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

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.

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).

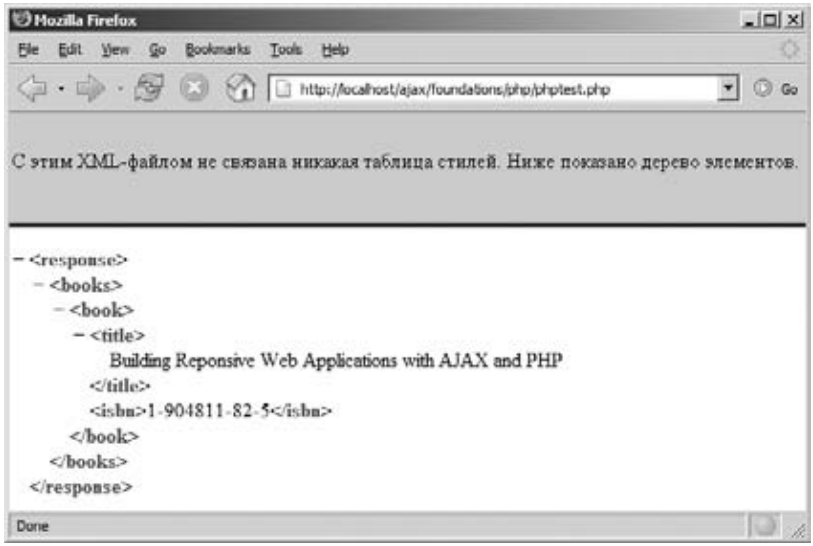

***Рис. 3.1.** Простой документ XML, сгенерированный PHP*

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).

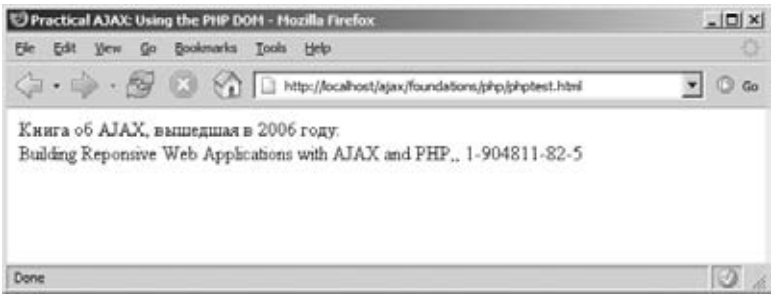

***Рис. 3.2.** AJAX и PHP*

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.

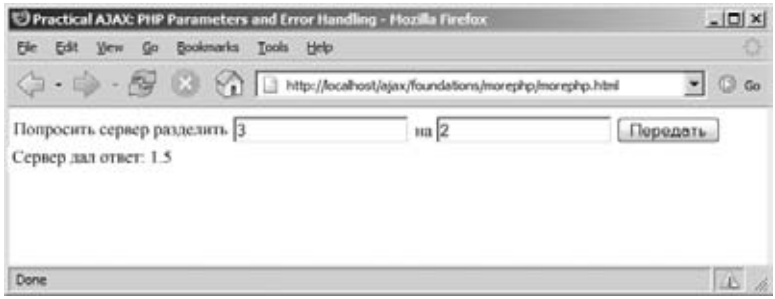

Рис. 3.3. *Передача параметров и обработка ошибок в PHP*

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):

