATABASE', 'ajax');
?>
```

4. **Теперь добавьте стандартный файл сценария обработки ошибок** `error_handler.php`**. Его можно скопировать из упражнения к предыдущей главе. Вот содержимое этого файла:**

```
<?php
// установить функцию error_handler как обработчик ошибок по умолчанию
set_error_handler('error_handler', E_ALL);
// функция обработки ошибок
function error_handler($errNo, $errStr, $errFile, $errLine)
{
    // удалить выходные данные, которые уже были созданы
    if(ob_get_length()) ob_clean();
    // вывести сообщение об ошибке
    $error_message = 'ERRNO: ' . $errNo . chr(10) .
                     'TEXT: ' . $errStr . chr(10) .
                     'LOCATION: ' . $errFile .
                     ', line ' . $errLine;
    echo $error_message;
    // прервать дальнейшую работу сценария PHP
    exit;
}
?>
```

5. **Скачайте архив с библиотекой** `script.aculo.us` **с сайта** ***http://script.aculo.us/downloads*** **и распакуйте его в каталог** `drag-and-drop`**. Измените имя каталога с библиотекой, которое будет выглядеть примерно как** `scriptaculous-js-`*`x.y.z`*`, на` `scriptaculous`**.**

6. **Создайте файл с именем** `index.php` **и добавьте в него следующий код:**

```
<?php
require_once ('taskslist.class.php');
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>
        Сортируемые списки с поддержкой механизма Drag and Drop
        на базе технологии AJAX
    </title>
    <link href="drag-and-drop.css" rel="stylesheet" type="text/css" />
    <script src="drag-and-drop.js" type="text/javascript"></script>
    <script src="scriptaculous/lib/prototype.js" type="text/javascript">
    </script>
   <script src="scriptaculous/src/scriptaculous.js" type="text/javascript">
    </script>
  </head>
  <body onload="startup()">
    <h1>Управление заданиями</h1>
    <h2>Добавить новое задание</h2>
    <div>
      <input type="text" id="txtNewTask" name="txtNewTask"
             size="30" maxlength="100" onkeydown="handleKey(event)"/>
      <input type="button" name="submit" value="Добавить задание"
             onclick="process('txtNewTask', 'addNewTask')" />
    </div>
    <br />
    <h2>Список заданий</h2>
    <ul id="tasksList" class="sortableList"
        onmouseup="process('tasksList', 'updateList')">
      <?php
      $myTasksList = new TasksList();
      echo $myTasksList->BuildTasksList();
      ?>
    </ul>
    <br /><br />
    <div id="trash">
      ЧТОБЫ УДАЛИТЬ, ПЕРЕМЕСТИТЕ ЗАДАНИЕ СЮДА
      <br /><br />
    </div>
  </body>
</html>
```

7. **Создайте файл с именем** `taskslist.class.php` **и добавьте в него следующий код:**

```
<?php
// загрузить модуль обработки ошибок и параметры доступа к базе данных
require_once ('error_handler.php');
require_once ('config.php');
// этот класс создает список заданий и выполняет
// операции добавления/удаления/переупорядочивания
class TasksList
{
```

```
// дескриптор соединения с базой данных
private $mMysqli;

// конструктор, открывает соединение с базой данных
function __construct()
{
    // установить соединение с базой данных
    $this->mMysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD,
                                DB_DATABASE);
}

// деструктор, закрывает соединение с базой данных
public function __destruct()
{
    $this->mMysqli->close();
}

// создает список заданий
public function BuildTasksList()
{
    // инициализировать пустой список
    $myList = '';
    // построить запрос
    $result = $this->mMysqli->query('SELECT * FROM tasks ' .
                                    'ORDER BY order_no ASC');
    // создать список заданий с помощью элементов <li>
    while ($row = $result->fetch_assoc())
    {
        $myList .= '<li id="' . htmlentities($row['id']) . '">' .
                   htmlentities($row['description']) . '</li>';
    }
    // вернуть список
    return $myList;
}

// выполняет запрошенную операцию на стороне сервера
public function Process($content, $action)
{
    // выполнить действие, запрошенное клиентом
    switch($action)
    {
        // переупорядочить список заданий
        case 'updateList':
            // извлечь дополнительные сведения об операции
            $new_order = explode('_', $content);
            // обновить список
            for ($i=0; $i < count($new_order); $i++)
            {
                // экранировать специальные символы в данных,
                // пришедших от клиента
                $new_order[$i] =
                  $this->mMysqli->real_escape_string($new_order[$i]);
                // записать изменения в базу данных
```

```
                    $result = $this->mMysqli->query
                                ('UPDATE tasks SET order_no="' .
                                $i . '" WHERE id="' . $new_order[$i] . '"');
                }
                $updatedList = $this->BuildTasksList();
                return $updatedList;
                break;
            // добавить новое задание
            case 'addNewTask':
                // экранировать специальные символы
                $task = trim($this->mMysqli->real_escape_string($content));
                // продолжать, только если имя задание – не пустая строка
                if ($task)
                {
                    // найти задание, для которого order_no
                    // имеет наибольшее значение
                    $result = $this->mMysqli->query
                                ('SELECT (MAX(order_no) + 1) ' .
                                 'AS order_no FROM tasks');
                    $row = $result->fetch_assoc();
                    // если таблица пустая, в order_no будет
                    // содержаться пустая ссылка
                    $order = $row['order_no'];
                    if (!$order) $order = 1;
                    // вставить новое задание в конец списка
                    $result = $this->mMysqli->query
                         ('INSERT INTO tasks (order_no, description) ' .
                          'VALUES ("' . $order . '", "' . $task . '")');
                    // вернуть обновленный список заданий
                    $updatedList = $this->BuildTasksList();
                    return $updatedList;
                }
                break;
            // удалить задание
            case 'delTask':
                // экранировать специальные символы
                $content = trim($this->mMysqli->
                                        real_escape_string($content));
                // удалить задание
                $result = $this->mMysqli->query
                            ('DELETE FROM tasks WHERE id="' .
                                          $content . '"');
                $updatedList = $this->BuildTasksList();
                return $updatedList;
                break;
        }
    }
}
?>
```

8. **Создайте файл с именем** `drag-and-drop.php` **и добавьте в него следующий код:**

```
<?php
// загрузить вспомогательный класс
require_once ('taskslist.class.php');
// создать объект TasksList
$myTasksList = new TasksList();
// прочитать входные параметры
$action = $_GET['action'];
$content = $_GET['content'];
// очистить выходной буфер
if(ob_get_length()) ob_clean();
// предотвратить возможность кэширования страницы броузерами
header('Expires: Fri, 25 Dec 1980 00:00:00 GMT'); // время в прошлом
header('Last-Modified: ' . gmdate( 'D, d M Y H:i:s') . 'GMT');
header('Cache-Control: no-cache, must-revalidate');
header('Pragma: no-cache');
header('Content-Type: text/html');
// выполнить операцию, запрошенную клиентом
// и вернуть обновленный список заданий
echo $myTasksList->Process($content, $action);
?>
```

9. **Создайте файл с именем** `drag-and-drop.js` **и добавьте в него следующий код:**

```
// ссылка на экземпляр XMLHttpRequest
var xmlHttp = createXmlHttpRequestObject();
// когда установлено значение true, выводятся подробные
// сообщения об ошибках
var showErrors = true;
// инициализировать буфер запросов
var cache = new Array();

// создает экземпляр XMLHttpRequest
function createXmlHttpRequestObject()
{
    // ссылка на объект XMLHttpRequest
    var xmlHttp;
    // эта часть кода должна работать во всех броузерах, за исключением
    // IE6 и более старых его версий
    try
    {
        // попытаться создать объект XMLHttpRequest
        xmlHttp = new XMLHttpRequest();
    }
    catch(e)
    {
        // предполагается, что в качестве броузера используется
        // IE6 или более старая его версия
        var XmlHttpVersions = new Array("MSXML2.XMLHTTP.6.0",
                                        "MSXML2.XMLHTTP.5.0",
                                        "MSXML2.XMLHTTP.4.0",
                                        "MSXML2.XMLHTTP.3.0",
```

```
                                        "MSXML2.XMLHTTP",
                                        "Microsoft.XMLHTTP");
        // попробовать все возможные prog id,
        // пока какая-либо попытка не увенчается успехом
        for (var i=0; i<XmlHttpVersions.length && !xmlHttp; i++)
        {
            try
            {
                // попытаться создать объект XMLHttpRequest
                xmlHttp = new ActiveXObject(XmlHttpVersions[i]);
            }
            catch (e) {} // игнорировать возможные ошибки
        }
    }
    // вернуть созданный объект или вывести сообщение об ошибке
    if (!xmlHttp)
        alert("Ошибка создания объекта XMLHttpRequest.");
    else
        return xmlHttp;
}

