Вадим Будилов

PHP 5

Экспресс-курс

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06+800.92PHP5
ББК 32.973.26-018.1
Б90

Будилов В. А.

Б90 PHP 5. Экспресс-курс. — СПб.: БХВ-Петербург, 2005. — 240 с.: ил.

ISBN 5-94157-615-3

Рассмотрены базовые конструкции языка PHP 5, на примерах показаны основные приемы написания базовых сценариев, наиболее употребительных при разработке Web-приложений. Важное место в книге уделено способам работы с реляционными базами данных и с данными в формате XML. Подробно представлен материал об использовании стилей XSL для преобразования и представления XML-данных. Рассмотрены вопросы создания "с нуля" клиент-серверных приложений, разработки SOAP-приложений. Книга полезна как начинающим разработчикам, так и тем, кто уже имеет опыт создания интернет-приложений. Также она может использоваться как справочник.

Для Web-разработчиков

УДК 681.3.06+800.92PHP5
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Алия Амирова*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 14.03.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 19,35.
Тираж 4000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-615-3

Оглавление

Введение

Язык PHP сравнительно молод, ему нет и десяти лет. Несмотря на свою молодость, он пользуется заслуженной популярностью, которая растет год от года. Язык PHP сочетает в себе простоту и достаточную для создания полноценных Web-приложений глубину, при этом предоставляя возможности для решения и более широких, не ограниченных Web-программированием, задач.

Разработка Web-приложений, независимо от степени их сложности, подразумевает создание пользовательского интерфейса, где немаловажное, а порой и наиболее существенное значение имеет дизайн, в который "вписана" функциональность пользовательского интерфейса. В данной книге вопросы разработки дизайна не будут освящены. Эту задачу решают дизайнеры, а наше пособие предназначено для тех, кто хочет научиться программировать, разрабатывать и создавать функциональные Web-страницы.

Web-программирование условно можно разделить на две части: это клиентское Web-программирование и серверное Web-программирование. Клиентское Web-программирование решает задачи отображения страниц в клиентских приложениях-браузерах. Основным инструментарием клиентского программирования являются языки HTML, JavaScript, VBScript, стили CSS, а также Java-апплеты и ActiveX-компоненты. Набор доступных средств, изначально встроенных в современные браузеры, постоянно увеличивается. При желании на сторону клиента можно переложить обработку XML-документов с использованием XSL-стилей.

Серверное программирование предполагает решение различных задач на стороне сервера, с которым взаимодействует клиент, посылая ему запросы. Основными задачами серверного Web-программирования обычно являются: обработка клиентских запросов, формирование ответа клиенту, работа с серверами баз данных. Этот перечень можно продолжать, поскольку он ограничивается исключительно требованиями к создаваемому приложению и возможностями языка. Серверные Web-приложения создаются с помощью

различных языков программирования, одним из которых является язык PHP.

Язык PHP изначально был задуман как язык серверного Web-программирования, который позволяет быстро и просто решать основные задачи, стоящие перед разработчиком Web-сайтов. Язык PHP тесно взаимодействует с отсылаемым браузеру клиентским кодом. *Глава 1* книги посвящена рассмотрению способов работы с основными языками, используемыми при создании Web-странички: языку HTML, объектной модели документа DOM, работе с объектами, событиями, свойствами с использованием языка клиентских сценариев JavaScript.

В *главе 2* показаны основные приемы работы с языком PHP на примерах. *Глава 3* представляет собой краткий обзор основных лексических конструкций языка PHP. В *главе 4* описаны возможные способы работы с данными, хранимыми как в привычных реляционных базах данных, так и в XML-представлении. В *главе 5* представлены продвинутые возможности работы с использованием языка PHP, не ограниченные Web-программированием. Здесь рассмотрены примеры создания сетевых клиент-серверных приложений, основы работы с Web-сервисами. Отражены воплощенные в новой — пятой — версии языка PHP возможности, в том числе возможность создания абстрактных классов и интерфейсов, возможность работать с модификаторами `public`, `protected`, `private`, использовать наследование, создавать конструкторы и деструкторы, создавать защищенные методы классов с применением модификатора `final`, создавать статические методы и переменные.

Книга построена таким образом, чтобы в максимально сжатые сроки приблизить читателя к понимаю самых современных технологий. В книге не рассматриваются средства разработки — приоритет отдан технологиям. Приводимые примеры можно создавать в любом удобном для читателя редакторе, включая Блокнот.