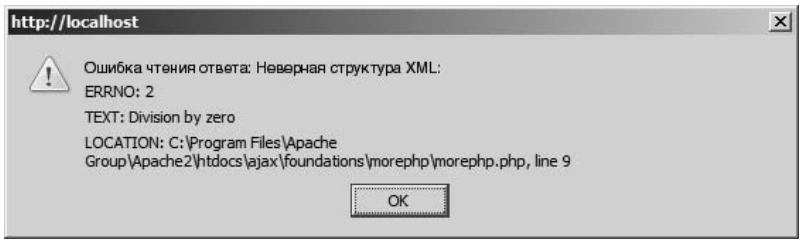

***Рис. 3.4.** Удобочитаемое сообщение об ошибке*

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.*

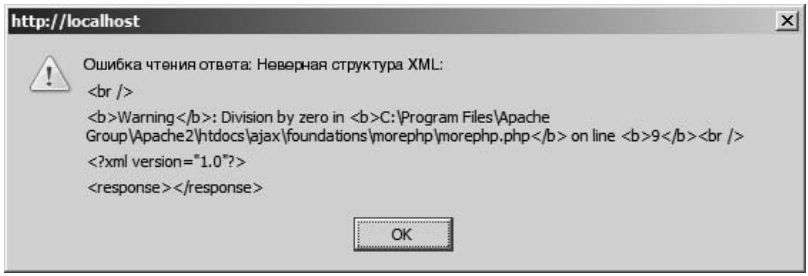

***Рис. 3.5.** Неудобочитаемое сообщение об ошибке*

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).

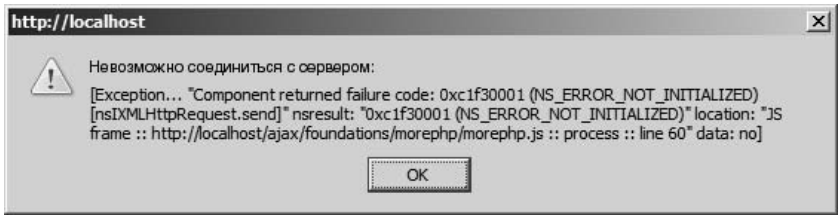

Рис. 3.6. *Попытка выполнения запроса в то время, когда XMLHttpRequest занят обработкой предыдущего запроса*

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",
```

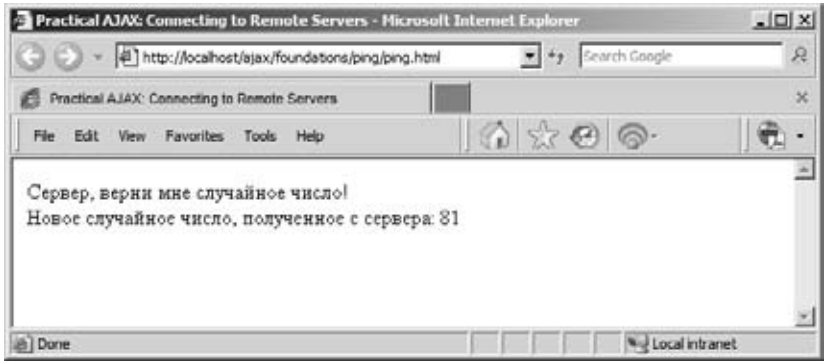

Рис. 3.7. *Соединение с удаленным сервером*

```
                                        "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

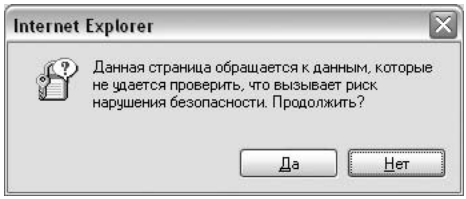

Рис. 3.8. *Internet Explorer запрашивает разрешение на соединение с удаленным сервером*