// эта функция отображает сообщения об ошибках
function displayError($message)
{
    // игнорировать ошибки, если в showErrors записано значение false
    if (showErrors)
    {
        // отключить вывод сообщений об ошибках
        showErrors = false;
        // вывести сообщение
        alert("Возникла ошибка: \n" + $message);
    }
}

// код, который обращается к библиотеке Scriptaculous для описания
// сортируемого списка и области сброса удаляемых элементов
function startup()
{
    // преобразовать неупорядоченный список в сортируемый, элементы
    // которого могут перемещаться с помощью мыши
    Sortable.create("tasksList", {tag:"li"});
    // описать область сброса для удаления заданий
    Droppables.add("trash",
    {
        onDrop: function(element)
        {
            var deleteTask =
              confirm("Вы уверены в том, что желаете удалить это задание?")
            if (deleteTask)
            {
                Element.hide(element);
                process(element.id, "delTask");
```

```
            }
        }
    });
}

// преобразует значения id элементов списка (<li> в последовательную форму)
function serialize(listID)
{
    // подсчитать число элементов списка
    var length = document.getElementById(listID).childNodes.length;
    var serialized = "";
    // обойти элементы в цикле
    for (i = 0; i < length; i++)
    {
        // получить ссылку на текущий элемент
        var li = document.getElementById(listID).childNodes[i];
        // извлечь id текущего элемента без текстовой части
        var id = li.getAttribute("id");
        // добавить в строку идентификаторов только номер
        serialized += encodeURIComponent(id) + "_";
    }
    // вернуть массив без завершающего символа '_'
    return serialized.substring(0, serialized.length - 1);
}

// отправляет запрос серверу
function process(content, action)
{
    // продолжать, только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // очистить строку запроса
        params = "";
        // экранировать специальные символы для обеспечения безопасной
        // передачи запроса серверу
        content = encodeURIComponent(content);
        // подготовить к отправке параметры запроса, в зависимости
        // от типа выполняемой операции
        if (action == "updateList")
            params = "?content=" + serialize(content) +
                                                    "&action=updateList";
        else if (action == "addNewTask")
        {
            // подготовить сведения о новом задании
            var newTask = trim
              (encodeURIComponent(document.getElementById(content).value));
            // не добавлять пустое задание
            if (newTask)
                params = "?content=" + newTask + "&action=addNewTask";
        }
        else if (action =="delTask")
            params = "?content=" + content + "&action=delTask";
```

```
            // не добавлять в буфер пустую строку параметров
            if (params) cache.push(params);
            // попытаться соединиться с сервером
            try
            {
                // продолжать, только если соединение не занято
                // и буфер запросов не пуст
                if ((xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
                        && cache.length>0)
                {
                    // извлечь из буфера следующий набор значений
                    var cacheEntry = cache.shift();
                    // инициировать запрос
                    xmlHttp.open("GET", "drag-and-drop.php" + cacheEntry, true);
                    xmlHttp.setRequestHeader("Content-Type",
                                             "application/x-www-form-urlencoded");
                    xmlHttp.onreadystatechange = handleRequestStateChange;
                    xmlHttp.send(null);
                }
                else
                {
                    setTimeout("process();", 1000);
                }
            }
            // вывести сообщение об ошибке в случае неудачи
            catch (e)
            {
                displayError(e.toString());
            }
        }
    }

    // эта функция принимает ответ HTTP
    function handleRequestStateChange()
    {
        // когда readyState = 4, мы можем прочитать ответ сервера
        if (xmlHttp.readyState == 4)
        {
            // продолжать, только если статус HTTP равен «OK»
            if (xmlHttp.status == 200)
            {
                try
                {
                    postUpdateProcess();
                }
                catch(e)
                {
                    // вывести сообщение об ошибке
                    displayError(e.toString());
                }
            }
```

```
        else
        {
            displayError(xmlHttp.statusText);
        }
    }
}

// обрабатывает ответ сервера
function postUpdateProcess()
{
    // прочитать ответ сервера
    var response = xmlHttp.responseText;
    // ошибка сервера?
    if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)
        alert(response);
    // обновить список заданий
    document.getElementById("tasksList").innerHTML = response;
    Sortable.create("tasksList");
    document.getElementById("txtNewTask").value = "";
    document.getElementById("txtNewTask").focus();
}

/* обрабатывает событие keydown,
   чтобы определить момент нажатия на клавишу Enter */
function handleKey(e)
{
    // получить ссылку на событие
    e = (!e) ? window.event : e;
    // получить код символа нажатой клавиши
    code = (e.charCode) ? e.charCode :
         ((e.keyCode) ? e.keyCode :
         ((e.which) ? e.which : 0));
    // обработать событие keydown
    if (e.type == "keydown")
    {
        // если была нажата клавиша Enter (код 13)
        if(code == 13)
        {
            // отправить текущее сообщение
            process("txtNewTask", "addNewTask");
        }
    }
}

/* удаляет из строки начальные и конечные пробелы */
function trim(s)
{
    return s.replace(/(^\s+)|(\s+$)/g, "")
}
```

10. **Создайте файл с именем** `drag-and-drop.css` **и добавьте в него следующий код:**

```
body
{
    font-family: Arial, Helvetica, sans-serif;
    font-size: 12px;
}

ul.sortableList
{
    list-style-type: none;
    padding: 0px;
    margin: 0px;
    width: 300px;
}

ul.sortableList li
{
    cursor: move;
    padding: 2px 2px;
    margin: 2px 0px;
    border: 1px solid #00CC00;
    background-color: #F4FFF5;
}

h1
{
    border-bottom: 1px solid #cccccc;
}

#trash
{
    border: 4px solid #ff0000;
    width: 270px;
    padding: 10px;
}
```

11. Откройте страницу *http://localhost/ajax/drag-and-drop* в броузере и попробуйте выполнить несколько действий, чтобы убедиться, что приложение работает так, как мы и ожидали (см. рис. 10.1 и 10.2).

Что происходит внутри?

Добавление задания в список предполагает выполнение следующих действий:

1. Пользователь вводит название задания.
2. Когда пользователь щелкает по кнопке «Добавить задание» или нажимает клавишу Enter, на сервер отправляются данные в виде асинхронного запроса HTTP. Сценарий на сервере вставляет новое задание в базу данных и возвращает обновленный список, который вставляется в страницу кодом JavaScript.

При переупорядочивании списка заданий выполняются следующие действия:

1. Каждое задание представлено в виде элемента списка на языке XHTML: `<li>`. Пользователь начинает перетаскивать элемент списка, и при его отпускании серверу посылается запрос HTTP, который состоит из строки, содержащей идентификаторы каждого элемента списка.
2. Переупорядоченный список отображается на стороне клиента сразу после того, как сервер запишет его в базу данных.

Далее приводится порядок действий, выполняемых при удалении задания из списка:

1. Пользователь захватывает мышью элемент списка и перетаскивает его в область с надписью DROP HERE TO DELETE (чтобы удалить, переместите задание сюда).
2. После этого серверу отправляется запрос HTTP, и сервер удаляет элемент списка из базы данных, после чего элемент списка XHTML уничтожается.

Мы включили в файл `index.php` ссылки на библиотеки JavaScript, которые будут использоваться сценарием на стороне клиента:

```
<script src="drag-and-drop.js" type="text/javascript"></script>
<script src="scriptaculous/lib/prototype.js" type="text/javascript">
</script>
<script src="scriptaculous/src/scriptaculous.js" type="text/javascript">
</script>
```

Первая строка подключает основной сценарий JavaScript, выполняющий функции управления списком и реализующий технологию AJAX. Вторая строка подключает библиотеку Prototype, а третья – библиотеку script.aculo.us.

По событию `onload` в теге `<body>` вызывается функция `startup()`, которая описывает неупорядоченный список с `id="tasksList"` как сортируемый элемент (`Sortable.create`). Тем самым обеспечивается поддержка механизма drag-and-drop для элементов `<li>` внутри списка. Функция `startup()` также описывает элемент `Droppables.add`, который мы будем использовать в качестве области сброса удаляемых заданий.

Кроме того, в функции `startup()` мы описываем поведение элемента списка в области сброса:

```
onDrop: function(element)
{
    var deleteTask =
        confirm("Вы уверены, что желаете удалить это задание?")
    if (deleteTask)
    {
        Element.hide(element);
        process(element.id, "delTask");
    }
}
```

Здесь у пользователя запрашивается подтверждение на выполнение операции. Если оно будет получено, элемент исчезает с экрана и вызывается функция process, которая отправляет серверу запрос HTTP.

В файле index.php имеется небольшой участок программного кода, который на лету создает список заданий:

```
<ul id="tasksList" class="sortableList"
    onmouseup="process('tasksList', 'updateList')">
  <?php
    $myTasksList = new TasksList();
    echo $myTasksList->BuildTasksList();
  ?>
</ul>
```

Добавление нового задания производится по щелчку на кнопке «Добавить задание» или по нажатию на клавишу Enter.

Фактический запрос AJAX посылается функцией process. Эта функция выполняет запросы для всех трех операций (переупорядочивания/добавления/удаления задания), указывая выбранное действие в виде отдельного параметра запроса.

При добавлении нового задания первым параметром в функцию process передается идентификатор (id) текстового поля ввода, в котором только что было введено название задания.