Все, что требуется для того, чтобы научиться использовать этот язык, — это желание и компьютер.

ГЛАВА 1

Первые странички

Создать собственный сайт или страничку в Интернете нетрудно, для этого используется язык гипертекстовой разметки HTML. Чтобы ваша страничка ожила, стала интерактивной, необходимо, чтобы сервер, на котором она расположена, смог выполнить определенные действия перед отправкой странички в окно браузера. Описание таких действий содержится в программах. Программы, работающие совместно с Web-сервером, могут быть написаны на разных языках. Эта книга посвящена одному из наиболее популярных языков программирования для Web — языку PHP, но изучение этого языка невозможно без знания основ работы с языком гипертекстовой разметки HTML.

Язык гипертекстовой разметки HTML

HTML — это язык разметки гипертекста. Web-консорциум принял этот язык как стандарт представления информации в Интернете. HTML-файл можно создать в простейшем текстовом редакторе, например, в Блокноте. Для облегчения работы с HTML существует большое количество специально созданных редакторов, в числе наиболее популярных — редактор Microsoft FrontPage.

При загрузке HTML-файла браузер автоматически анализирует код HTML и на основе описанных в нем HTML-элементов создает соответствующие объекты. Созданные таким образом объекты подчиняются набору стандартных правил, образуя объектную модель документа (DOM). Элементы HTML описываются при помощи тэгов HTML. Сформированные браузером объекты обладают стандартным набором методов, свойств и событий, присущих им. Существуют средства, позволяющие управлять методами и свойствами объектов, а также реагировать на возникающие в них события. Наиболее

удобные средства для работы с объектами предоставляют языки сценариев, такие как JavaScript, VBScript.

Язык разметки гипертекста позволяет легко и просто организовать форматирование текста и графической информации.

Самый простой и эффективный способ научиться пользоваться средствами HTML — рассмотреть примеры.

Простейший HTML-файл

Каждый HTML-файл должен открываться тэгом `<html>`. Тело HTML-документа помещается в элементе, описываемом тэгом `<body>`. В листинге 1.1 приведен пример предельно простого HTML-файла.

Листинг 1.1. Простейший HTML-файл

```
<html>
<body>
Это простой текст в элементе 'body'.
</body>
</html>
```

Наберем приведенный выше код в простейшем редакторе, например в Блокноте, и сохраним под именем file1.html. Для просмотра в браузере нужно открыть этот файл, щелкнув на нем мышью, в результате появится наша страничка (рис. 1.1).

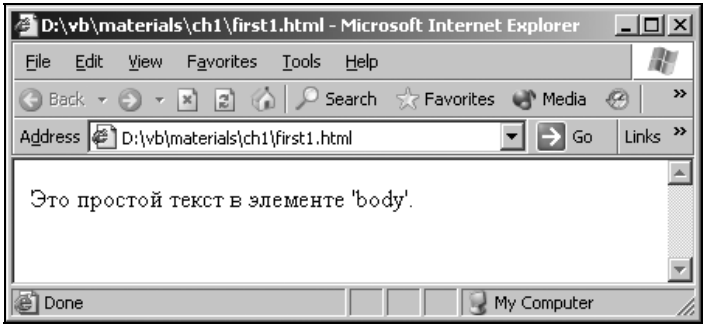

Рис. 1.1. Страничка загружена

Элемент `<body>` обладает набором свойств. В частности, оказывается полезным задание цвета фона или картинки, используемой в качестве такого фона. Цвет фона задается при помощи атрибута `bgcolor`, значением которого можно указать цвет в формате RGB (`<body bgcolor="#bcdebc">`) или задать цвет в виде его названия: `<body bgcolor="red">`. По правилам все значения

атрибутов элементов заключаются в кавычки. Заданные атрибуты преобразуются браузером в значения соответствующих свойств объектов.

В дальнейшем, чтобы проверить, как работают приведенные примеры, нужно сохранить их в файле с произвольным именем и расширением либо html, либо htm, после чего загрузить файл в браузер, щелкнув на нем мышью.

Параграфы

Для формирования параграфов используем тэг `<p>` и соответствующий ему закрывающий тэг `</p>` (листинг 1.2).