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

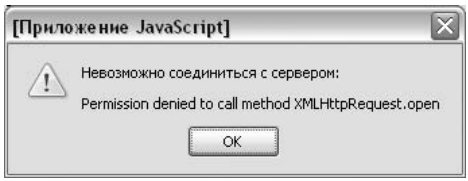

Рис. 3.9. *Firefox закрывает доступ*

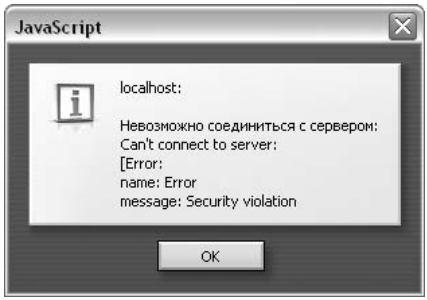

Рис. 3.10. *Opera закрывает доступ*

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

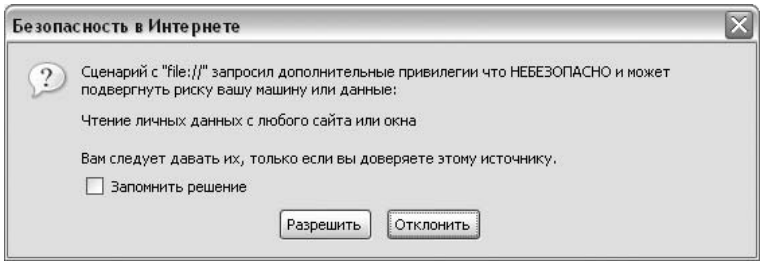

Рис. 3.11. *Firefox запрашивает разрешение на доступ к удаленному серверу*

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.

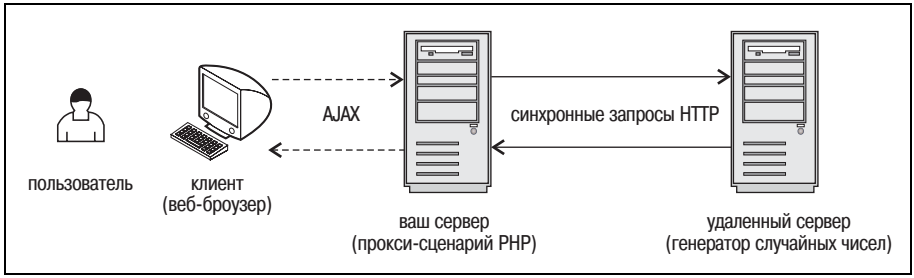

Рис. 3.12. *Использование доверенного сценария PHP для доступа к удаленному серверу*

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).

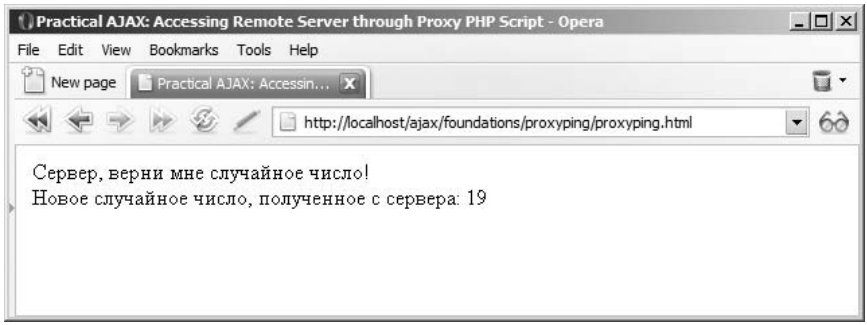

Рис. 3.13. *Использование доверенного сценария PHP для доступа к удаленному серверу*

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.