```
<input type="button" name="submit" value="Добавить задание"
       onclick="process('txtNewTask', 'addNewTask')" />
```

Обновление базы данных после переупорядочивания списка выполняется по событию onmouseup, которое возникает внутри списка с id="tasksList" – нашего сортируемого списка. По событию вызывается функция process, которой в качестве первого параметра передается идентификатор списка.

```
<ul id="tasksList" class="sortableList" onmouseup="process('tasksList',
'updateList')">
```

Поскольку серверу будет посылаться массив значений, мы должны преобразовать эти значения в последовательную форму, что мы и делаем с помощью нашей функции serialize. Эта функция подсчитывает число элементов <li> в списке и затем в цикле обходит их, извлекая из элементов идентификаторы и добавляя полученные значения к строке. И наконец, функция удаляет из возвращаемого значения завершающий символ «_».

```
function serialize(listID)
{
    // подсчитать число элементов списка
    var length = document.getElementById(listID).childNodes.length;
    var serialized = "";
    // обойти элементы в цикле
```

```
    for (i = 0; i < length; i++)
    {
        // получить ссылку на текущий элемент
        var li = document.getElementById(listID).childNodes[i];
        // извлечь id текущего элемента без текстовой части
        var id = li.getAttribute("id");
        // добавить в строку идентификаторов только номер
        serialized += encodeURIComponent(id) + "_";
    }
    // вернуть массив без завершающего символа '_'
    return serialized.substring(0, serialized.length - 1);
}
```

Мы уже не раз говорили, что `XMLHttpRequest` не может одновременно обслуживать два или более запросов HTTP, поэтому, если объект занят обработкой предыдущего запроса, мы сохраняем информацию о текущем запросе в буфере для последующего использования. Такой прием бывает особенно удобен, когда подключение к сети слишком медленное. Информация о запросе сохраняется в буфере, поддерживающем структуру FIFO. К счастью, класс `Array` в JavaScript обладает всей необходимой нам функциональностью (через его методы `push` и `shift`), поэтому мы используем его в качестве буфера:

```
var cache = new Array();
```

В функции `process`, прежде чем посылать новый запрос, мы сначала собираем строку параметров из входных аргументов функции: `content` и `action`. В зависимости от содержимого аргумента `action` строка параметров запроса собирается по-разному, причем `content` (идентификатор элемента HTML) и `action` (операция, которая должна быть выполнена сервером) отделяются друг от друга символом «&»:

```
    // продолжать, только если в xmlHttp не пустая ссылка
    if (xmlHttp)
    {
        // очистить строку запроса
        params = "";
        // экранировать специальные символы для обеспечения безопасной
        // передачи запроса серверу
        content = encodeURIComponent(content);
        // подготовить к отправке параметры запроса, в зависимости
        // от типа выполняемой операции
        if (action == "updateList")
            params = "?content=" + serialize(content) + "&action=updateList";
        else if (action == "addNewTask")
        {
            // подготовить сведения о новом задании
            var newTask = trim
                (encodeURIComponent(document.getElementById(content).value));
            // не добавлять пустое задание
            if (newTask)
```

```
            params = "?content=" + newTask + "&action=addNewTask";
        }
        else if (action =="delTask")
            params = "?content=" + content + "&action=delTask";
        // не добавлять в буфер пустую строку параметров
```

Затем мы сохраняем текущий запрос в буфере:

```
        if (params) cache.push(params);
```

Эта строка добавляет новый элемент в конец буферного массива. Обратите внимание, что новый элемент добавляется в массив, только если строка `params` не пустая, что может произойти, когда функция вызывается не в результате действий пользователя, а для проверки наличия запросов, ожидающих обработки.

Затем, если объект `XMLHttpRequest` свободен и готов обслужить очередной запрос, мы вызываем функцию `shift()`, с помощью которой извлекаем очередной запрос из массива. Учтите, что это может быть не тот запрос, который только что был поставлен в очередь; в случае очередей FIFO первой обслуживается самая старая запись.

```
// попытаться соединиться с сервером
try
{
    // продолжать только если соединение не занято
    // и буфер запросов не пуст
    if ((xmlHttp.readyState == 4 || xmlHttp.readyState == 0)
            && cache.length>0)
    {
        // извлечь из буфера следующий набор значений
        var cacheEntry = cache.shift();
```

Состояние запроса HTTP 0 или 4 говорит об отсутствии активных запросов, и мы можем послать новый запрос:

```
// инициировать запрос
xmlHttp.open("GET", "drag-and-drop.php" + cacheEntry, true);
xmlHttp.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
xmlHttp.onreadystatechange = handleRequestStateChange;
xmlHttp.send(null);
```

Ответ сервера обрабатывается функцией `handleRequestStateChange`, которая в свою очередь вызывает `postUpdateProcess()`. Функция `postUpdateProcess()` извлекает ответ сервера, который будет либо сообщением об ошибке, либо будет содержать код HTML с обновленным списком заданий:

```
// прочитать ответ сервера
var response = xmlHttp.responseText;
// ошибка сервера?
if (response.indexOf("ERRNO") >= 0 || response.indexOf("error") >= 0)
    alert(response);
```