Листинг 1.2. Описываем параграфы

```
<html>
<body>
<p>Первый параграф.</p>
<p>Еще параграф.</p>
<p>Новый параграф.</p>
<p>Элемент параграфа описывается при помощи тэга p.</p>
</body>
</html>
```

Разрыв строки

Разрыв строки задается с помощью тэга `<br>` (листинг 1.3).

Листинг 1.3. Разрыв строки

```
<html>
<body>
Для простого <br> переноса <br>(разрыва) <br> строки используется другой
тэг - <br> br.
</body>
</html>
```

Заголовки

Заголовки шести уровней можно описать при помощи тэгов `<h1>`—`<h6>` (листинг 1.4).

Листинг 1.4. Заголовки

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

```
<P>Заголовки h1 - h6 используются для описания заголовков разных
уровней</p>
<h1>Заголовок первого уровня</h1>
<h2>Заголовок второго уровня</h2>
<h3>Заголовок третьего уровня</h3>
<h4>Заголовок четвертого уровня</h4>
<h5>Заголовок пятого уровня</h5>
<h6>Заголовок шестого уровня</h6>
</body>
</html>
```

Заголовки помогают организовать представление текста наиболее выгодным образом (рис. 1.2).

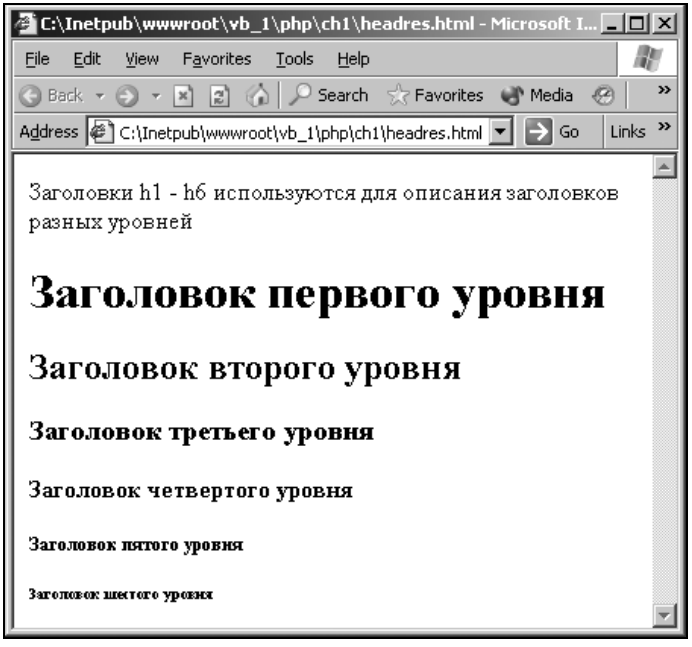

Рис. 1.2. Заголовки

Выравнивание текста

Приведем еще один пример использования атрибутов при работе с текстом (листинг 1.5). Текст можно выровнять с применением атрибута `align`, значениями этого атрибута могут быть: `left` (выравнивание по левому краю), `right` (выравнивание по правому краю), `center` (выравнивание по центру).

Листинг 1.5. Выравнивание

```
<h1>Выравнивание</h1>
<p align="right">Многие текстовые элементы позволяют
задавать способ выравнивания текста.
Текст этого параграфа будет
выравнен по правому краю.</p>
```

На рис. 1.3 показан пример выравнивания текста по правому краю.

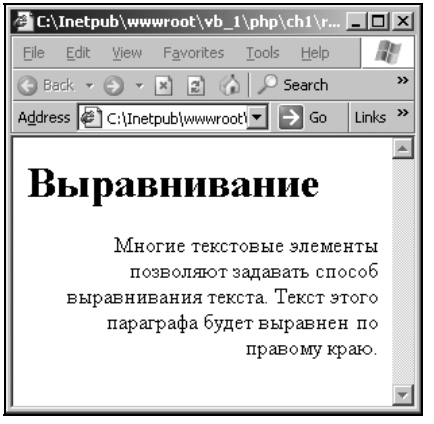

Рис. 1.3. Выравнивание текста по правому краю

Горизонтальная линия

Горизонтальная линия вставляется с помощью тэга `<hr>` (листинг 1.6).