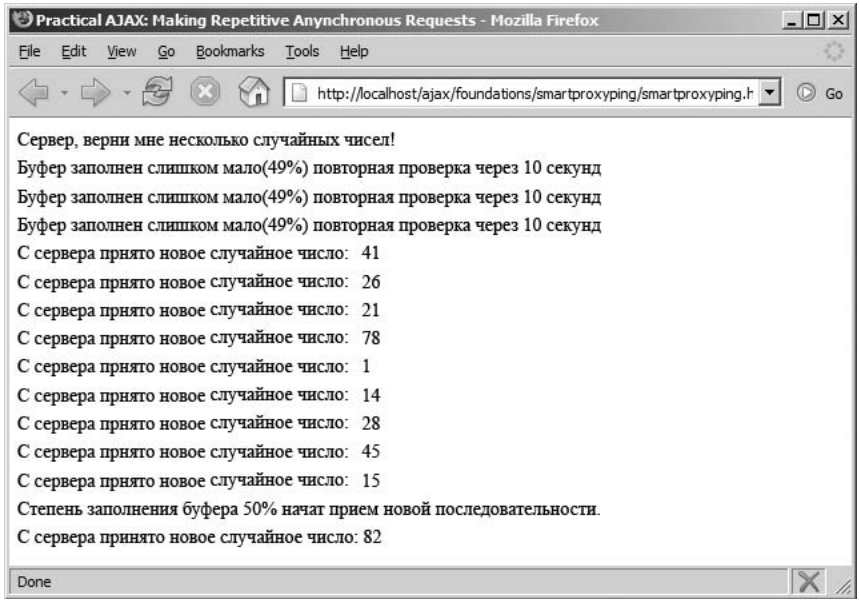

Рис. 3.15. *Выполнение повторяющихся асинхронных запросов*

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>
```

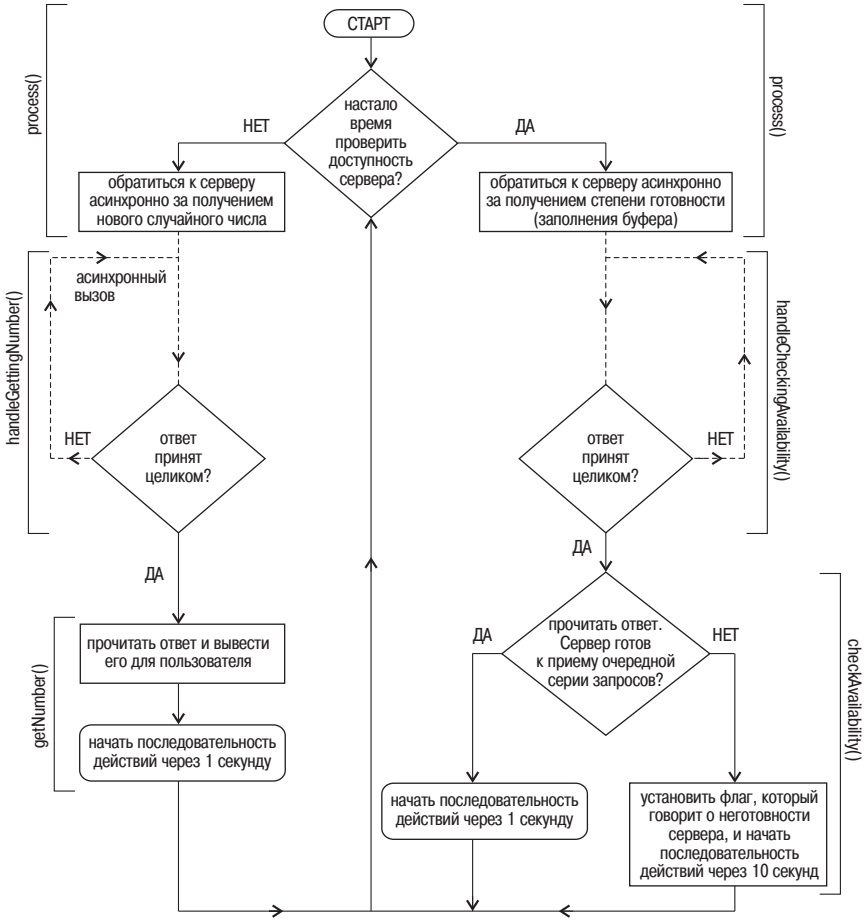

Рис. 3.16. *Блок-схема алгоритма приема случайных чисел*