```
// обновить список заданий
document.getElementById("tasksList").innerHTML = response;
Sortable.create("tasksList");
document.getElementById("txtNewTask").value = "";
document.getElementById("txtNewTask").focus();
```

Последние две строки очищают содержимое текстового поля ввода и устанавливают на него фокус ввода.

Сценарий `drag-and-drop.php` очень прост. Он подключает файл `taskslist.class.php`, создает новый объект `TasksList` и возвращает обновленный список после того, как мы вызовем метод `Process` класса, который отвечает за выполнение одного из трех возможных действий: добавление нового задания, переупорядочивание списка и удаление задания из списка.

Файл `taskslist.class.php` содержит описание класса, который используется для выполнения необходимых действий над списком заданий на стороне сервера. Конструктор класса открывает соединение с базой данных. Далее следуют два общедоступных метода:

- `BuildTasksList` – создает список элементов, в котором каждый элемент представляет отдельное задание.
- `Process` – принимает два входных аргумента, `$content` и `$action`. Первый аргумент содержит данные, определяемые пользователем, в зависимости от содержимого второго аргумента, который сообщает сценарию, какая операция должна быть выполнена.

Когда выполняется переупорядочивание списка (`case 'updateList'`), из массива `$content`, который хранит строку с новым порядком следования элементов `<li>`, т. е. заданий, мы извлекаем значения, а затем в цикле обходим их и обновляем содержимое базы данных.

Чтобы добавить новое задание, мы сначала выполняем экранирование специальных символов в строке, полученной от пользователя с помощью метода `real_escape_string` объекта `mysqli`. Затем в базе данных находим самый большой порядковый номер и увеличиваем его на единицу. Полученное число станет порядковым номером нового задания. После этого мы добавляем задание в базу данных и возвращаем строку, содержащую порядковый номер и описание задания. Эта строка затем отсылается обратно клиенту, который на основе полученных данных построит новый элемент списка.

При удалении задания из списка (`case 'delTask'`) единственное, что нам необходимо сделать, – это удалить задание из базы данных.

После выполнения любой операции метод возвращает строку с новым списком заданий, а именно строку из элементов `<li>`.

Совет

Всегда фильтруйте данные, получаемые от пользователя.

Если вы хотите уберечь себя от крупных неприятностей, то *всегда* должны фильтровать данные, получаемые от пользователя. Для этого в сценариях JavaScript мы вызываем функцию `encodeURIComponent`, прежде чем послать данные серверу. На стороне сервера мы вызываем метод `real_escape_string` объекта `mysqli`, чтобы предотвратить падение за счет внедренного кода SQL. Кроме того, на стороне сервера для подготовки текста к отправке клиенту мы вызываем функцию PHP `htmlentities`.

Подведение итогов

Вот и все! Мы получили работоспособное приложение управления списком заданий с поддержкой механизма перетаскивания, и для этого потребовалось написать совсем немного кода. На следующем этапе развития данного приложения можно было бы сделать задания доступными для редактирования по двойному щелчку мыши. Библиотека script.aculo.us предоставляет отличную возможность для этого с помощью `Ajax.InPlaceEditor`. За дополнительной информацией о том, как этого добиться, обращайтесь к документации, которую вы найдете по адресу *http://wiki.script.aculo.us/scriptaculous/show/Ajax.InPlaceEditor*.

Еще одно практическое применение перетаскивания к сортируемым спискам можно найти в **системе управления контентом (Content Management System – CMS)**, предназначенной для управления списками страниц, проектов, продуктов, новостей и т. д. В любом случае конечный результат зависит только от вашего воображения и от того, как далеко вы можете зайти по пути улучшения пользовательского интерфейса.

A

Подготовка рабочего окружения

Во избежание разного рода затруднений при изучении упражнений из этой книги с самого начала необходимо установить и правильно сконфигурировать необходимое ПО. Мы полагаем, что вы уже имеете некоторый опыт в разработке приложений на языке PHP, поэтому установку ПО рассмотрим очень коротко.

Хорошая новость: все необходимое ПО является бесплатным, эффективным и (что очень важно!) распространяется с инсталляторами, которые значительно облегчают установку и конфигурирование. Есть и плохая новость: варианты конфигураций многочисленны и разнообразны, поэтому приводимые ниже инструкции могут быть не на 100% применимы к вам (например, в MS Windows можно вместо веб-сервера Apache установить IIS и т. д.).

Мы дадим инструкции по установке отдельно для Windows и отдельно для UNIX-подобных ОС. Мы расскажем о том, как подготовить к работе базу данных, фигурирующую во многих примерах из этой книги. Эти инструкции одинаково применимы как к Windows, так и к UNIX, поэтому постарайтесь не пропустить этот раздел в конце приложения.

Для создания сайтов на основе технологии AJAX и PHP необходимо установить **PHP** предпочтительно версии 5, потому что некоторые из его характерных особенностей активно используются в этой книге. Также потребуется установить HTTP-сервер. Мы расскажем, как установить сервер **Apache**, который предпочитают большинство разработчиков PHP и компаний веб-хостинга. Поскольку мы постарались сделать примеры в этой книге настолько близкими к реальной жизни, насколько это было возможно, многие из них требуют наличия базы данных. Мы опишем процесс установки **MySQL** – самого популярного сервера баз данных в мире PHP. Учитывая, что код SQL, задействованный в примерах, чрезвычайно прост, его можно без особого труда адаптировать к другому серверу баз данных (к SQL или более ранней версии MySQL).

В конце приложения А мы дадим инструкции по установке приложения **phpMyAdmin** – очень удобного инструмента администрирования базы данных. Затем мы покажем, как с помощью этого инструмента создать новую базу данных и как предоставить пользователю полный доступ к этой базе данных.

Далее вы узнаете:

- Как установить Apache 2, PHP 5 и MySQL 5 под операционной системой Windows.
- Как установить Apache 2, PHP 5, и MySQL 5 под UNIX-подобной операционной системой.
- Как установить phpMyAdmin.
- Как создать новую базу данных и добавить пользователя с помощью phpMyAdmin.

Совет

К услугам тех, кто не хочет устанавливать ПО вручную, есть пакет программ XAMPP, содержащий все перечисленные выше (и еще ряд других) компоненты в виде отдельного файла инсталлятора. Пакет XAMPP распространяется в версиях для ОС Linux, Windows, Mac OS X и Solaris и совершенно бесплатен. XAMPP можно скачать по адресу *http://www.apachefriends.org/en/xampp.html*.

Приверженцы XAMPP могут сразу перейти в конец этого приложения к разделу, где описывается процесс установки базы данных ajax.

Подготовительные операции в Windows

Здесь мы расскажем о том, как установить в операционной системе Windows следующее ПО:

- Apache 2
- PHP 5
- MySQL 5

Установка Apache

Загрузить последнюю версию HTTP-сервера Apache в виде установочного пакета MSI Installer можно по адресу *http://httpd.apache.org/download.cgi*. Загрузите последнюю версию файла, его имя `apache_2.x.y-win32-x86-no_ssl.msi`, и запустите его на исполнение.

По умолчанию веб-сервер будет установлен в каталог `C:\Program Files\Apache Group\Apache2\`, но в процессе установки можно выбрать другой каталог (например, `C:\Apache2\`), если вы думаете, что это облегчит вам жизнь.

В процессе установки потребуется указать информацию о сервере (рис. А.1).

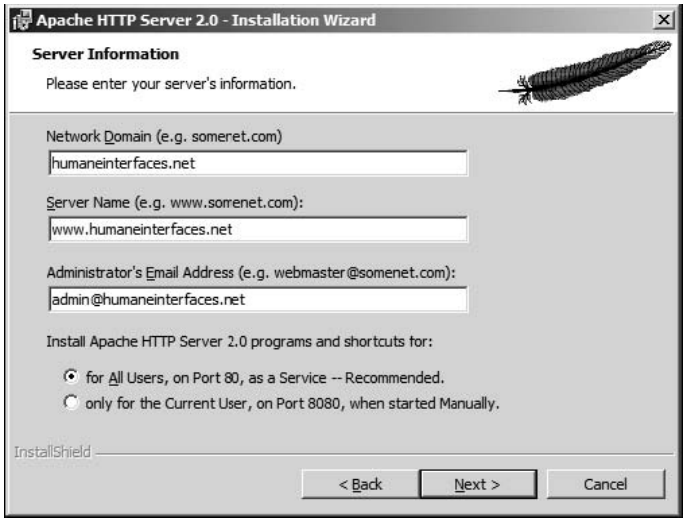

Рис. А.1. *Установка Apache 2.0*

В первые два поля можно ввести имя **localhost** (если точные значения неизвестны), а в последнем указать адрес электронной почты. Эту информацию всегда можно будет изменить, отредактировав конфигурационный файл сервера. По умолчанию это файл `C:\Program Files\Apache Group\Apache2\conf\httpd.conf`.

Вы также сможете выбрать порт, на котором будет работать сервер, – 80 или 8080. По умолчанию выбирается порт 80, но если уже установлен какой-либо другой HTTP-сервер (например, IIS), то для веб-сервера Apache потребуется определить отличающийся номер порта. Если вы предпочтете порт 8080, то службу Apache придется запустить вручную, для чего надо перейти в каталог с исполняемым файлом сервера (по умолчанию `C:\Program Files\Apache Group\Apache2\bin`) и дать команду

```
apache -k install
```

Если веб-сервер работает не с портом 80, то номер порта придется указывать в адресной строке броузера, например так: *http://localhost:8080/ajax/suggest*.

В следующем экране программы-инсталлятора можно оставить значения предлагаемых настроек по умолчанию.

Вместе с сервером Apache инсталлятор также запустит программу **Apache Service Monitor**, которая появится на панели задач. Значок на панели задач будет отражать текущее состояние веб-сервера (остановлен, запущен и т. д.), а также позволит запускать, останавливать или перезапускать службу Apache.

Примечание

Запомните: вам необходимо перезапускать (или останавливать, а затем запускать) службу Apache каждый раз, когда вы будете вносить любые изменения в конфигурационный файл `httpd.conf`, для того, чтобы эти изменения вступили в силу.

После установки Apache2 необходимо проверить его работоспособность. Если ему назначен порт 80, откройте страницу *http://localhost/*, а если порт 8080, то страницу *http://localhost:8080/*. В окне броузера вы должны увидеть примерно такую начальную страницу Apache, как на рис. А.2.

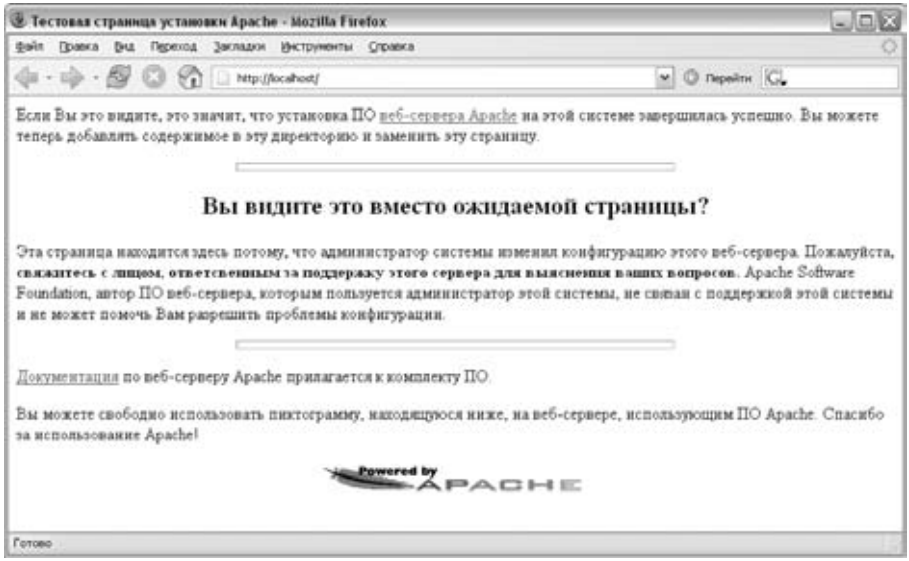

Рис. А.2. *Установка Apache прошла успешно*

Установка MySQL

Официальный сайт MySQL расположен по адресу *http://www.mysql.com*. На момент написания этих строк последней стабильной версией MySQL была 5.0, которую можно загрузить со страницы *http://dev.mysql.com/downloads/mysql/5.0.html*. Однако вы должны знать, что текст наших запросов SQL написан в соответствии со стандартом SQL 92, поэтому он будет работать с другими современными СУБД данных практически без переделок.

На странице `Downloads` дойдите до раздела `Windows downloads` и скачайте файл `Windows Essentials`. Вам будет предложено выбрать сайт зеркала, а имя файла будет иметь следующий формат: `mysql-essential-5.0.xx-win32.msi`. Скачав этот файл, запустите его на исполнение.

После установки вам будет предоставлена возможность сконфигурировать сервер баз данных. Сделайте это. Во время конфигурирования вполне допустимо принять значения всех параметров, предложенные по умолчанию. В какой-то момент вам будет предложено ввести пароль для суперпользователя (пользователь с именем root). Выберите такой пароль, чтобы вы могли его легко запомнить, но чтобы другим сложно было подобрать его.[1]

Прежде чем перейти к опробованию примеров из этой книги, не забудьте заглянуть в раздел «Подготовка базы данных ajax» в конце этого приложения.

Установка PHP

Официальный сайт PHP располагается по адресу *http://www.php.net*. Скачайте из раздела `Windows Binaries` последнюю версию PHP 5 в виде архива zip (это не программа установки!) со страницы *http://www.php.net/downloads.php*. Мы не советуем использовать инсталлятор, поскольку в его состав не входят расширения PHP, которые могут потребоваться для реализации ваших проектов, а кроме того, он практически не выполняет действий по конфигурированию.

Скачав архивный файл с PHP 5, выполните следующие действия:

1. Распакуйте архив (это должен быть файл с именем в формате `php-5.x.y-win32.zip`) в каталог `C:\PHP\`. Но можно выбрать и любой другой каталог.
2. Скопируйте файл `php.ini-recommended` из каталога `C:\PHP\` в каталог Windows (`C:\Windows`) и переименуйте его в `php.ini`.
3. Откройте файл `php.ini` в любом текстовом редакторе (например, в Noptepad), раскомментируйте строки, содержащие ссылки на расширения `php_gd2.dll`, `php_mysql.dll` и `php_xsl.dll` (удалив символ точки с запятой в первой позиции строки), и добавьте строку с именем расширения для `php_mysqli`:

```
extension=php_gd2.dll
extension=php_mysql.dll
extension=php_mysqli.dll
extension=php_xsl.dll
```

[1] Традиция суперпользователя root зародилась в мире UNIX – этот и только этот пользователь имеет привилегии на выполнение любых действий без ограничений. Ввиду этого пароли в таких системах достаточно сильно защищены (например, зашифрованы), а это значит, что если вы забудете пароль root, то уже не будет никакого способа его восстановить, и вам скорее всего придется полностью переустанавливать систему или подсистему (часто с невосстановимой потерей пользовательских данных). Резюме: пароли, и особенно пароли для root, лучше не запоминать, а **записывать** где-то в укромном месте. – *Примеч. науч. ред.*

4. Рекомендуем разрешить вывод подробных сообщений об ошибках в PHP на машине, где ведется разработка, но это совершенно не обязательно (вывод подробных сообщений об ошибках включен по умолчанию). Будьте внимательны: это изменение может повлиять на работу других сценариев на вашем сервере. Найдите строку `error_reporting` в файле `php.ini` и измените ее на:

```
error_reporting = E_ALL
```

5. Скопируйте файлы `php5ts.dll` и `libmysql.dll` из каталога `C:\PHP\` в каталог системных библиотек `System32` ОС Windows (по умолчанию это `C:\Windows\System32`).

6. Скопируйте файлы `php_mysql.dll`, `php_mysqli.dll`, `php_xsl.dll` и `php_gd2.dll` из каталога `C:\PHP\ext` в тот же каталог `System32` операционной системы Windows.

7. Откройте конфигурационный файл Apache в любом текстовом редакторе. По умолчанию этот файл носит имя `C:\Program Files\Apache Group\Apache2\conf\httpd.conf`.

8. Найдите в файле `httpd.conf` раздел с множеством строк, которые начинаются с `LoadModule`, и добавьте туда следующие строки:

```
LoadModule php5_module c:/php/php5apache2.dll
AddType application/x-httpd-php .php
```

9. Найдите в файле `httpd.conf` запись `DirectoryIndex` и добавьте в конец строки `index.php`, чтобы она выглядела примерно так:

```
DirectoryIndex index.html index.html.var index.php
```

10. Сохраните файл `httpd.conf` и перезапустите службу Apache 2 с помощью Apache Service Monitor, щелкнув по значку, расположенному в системном трее на панели задач. (Получив при этом какие-либо сообщения об ошибках, проверьте еще раз, насколько точно были выполнены вышеуказанные инструкции.) Если Apache перезапустится без каких-либо сообщений, то это добрый знак.

11. Создайте каталог `ajax` в каталоге `htdocs` (по умолчанию `C:\Program Files\Apache Group\Apache2\htdocs`).

12. Убедитесь, что ваш Apache корректно разбирает код PHP. Для этого создайте в каталоге `ajax` файл с именем `test.php` и добавьте в него следующий код:

```
<?php
phpinfo();
?>
```

13. Откройте в броузере страницу *http://localhost/ajax/test.php* (или *http://localhost:8080/ajax/test.php*, если веб-серверу назначен порт 8080), если все было установлено без ошибок, то страница будет выглядеть примерно так, как на рис. А.3.

Поздравляем, вы только что установили Apache, MySQL и PHP!

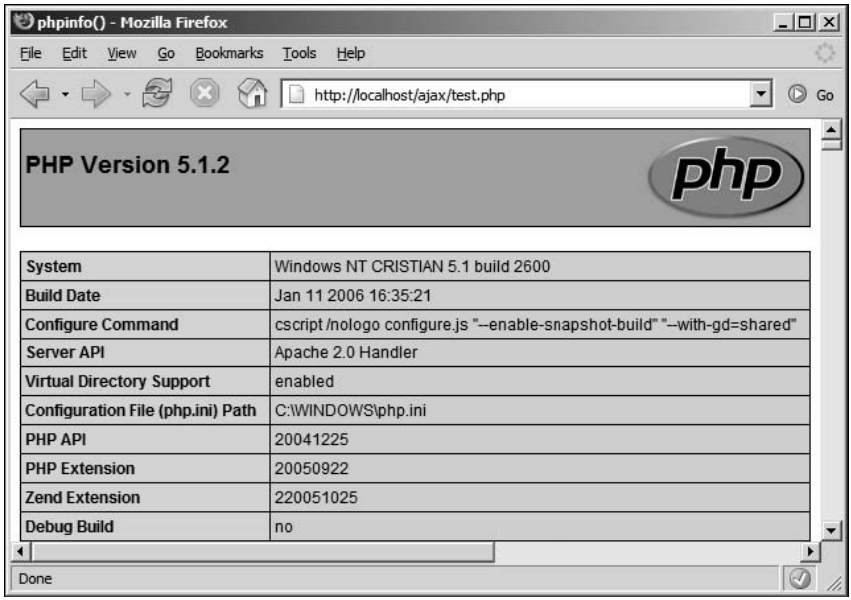

Рис. А.3. *PHP был успешно установлен*

Но процесс конфигурирования на этом не заканчивается. Если вы работаете с Windows (а это скорее всего именно так, раз вы читаете этот раздел), пропустите следующий раздел «Подготовительные операции в UNIX-подобных ОС» и перейдите к разделам «Установка phpMyAdmin» и «Подготовка базы данных ajax» в конце этого приложения.

Подготовительные операции в UNIX-подобных ОС

Практически все версии UNIX и дистрибутивы Linux поставляются в комплекте с Apache, PHP и MySQL, однако вам необходимо проверить версии этих программ. Будет неплохо, если у вас имеется MySQL 4.1 или выше, и очень важно, чтобы PHP имел версию не ниже 5.0. Код сценариев PHP, приводимых в этой книге, просто не будет работать с более ранними версиями PHP.

Установка Apache

Для того чтобы установить сервер Apache в UNIX-подобной ОС, выполните два следующих действия:

1. Скачайте последнюю версию файла Apache Unix Source (в исходных кодах) для своей системы со страницы *http://httpd.apache.org/download.cgi* и распакуйте его примерно такой командой:

```
tar -zxvf httpd-2.0.55.tar.gz
```

2. Чтобы скомпилировать и установить Apache Web Server, перейдите в каталог, содержащий полученные исходные тексты, и выполните следующие команды, не забыв предварительно зарегистрироваться в системе под пользователем root:

```
./configure --prefix=/usr/local/apache2 --enable-so --enable-ssl --withssl
--enable-auth-digest

make

make install
```

Установка MySQL

Адрес официального сайта MySQL: *http://www.mysql.com*. На момент написания этих строк последней стабильной версией была MySQL 5.0, которую можно скачать со страницы *http://dev.mysql.com/downloads/mysql/5.0.html*. Однако текст наших запросов SQL соответствует стандарту SQL 92, поэтому они будут работать с другими СУБД (или с более ранними версиями MySQL) практически без переделок. Инструкции по установке для всех поддерживаемых платформ приведены в главе 2 руководства по MySQL 5 (*http://dev.mysql.com/doc/refman/5.0/en/installing.html*).

Если у вас установлен дистрибутив Linux, поддерживающий формат RPM, придется скачать RPM-пакеты «Server», «Client» и «Headers and Libraries». Процесс установки описан в руководстве, расположенном по адресу *http://dev.mysql.com/doc/refman/5.0/en/linux-rpm.html*. Если ваша платформа не поддерживает работу с пакетами RPM, то описание процесса установки MySQL вы найдете по адресу *http://dev.mysql.com/doc/refman/5.0/en/installing-binary.html*.

Установив MySQL, измените пароль администратора (пользователь `root@localhost`), который по умолчанию не задан. Подробнее о паролях в MySQL рассказано по адресу *http://dev.mysql.com/doc/mysql/en/Passwords.html*. Один из способов изменить пароль администратора состоит в том, чтобы выполнить команду:

```
mysqladmin -u root password 'ваш_новый_пароль'
```

Другой способ заключается в том, чтобы получить доступ к MySQL с помощью консольного приложения или программы администрирования, такой как phpMyAdmin, и выполнить команду:

```
SET PASSWORD FOR root@localhost=PASSWORD('your_new_password');
```

Теперь можно проверить сервер MySQL, выполнив в консоли команду:

```
# mysql -u root -p
```

Установка PHP

Всякий раз, когда вы захотите доустановить под Linux новую библиотеку PHP, вам придется пересобирать модуль PHP (из исходных ко-

дов). Именно поэтому мы рекомендуем сразу же собрать PHP вместе со всеми необходимыми библиотеками расширения.

1. Откройте страницу *http://www.php.net/downloads.php*, скачайте архив с полными исходными текстами PHP 5.x и распакуйте его в каталог (той же командой `tar`, формат которой был показан ранее). На момент написания этих строк последняя версия PHP была 5.1.2.
2. Перейдите в каталог с исходными текстами и выполните следующие команды:

```
./configure --with-config-file-path=/etc --with-mysql=/usr/include/mysql
--with-apxs2=/usr/local/apache2/bin/apxs --with-zlib --with-gd --with-xsl

make

make install
```

Примечание

Если вы собираете PHP для XAMPP, вы должны использовать следующую команду `configure`:

```
./configure --with-config-file-path=/opt/lampp/etc --with-mysql=/opt/lampp
--with-apxs2=/opt/lampp/bin/apxs --with-zlib --with-gd
```

После исполнения команд `make` и `make install` вы должны скопировать файл `php_src/libs/libphp5.so` в каталог `/opt/lamp/modules/`.

3. Скопируйте файл `php.ini-recommended` в каталог `/etc`, выполнив следующую команду:

```
cp php.ini-recommended /etc/php.ini
```

4. Откройте конфигурационный файл Apache (`httpd.conf`), найдите в нем запись `DirectoryIndex` и убедитесь, что она содержит имя `index.php`:

```
DirectoryIndex index.html index.html.var index.php
```

5. Перезапустите веб-сервер Apache с помощью следующей команды:

```
/usr/local/apache2/bin/apachectl restart
```

6. Создайте каталог `ajax` в каталоге `htdocs` (по умолчанию: `/usr/local/apache2/htdocs`).
7. Убедитесь в том, что установка PHP произведена корректно, для чего создайте в каталоге `ajax` файл с именем `test.php` и добавьте в него следующий код:

```
<?php
phpinfo();
?>
```

8. И наконец, откройте в броузере страницу *http://localhost/ajax/test.php*. Если установка PHP была произведена без ошибок, страница будет выглядеть примерно так, как на рис. А.3.

Установка phpMyAdmin

Программа phpMyAdmin – это очень популярный инструмент администрирования MySQL, написанный на PHP. Она позволяет управлять базами данных MySQL через простой и удобный веб-интерфейс. Официальная веб-страница программы: *http://www.phpmyadmin.net*. Чтобы установить и сконфигурировать программу, выполните следующие действия:

1. Загрузите последнюю версию phpMyAdmin с адреса *http://www.phpmyadmin.net/home_page/downloads.php*. Если вы не уверены, какой файл скачивать, берите zip-архив.
2. Распакуйте архив на жесткий диск. В архиве содержится каталог, имя которого включает полный номер версии phpMyAdmin (на момент написания этих строк это был каталог с бета-версией phpMyAdmin, с именем `phpMyAdmin-2.8.0-beta1`).
3. Чтобы облегчить себе жизнь, переименуйте этот каталог в `phpMyAdmin`.
4. Переместите каталог `phpMyAdmin` в каталог `htdocs` веб-сервера Apache 2 (по умолчанию `C:\Program Files\Apache Group\Apache2\htdocs`).
5. Убедитесь в том, что программа phpMyAdmin доступна для Apache, для чего откройте в броузере страницу *http://localhost/phpMyAdmin*. Если все работает нормально, вы должны получить примерно такое сообщение (рис. А.4).

Рис. А.4. *Программа phpMyAdmin не имеет доступа к MySQL*

6. Сообщение об ошибке сразу наводит на мысль: необходимо показать phpMyAdmin, как получить доступ серверу MySQL. В каталоге

phpMyAdmin найдите файл с именем config.inc.php и измените параметры в нем так, как показано ниже. Если файл найти не удалось, создайте его и добавьте туда следующий код:

```
<?php
$cfg['PmaAbsoluteUri'] = "http://localhost/phpMyAdmin/";
$cfg['Servers'][1]['host'] = "localhost";
$cfg['Servers'][1]['auth_type'] = 'config';
$cfg['Servers'][1]['user'] = "root";
$cfg['Servers'][1]['password'] = "password";
?>
```

Примечание

За дополнительной информацией о порядке установки и пользовании программой phpMyAdmin обращайтесь к документации, расположенной по адресу *http://www.phpmyadmin.net/home_page/docs.php*. Издательством Packt Publishing была выпущена отдельная книга специально для тех, кто хочет побольше узнать о phpMyAdmin, – «Mastering phpMyAdmin for Effective MySQL Management» (ISBN: 1-904811-3-5). На тот случай, если вы не владеете английским языком, можем вас порадовать – она переведена на чешский, немецкий, французский и итальянский языки.

Подготовка базы данных ajax

В качестве упражнения работы с phpMyAdmin и MySQL создадим базу данных с именем ajax и добавим в нее пользователя с неограниченными привилегиями. Эта база данных и пользователь фигурируют во всех примерах в этой книге. Для этого выполните следующие действия:

1. Откройте в броузере страницу *http://localhost/phpMyAdmin*. Если информация, внесенная вами в файл config.inc.php, корректна, вы должны увидеть нечто подобное тому, что изображено на рис. А.5.
2. В поле Create new database введите имя ajax и щелкните по кнопке Create.
3. Программа phpMyAdmin не предусматривает возможности добавления пользователей с помощью визуального интерфейса, поэтому теперь вам придется написать SQL-запрос. Вам предстоит создать в базе данных ajax пользователя с неограниченными правами. Этот пользователь будет называться (сюрприз!) ajaxuser, и мы назначим ему пароль practical. Чтобы добавить этого пользователя, щелкните по вкладке SQL в верхней части страницы и введите следующий код:

```
GRANT ALL PRIVILEGES ON ajax.*
TO ajaxuser@localhost IDENTIFIED BY "practical"
```

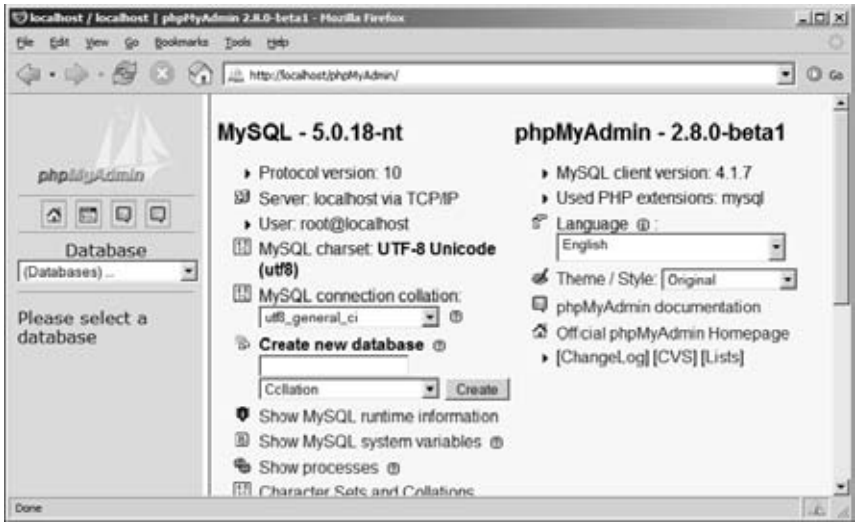

Рис. А.5. *phpMyAdmin в действии*

Примечание

Текст SQL-запроса очень напоминает обычное предложение на английском языке, нам остается прояснить лишь несколько моментов. Символ «*» в `ajax.*` означает: *все объекты в базе данных ajax*. Таким образом, эта команда говорит MySQL: «выдать все возможные привилегии на работу с базой данных `ajax` пользователю `ajaxuser`, работающему на этой машине и идентифицируемому паролем `practical`».

4. Щелкните по кнопке Go.

Поздравляем, вы установили все необходимое для путешествия по этой книге. Желаем вам благополучного освоения технологии AJAX!

Алфавитный указатель

A

abort(), метод объекта XMLHttpRequest, 65
addPoint(), метод, realTimeChart.js, 251
addToCache, функция, подсказки и автодополнение средствами AJAX, 231
AJAX, 24
  JavaScript и DOM, 44
  автодополнение, 207
  безопасность в многопоточной среде, 158
  механизм drag-and-drop, 293
  подготовка базы данных, 324
  подсказки, 207
  построение диаграмм SVG в реальном времени, 237
  потенциальные проблемы, 28
  преимущества, 27
  программа чтения лент новостей в формате RSS, 279
  простое приложение quickstart, 30
  таблицы данных, 252
  чат, 183
ajaxRequest(), метод, ajaxRequest.js, 250
ALTER, оператор SQL, MySQL, 132
ASP.NET, серверная технология, 21
async, аргумент метода open(), 66
Atom, формат агрегирования информации, 276
autocompleteKeyword, функция, подсказки и автодополнение средствами AJAX, 233

C

checkCache, функция, подсказки и автодополнение средствами AJAX, 231
checkForUpdates(), функция, подсказки и автодополнение средствами AJAX, 231
checkUsername(), функция, чат AJAX, 204
clearInterval(), функция, 118
clearTimeout(), функция, 118
createPointInfo(), метод, realTimeChart.js, 251
createTextRange(), метод, объект TextRange, 234
createXmlHttpRequestObject(), функция, 63, 60
  quickstart.js, 38
CRssReader, класс, rss_reader.class.php, 290

D

DELETE, команда MySQL, 134
deleteMessages, функция, чат AJAX, 204, 205
deselectAll(), функция, подсказки и автодополнение средствами AJAX, 232
displayError(), функция, чат AJAX, 204
display_errors, параметр php.ini, 101
displayMessages(), функция, чат AJAX, 204
displayResults(), функция, подсказки и автодополнение средствами AJAX, 232
, элемент, 49
DML (Data Manipulation Language), язык манипулирования данными, 134
DOM (Document Object Model), объектная модель документа, 26, 44
  интерфейс, 44
  использование на стороне клиента и на стороне сервера, 45

функции в PHP, 87
document.write, команда, 46
DOMDocument, класс PHP DOM, 93
DOMParser, класс, 272
drag-and-drop
покупательская корзина, 293
приложение обслуживания сортируемых списков, 295
сортируемые списки, 293
DROP, оператор SQL, MySQL, 132

E

editId(), функция, 274
Emacs, редактор, 207
encode(), функция, подсказки и автодополнение средствами AJAX, 232
encodeURIComponent, функция(), JavaScript, 178

F

fetch_array(), метод, MySQL, 140
FIFO, очередь сообщений, 159
file_get_contents(), функция, 111, 117, 128, 129
friendly, приложение AJAX, 142

G

GD, библиотека, 186
getAllResponseHeaders(), метод объекта XMLHttpRequest, 65
getColor(), функция, чат AJAX, 203
getFormattedXML(), функция, 291
getMouseXY(), функция, 203
getResponseHeader(), метод объекта XMLHttpRequest, 65
Gmail, почтовая служба, 26
GNU Emacs, редактор, 207
Google Maps, веб-приложение, 26
Google Reader, веб-приложение, 278
Google Suggest, подсказки Google, 25, 208

H

handleCheckingAvailability(), функция, 127
handleGettingColor(), функция, чат AJAX, 203
handleGettingNumber(), функция, 127
handleGettingResults(), метод, ajaxRequest.js, 250
handleGettingSuggestions(), функция, подсказки и автодополнение средствами AJAX, 231
handleGridPageLoad(), функция, 273
handleKey(), функция, чат AJAX, 204
handleKeyUp(), функция, подсказки и автодополнение средствами AJAX, 231, 232
handleOnMouseOut(), функция, подсказки, автодополнение и AJAX, 232
handleOnMouseOver(), функция, подсказки, автодополнение и AJAX, 232
handleReceivingMessages(), функция, чат AJAX, 204
handleRequestStateChange(), метод, 68, 73
handleRequestStateChange(), функция, 71
handleResults(), метод, realTimeChart.js, 251
handleServerResponse(), функция, books.js, 79
handleServerResponse(), функция, quickstart.js, 38
hideSuggections(), функция, подсказки, автодополнение и AJAX, 232
HTML (HyperText Markup Language), язык разметки гипертекста, 19
HTTP (HyperText Transfer Protocol), протокол передачи гипертекстовой информации, 19
HTTPS (HyperText Transmission Protocol, Secure), безопасная версия протокола HTTP, 20

I

imagecolorat, функция, PHP, 203
imagecolorat, функция, чат AJAX, 205
imagecreatefrompng(), функция, PHP, 203
imagecreatefrompng(), функция, чат AJAX, 205
index.html, простое приложение quickstart, 32
init(), метод, realTimeChart.js, 250
init, функция, подсказки, автодополнение и AJAX, 231
init, функция, чат AJAX, 204

innerHTML, свойство DOM, 52
INSERT, команда, MySQL, 134
IntelliSense, Microsoft Visual Studio, 207
isDatabaseCleared, функция, чат AJAX, 205

J

JavaScript, язык программирования, 22, 44
  DOM и, 46
  безопасность, 104
  использование на стороне клиента, 44
  код в отдельном файле, 46
  события, 48
  функции выполнения повторяющихся действий, 118
JSON (JavaScript Object Notation), представление объектов в JavaScript, 27
JVM (Java Virtual Machine), виртуальная машина Java, 23

L

loadGridPage(), функция, 273
loadStylesheet(), функция, 272

M

Meebo, решение обмена мгновенными сообщениями на базе AJAX, 184
move(), метод, объект TextRange, 234
moveEnd(), метод, объект TextRange, 234
moveStart(), метод, объект TextRange, 234
MySQL
  атрибут NOT NULL, 131
  действия с данными, 134
  индексы, 132
  команды DML, 134
  система управления базами данных, 129
  соединение с базой данных, 135
  создание таблиц, 129

N

NOT NULL, атрибут, MySQL, 131

O

ob_clean(), функция, обработка ошибок, 101
onblur, событие, JavaScript, 176
onclick, событие , JavaScript, 230
onkeyup, событие, JavaScript, 230
onload, событие, 51
onreadystatechange, метод объекта XMLHttpRequest, 65
open(), метод объекта XMLHttpRequest, 65

P

PHP, язык сценариев, 21, 86
  DOM и, 86
  $_SESSION, массив переменных сессии, 176
  безопасность JavaScript, 104
  доверенный сценарий, 110
  использование на стороне сервера, 44
  обработка ошибок, 93
  объектно-ориентированное программирование, 141
  передача параметров, 93
  работа с MySQL, 129
  соединение с базой данных, 135
  удаленные серверы, 104
phpMyAdmin, приложение, 132
postMessages, функция, чат AJAX, 205
private, частные члены класса, 142
process(), функция, quickstart.js, 35, 38
protected, защищенные члены класса, 142
Prototype, интегрированная система, JavaScript, 295
public, общедоступные члены класса, 141
push(), метод, класс Array, JavaScript, 177

Q

quickstart.js, простое приложение quickstart, 33
quickstart.php, простое приложение quickstart, 34

R

readMessages(), функция, чат AJAX, 204
readyState(), метод объекта XMLHttpRequest, 65
refreshXYIndexes(), метод, realTimeChart.js, 251
removePointInfo(), метод, realTimeChart.js, 251
responseText(), метод объекта XMLHttpRequest, 65
responseXML(), метод объекта XMLHttpRequest, 65
retrieveNewMessages(), функция, чат AJAX, 205
RSS
  Google Reader, веб-приложение, 278
  описание, 276
  программа чтения лент новостей, 279
  структура документа, 277