Листинг 1.6. Горизонтальная линия

```
<html>
<body>
<p>тэг hr используется для вставки горизонтальной черты:</p>
<hr>
<p>Текст параграфа</p>
<hr>
<p>Текст параграфа</p>
<hr>
<p>Текст еще одного параграфа</p>
</body>
</html>
```

Для этого элемента можно указать его ширину с помощью атрибута `width`.

Комментарии

Комментарии служат для пояснения программного кода и не влияют на то, как будет отображаться HTML-код. Комментарии выделяются при помощи тэгов `<!-- ... -->` (листинг 1.7).

Листинг 1.7. Комментарии

```
<html>
<body>
<!-- Этот текст не будет отображен -->
<p>Комментарий не будет показан в окне браузера</p>
</body>
</html>
```

Использование комментариев особенно полезно тогда, когда код страницы становится сложным и большим.

Цвет фона документа

Цвет фона документа указывается в виде значения атрибута `bgcolor` (рис. 1.4), как это показано в листинге 1.8.

Листинг 1.8. Цвет фона

```
<html>
<body bgcolor="green">
<h2>Задаем цвет фона.</h2>
</body>
</html>
```

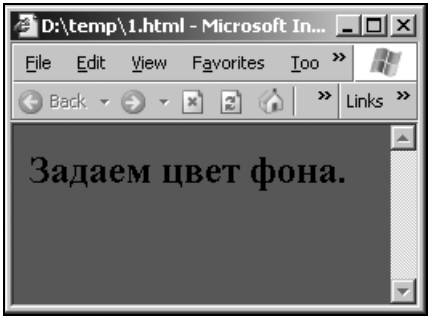

Рис. 1.4. Задан зеленый фон

Варианты форматирования текста

Ниже представлено несколько вариантов форматирования текста (листинг 1.9). Это форматирование осуществляется путем создания самостоятельных элементов, содержащих текст (рис. 1.5).

Листинг 1.9. Форматирование текста

```
<html>
<body>
<h3>Задаем форматирование текста</h3>
<b>Полужирный текст</b>
<br>
<strong>
Еще вариант выделения текста.
</strong>
<br>
<big>
Крупный текст
</big>
<br>
<em>
Выделенный текст
</em>
<br>
<i>
Текст курсивом
</i>
<br>
<small>
Маленький текст
</small>
<br>
Текст с
<sub>
нижним индексом
</sub>
<br>
Текст с
<sup>
верхним индексом
</sup>
</body>
</html>
```

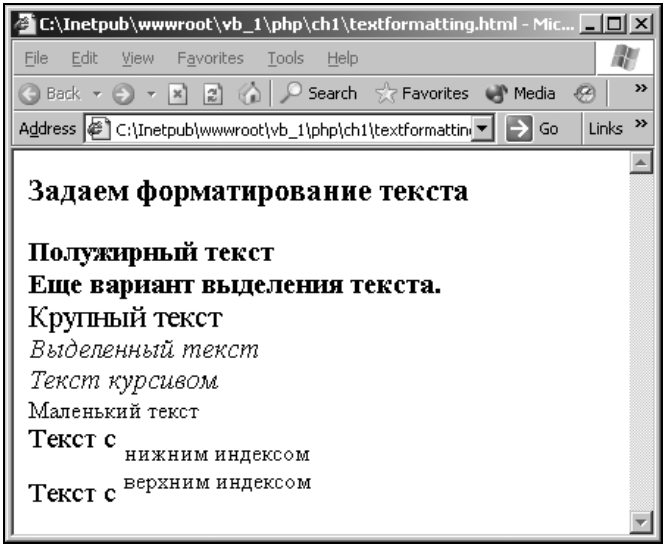

Рис. 1.5. Форматирование текста

Заранее отформатированный текст

Иногда требуется вывести на экран текст в том виде, в каком он изначально отформатирован. Это возможно с помощью тэга `<pre>`. При этом все переносы и пробелы будут выводиться так, как записано в тексте, и не будут игнорироваться, как это обычно происходит при отображении стандартного текста (листинг 1.10).

Листинг 1.10. Заранее отформатированный текст

```
<html>
<body>
<pre>
Это заранее отформатированный текст.
В этом тексте будут сохранены все пробелы
и
переносы
строк.
</pre>
<p>Такой формат часто используется для отображения исходных кодов
программ:</p>
<pre>
for k = 1 to 10
...
next k
```

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

Задание гиперссылок

Один из важных моментов — задание гиперссылок. Это делается с помощью тэга `<a href='URL'>`, как показано в листинге 1.11. URL — универсальный указатель ресурса (рис. 1.6).