```
            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.*

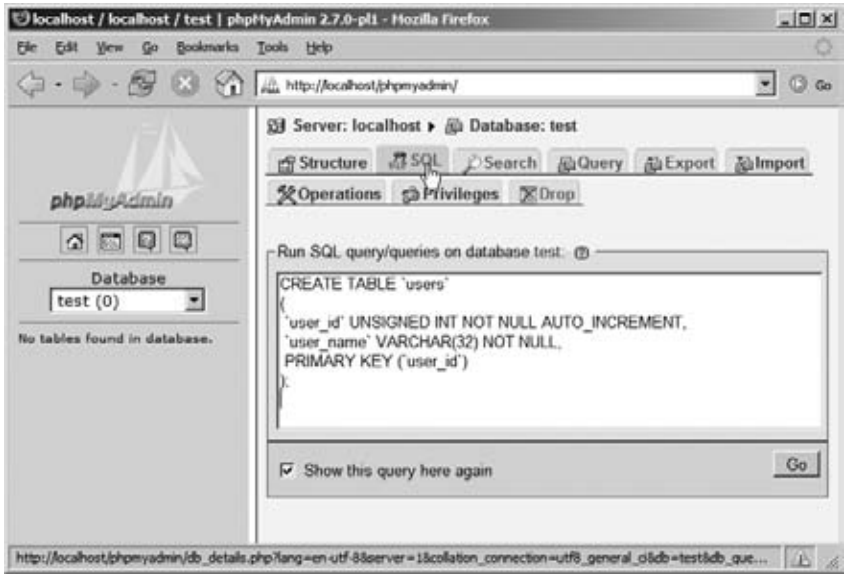

Рис. 3.17. *Исполнение кода SQL с помощью phpMyAdmin*

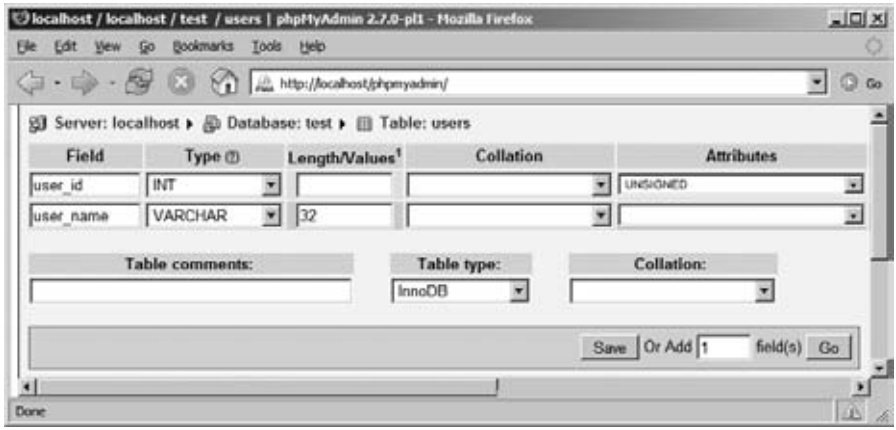

Рис. 3.18. *Создание новой таблицы с помощью phpMyAdmin Designer*

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).

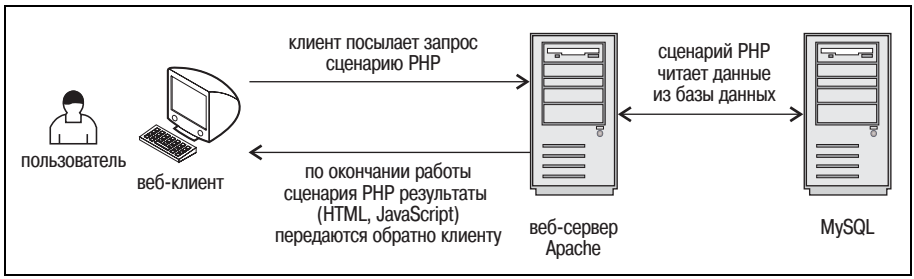

Рис. 3.19. *Пользователь подключается к базе данных через промежуточное программное обеспечение*

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).