S

save(), функция, 274
saveXML(), функция, 93
script.aculo.us, библиотека, 293, 295
SELECT, команда, MySQL, 134
select(), метод объекта TextRange, 234
selectRange(), функция, подсказки, автодополнение и AJAX, 233
send(), метод объекта XMLHttpRequest, 65
$_SESSION, массив переменных сессии, PHP, 176
session_start(), функция, 176
setFocus(), функция, JavaScript, 176
setInterval, функция, 118
setRequestHeader(), метод объекта XMLHttpRequest, 65
setSelectionRange, функция JavaScript, 234
SetStyle(), метод, 58
setTimeout(), функция, 118
shift, класс Array, JavaScript, 177
SimpleXML, прикладной программный интерфейс, 93
startup(), функция, приложение управления заданиями, 308
ststuaText(), метод объекта XMLHttpRequest, 65
SVG (Scalable Vector Graphics), масштабируемая векторная графика, 236

T

TextRange, объект, IE, 234
TRUNCATE, оператор SQL, MySQL, 132
try/catch, конструкция, 60

U

, элемент, 49
undo(), функция, 274
UPDATE, команда, MySQL, 134
updateChart(), метод, realTimeChart.js, 251
updateKeywordValue(), функция, подсказки, автодополнение и AJAX, 232
updateRow(), функция, 274
updateSuggestions(), функция, подсказки, автодополнение и AJAX, 231

V

validate(), функция, JavaScript, 176

W

W3C, консорциум World Wide Web, 236
Windows Live Local, 26

X

XML (eXtensible Markup Language), расширяемый язык разметки
  создание XML-документов в PHP, 87
    phptest.html, 87
    phptest.js, 88
    phptest.php, 90
XMLHttpRequest, класс, 272
XMLHttpRequest, объект, 26, 59
  XML и, 76
  асинхронные вызовы, 67
  занятость, исключение, 102
  инициализация запросов к серверу, 65
  методы и свойства, 65
  создание, 60
xmlHttp.send(), функция, 68
xmlToArray(), функция, подсказки, автодополнение и AJAX, 232
XPath, язык адресации частей XML-документа, 252

XSLTProcessor, класс, 272
XSL (eXstensible Stylesheet Language), расширяемый язык таблицы стилей, 252
XSL-преобразования, 252
XSLT (eXtensible Stylesheet Language for Transformations), расширяемый язык таблицы стилей для преобразований , 252

Y

Yahoo Maps, веб-приложение, 26
Yahoo! Instant Search, веб-приложение, 25

А

автодополнения, функция, 207
авторизация, 135
агрегатор, 278
асинхронные вызовы
  async.html, 69
  async.js, 70
  async.txt, 69
  повторяющиеся, 118
  с помощью XMLHttpRequest и XML, 76
    books.html, 76
    books.js, 77
    books.xml, 76

Б

безопасность сценариев JavaScript, 104

В

веб-клиент, 20
веб-приложения, 16
веб-сервер, 20
верификация входных данных, 154
верификация, ненавязчивая, 175
верификация заполнения формы с помощью AJAX, 155
  allok.php, 165
  config.php, 169
  error_handler.php, 169
  index.php, 162, 176
  index_top.php, 161, 176
  validate.class.php, 170, 175, 181
  validate.css, 160
  validate.js, 165, 176
  validate.php, 169, 175
виртуальная машина Java, 23
возможность определения объекта, 62

Д

деструктор, 141
действия с данными, MySQL, 134
диаграммы в реальном времени средствами SVG и AJAX, 237
доверенный сценарий PHP, 110
доверенный сценарий на стороне сервера, 111
  proxyping.html, 111
  proxyping.js, 112
  proxyping.php, 114
доступ к данным, удаленный сервер, 111

З

заполнитель, 48
зоны, модель безопасности, Internet Explorer, 104

И

индексы, MySQL, 132
инициализация запросов к серверу, 65

К

класс, объектно-ориентированное программирование, 141
клиент IRC, AJAX, 184
клиентские технологии, 21
консоль JavaScript в Firefox, 81
конструкция try/catch, 60

М

масштабируемая векторная графика (SVG), 236
модель безопасности в броузере, 104

О

обработка исключительных ситуаций в JavaScript, 60
обработка ошибок и возбуждение исключений, 81
  Firefox, 81
  Internet Explorer, 82

Mozilla, 81
Opera, 82
PHP, 93
конструкция try/catch, 81
общедоступный интерфейс, 141
объектная модель документа, *см.* DOM
обычные приложения, 16
ООП, объектно-ориентированное программирование, 141
основные принципы выполнения повторяющихся асинхронных запросов, 118
ответа сервера, асинхронные запросы, 68
очередь, 159
очередь сообщений, AJAX, безопасность в многопоточной среде, 159

П

первичный ключ, MySQL, 131
передача параметров и обработка ошибок в PHP, 93, 94
error_handler.php, 98, 99, 101
повторяющиеся асинхронные запросы, 118
подготовка базы данных, AJAX, 324
подсказки Google, 25
подсказки, автодополнение и AJAX, 207
config.php, 211
error_handler.php, 211
index.html, 213, 228
suggest.class.php, 212
suggest.css, 214
suggest.js, 216, 230
suggest.php, 212
построение диаграмм SVG в реальном времени
ajaxRequest.js, 241, 249, 250
chart.svg, 241, 249
index.html, 241, 249
realTimeChart.js, 243, 250
svg_chart.php, 248
представление объектов в JavaScript, 27
приложение AJAX friendly
config.php, 143
error_handler.php, 143
friendly.class.php, 142
friendly.css, 142
friendly.js, 142
friendly.php, 142
index.html, 142
приложение JavaScript и DOM,
jsdom.html, 47
jsdom.js, 47
приложение управления заданиями, 297
config.php, 298
drag-and-drop.css, 306
drag-and-drop.js, 302
drag-and-drop.php, 301, 312
error_handler.php, 298
index.php, 298, 308
taskslist.class.php, 299, 312
программа чтения лент RSS
config.php, 282
error_handler.php, 281
index.php, 282, 289
rss_reader.class.php, 280
rss_reader.css, 286
rss_reader.js, 283, 290
rss_reader.php, 280, 290
rss_reader.xsl, 281, 291
простое веб-приложение quickstart, 30
простое приложение AJAX, 29

Р

работа со структурой XML, 75
расширяемый язык стилей, *см.* XSL
расширяемый язык стилей для преобразований, *см.* XSLT
реализация алгоритма выполнения повторяющихся действий, 119
smartproxyping.html, 119
smartproxyping.js, 120
smartproxyping.php, 125
реализация чата на основе технологии AJAX, 185

С

сервер баз данных, 22
серверные технологии, 21
синтаксический разбор файла XML, 278
система безопасности базы данных, 135
события в JavaScript, 48
и DOM, 48
morejsdom.html, 50
morejsdom.js, 50
соединение с базой данных, 135

соединение с удаленным сервером, 104, 105
создание структур XML, 84
создание таблиц, MySQL, 130
сортируемые списки, drag-and-drop, 295
стандартизация, 18
строковые значения, 52
структура документа RSS, 277
СУРБД, системы управления реляционными базами данных, 129, 130

Т

таблица базы данных, создание, 130
таблицы данных
    config.php, 259
    error_handler.php, 260
    grid.class.php, 256
    grid.css, 269
    grid.js, 262, 272
    grid.php, 255, 270
    grid.xsl, 260, 272
    index.html, 260
тег script, внедрение кода JavaScript, 45
тип данных, MySQL, 131
толстый клиент, 17
тонкий клиент, 17

У

удаленный сервер
    ping.html, 105
    ping.js, 106
    соединение, 105
        Firefox, 108
        Internet Explorer, 108
        Opera, 108
упражнение CSS и JavaScript, 56
    csstest.html, 56
    csstest.js, 56
    styles.css, 57
упражнение «и еще о DOM», 53
    evenmorejsdom.html, 53
    evenmorejsdom.js, 54
установка Apache
    UNIX, 321
    Windows, 315
установка MySQL
    UNIX, 321
    Windows, 317
установка PHP
    UNIX, 322
    Windows, 318
установка phpMyAdmin, 323

Ц

централизованное распространение информации в веб, 276

Ч

чат на основе технологии AJAX, 183
    chat.class.php, 189, 205
    chat.css, 192
    chat.js, 194, 203
    chat.php, 188, 205
    config.php, 187
    error_handler.php, 187
    get_color.php, 192
    index.html, 193
    palette.png, 187
    выбор цвета, 203
    область прокрутки, 203
члены класса, 141
чтение данных, удаленный сервер, 111

Ю

юзабилити, 16

Я

язык адресации частей документа XML (XPath), 252
язык разметки гипертекста (HTML), 19