Листинг 1.11. Гиперссылка

```
<html>
<a href="http://www.yahoo.com/">Это гиперссылка. Щелкни меня!</a>
</body>
</html>
```

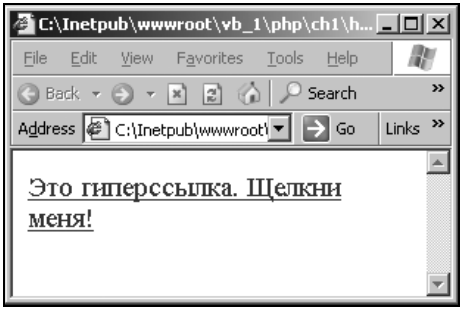

Рис. 1.6. Гиперссылка

Отображение документа по ссылке в новом окне

Чтобы документ, открытый по ссылке был выведен в новом окне, можно задать в качестве значения атрибута `target` для элемента `a` имя несуществующего окна (листинг 1.12).

Листинг 1.12. Гиперссылка в новом окне

```
<html>
<body>
<a href="http://www.yahoo.com" target="new_window">
откроем документ в новом окне.</a>
<p>
```

```
Если в качестве значения параметра target
указать имя несуществующего окна
(следует указать имя, которого нет ни у одного из открытых окон),
то документ будет показан в новом окне.
</p>
</body>
</html>
```

Фреймы

Фреймы представляют собой самостоятельные окна в окне браузера. Пример задания фреймов приведен в листинге 1.13, а результат выполнения кода показан на рис. 1.7.

Листинг 1.13. Фреймы

```
<html>
<frameset cols="25%,50%,25%">
<frame src="a.html">
<frame src="b.html">
<frame src="c.html">
</frameset>
</html>
```

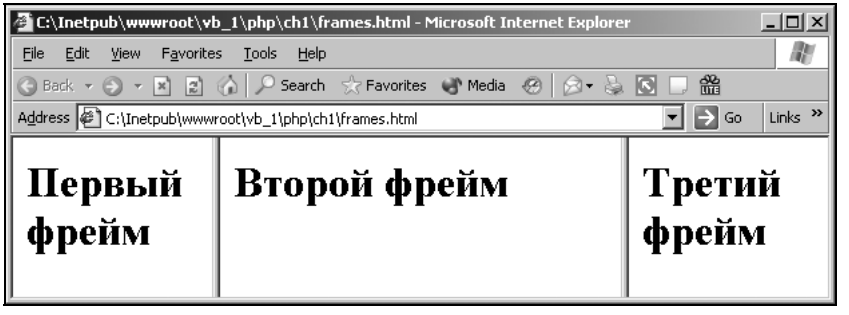

Рис. 1.7. Фреймы

Горизонтальные фреймы

В листинге 1.14 приведен пример задания горизонтальных фреймов (рис. 1.8).

Листинг 1.14. Горизонтальные фреймы

```
<html>
<html>
```

```
<frameset rows="25%,50%,25%">
<frame src="a.html">
<frame src="b.html">
<frame src="c.html">
</frameset>
</html>
```

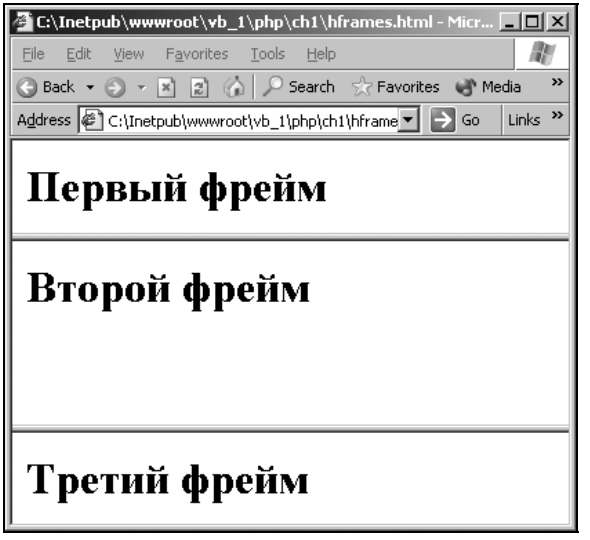

Рис. 1.8. Горизонтальный фреймы

Плавающие фреймы

Фрейм можно встроить в поток HTML-текста. Для этого используется тэг `<iframe>`. Пример кода приведен в листинге 1.15, а результат его выполнения — на рис. 1.9.