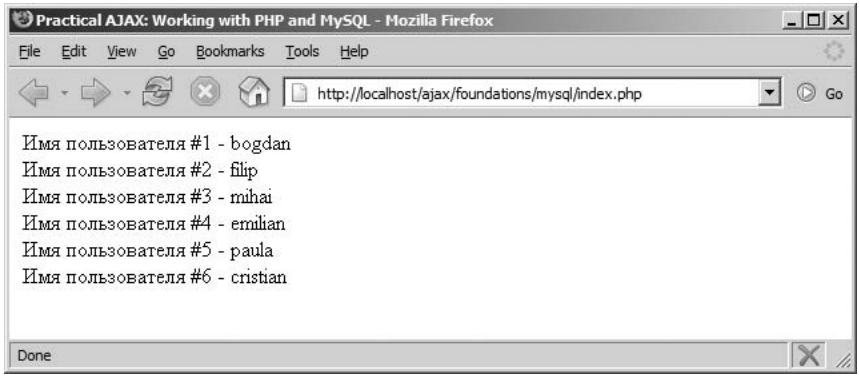

***Рис. 3.20.** Эти имена пользователей прочитаны из базы данных*

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.

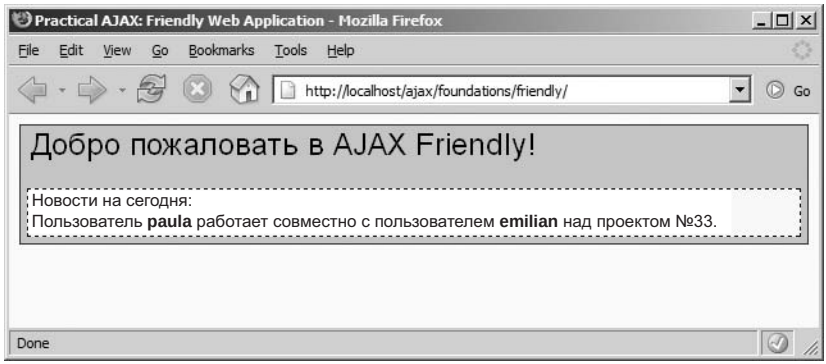

Рис. 3.21. *Веб-приложение Friendly*

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:

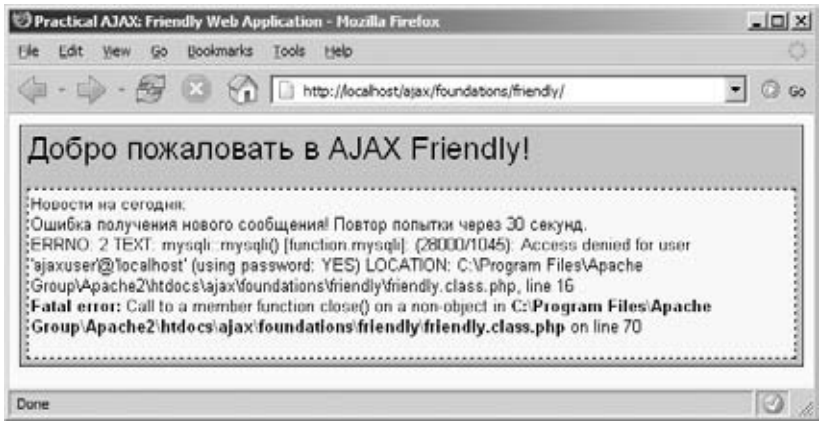

Рис. 3.22. *Если пароль доступа к базе данных утерян, выводится подробное сообщение об ошибке*

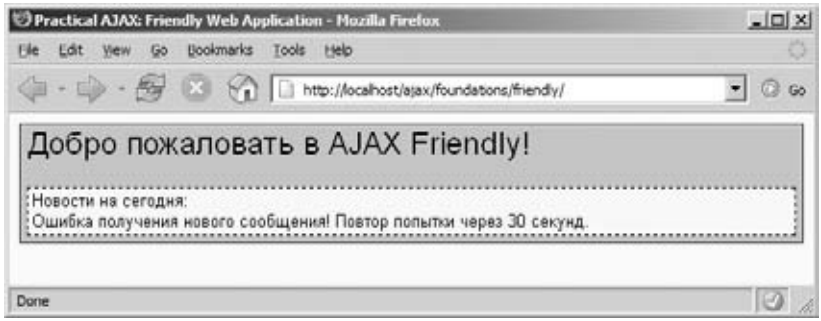

Рис. 3.23. *Более дружественное сообщение об ошибке*

```
<!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.