Листинг 1.15. Плавающий фрейм

```
<html>
<body>
<iframe src="http://yahoo.com"></iframe>
<p>Плавающий фрейм.</p>
<p>Такой фрейм размещается внутри контекста документа.</p>
</body>
</html>
```

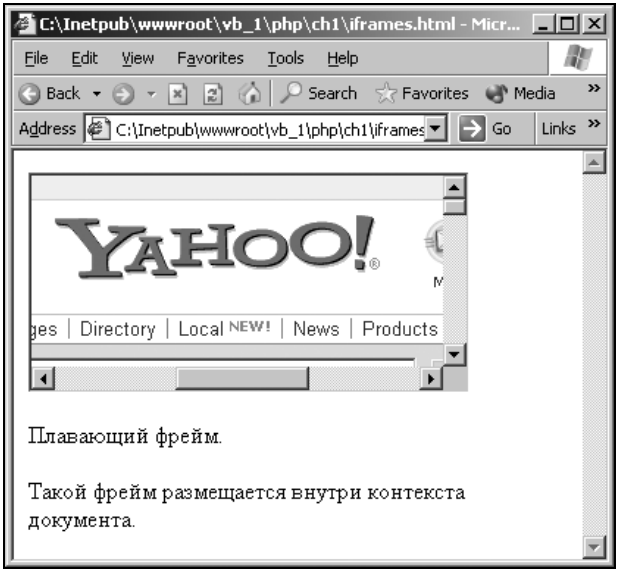

Рис. 1.9. Плавающий фрейм

Таблицы

Таблицы можно описать с применением тэга `<table>` (листинг 1.16 и рис. 1.10).

Листинг 1.16. Таблицы

```
<html>
<body>
<p>
Таблица открывается при помощи тэга table.
Строка таблицы открывается тэгом tr.
Каждая ячейка в строке начинается с тэга td.
</p>
<h4>Один столбец:</h4>
<table border="1">
<tr>
<td>1000</td>
</tr>
</table>
<h4>Одна строка и три столбца:</h4>
<table border="1">
<tr>
```

```
<td>1000</td>
<td>2000</td>
<td>3000</td>
</tr>
</table>
<h4>Две строки и три столбца:</h4>
<table border="1">
<tr>
<td>1000</td>
<td>2000</td>
<td>3000</td>
</tr>
<tr>
<td>4000</td>
<td>5000</td>
<td>6000</td>
</tr>
</table>
</body>
</html>
```

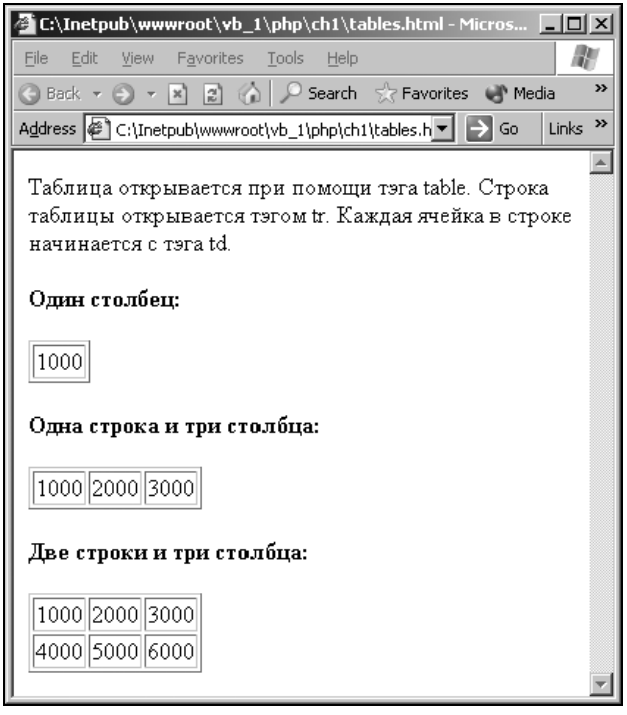

Рис. 1.10. Таблицы

Ненумерованный список

Ненумерованный список открывается тэгом `<ul>`, каждая запись описывается с применением `<li>` (листинг 1.17 и рис. 1.11).

Листинг 1.17. Ненумерованный список

```
<html>
<body>
<h4>Ненумерованный список:</h4>
<ul>
<li>Матрешка</li>
<li>Капуста</li>
<li>Разочарование</li>
</ul>
</body>
</html>
```

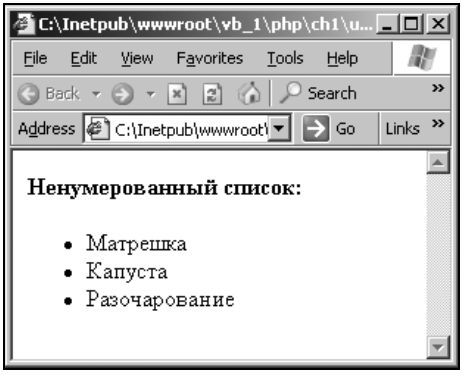

Рис. 1.11. Ненумерованный список

Нумерованный список

Нумерованный список открывается тэгом `<ol>`, каждая запись описывается с применением `<li>` (листинг 1.18 и рис. 1.12).

Листинг 1.18. Нумерованный список

```
<html>
<body>
<h4>Нумерованный список:</h4>
<ol>
```

```
<li>Матрешка</li>
<li>Капуста</li>
<li>Разочарование</li>
</ol>
</body>
</html>
```

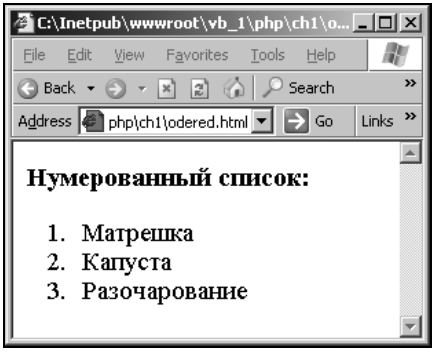

Рис. 1.12. Нумерованный список

Текстовое поле

Чтобы организовать поле для ввода текста, используется тэг `<input type="text">` (листинг 1.19 и рис. 1.13).

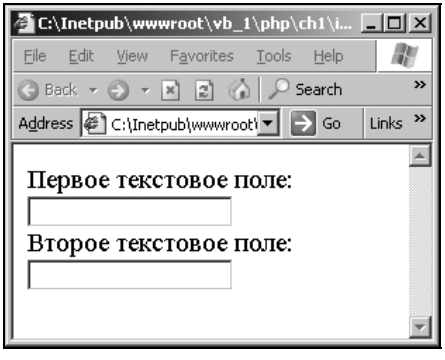

Рис. 1.13. Текстовое поле

Листинг 1.19. Текстовое поле

```
<html>
<body>
<form>
```

```
Первое текстовое поле:
<input type="text" name="firstname">
<br>
Второе текстовое поле:
<input type="text" name="lastname">
</form>
</body>
</html>
```

Флажок

Элемент управления "флажок" можно создать при помощи тэга `<input type="checkbox">` (листинг 1.20 и рис. 1.14).

Листинг 1.20. Флажок

```
<html>
<body>
<form>
Флажок:
<input type="checkbox" name="first">
<br>
Еще флажок:
<input type="checkbox" name="second">
</form>
</body>
</html>
```

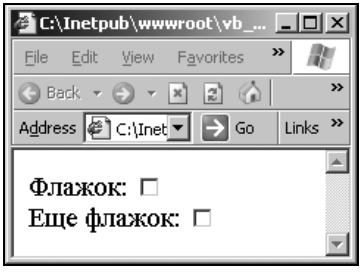

Рис. 1.14. Флажки

Выпадающий список

Выпадающий список создается посредством тэга `<select>`, каждый элемент списка описывается при помощи `<option>` (листинг 1.21 и рис. 1.15).

Листинг 1.21. Выпадающий список

```
<html>
<body>
<form>
<select name="cars">
<option value="volvo">Атмосфера
<option value="saab">Жимолость
<option value="fiat">Неосознанная психофизическая зависимость
<option value="audi">Внезапность
<option value="volvo">Дрофы и немного себялюбия
<option value="saab">Норма и Тоска
<option value="fiat">Норма, мера и линза Френеля
<option value="audi">Чуден Днепр при тихой погоде
<option value="volvo">Жизнь, деньги и любовь
<option value="saab">Желанность и жеманство
<option value="fiat">Конечные приращения
<option value="audi">Острая безысходная необходимость
</select>
</form>
</body>
</html>
```

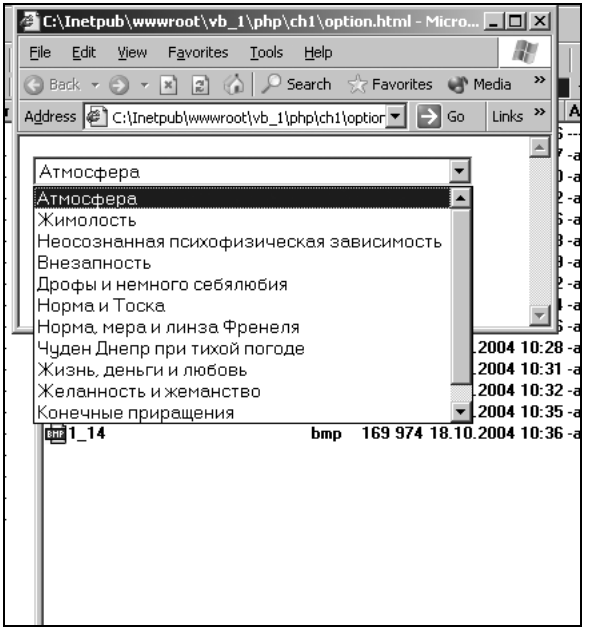

Рис. 1.15. Список

Текстовая область

Текстовая область описывается с применением тэга `<textarea>` (листинг 1.22 и рис. 1.16).

Листинг 1.22. Текстовая область

```
<html>
<body>
<p>
Пример текстовой области.
</p>
<textarea rows="10" cols="30">
Верхушки сосен были обожжены отсветом солнечных искр,
разбрызганных несбывшимися надеждами.
</textarea>
</body>
</html>
```

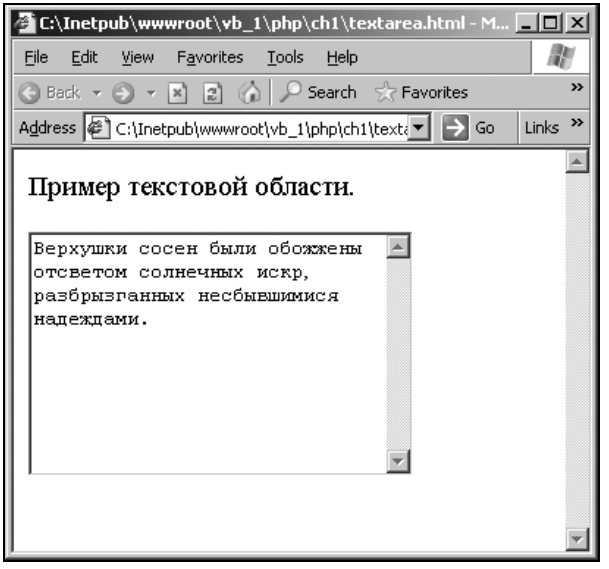

Рис. 1.16. Текстовая область

Кнопка

Элемент управления "кнопка" можно вставить с использованием тэга `<input type="button">` (листинг 1.23 и рис. 1.17).

Листинг 1.23. Кнопка

```
<html>
<body>
<form>
<input type="button" value=" Кнопка я! Привет! ">
</form>
</body>
</html>
```

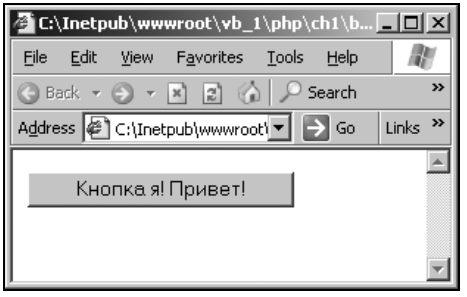

Рис. 1.17. Кнопка

Почта

Можно отправить почту прямо из HTML-документа. Для этого используется элемент управления `<form action="MAILTO:name@pochta.ru" method="post" enctype="text/plain">` (листинг 1.24).

Листинг 1.24. Отправка почты

```
<html>
<body>
<form action="MAILTO:name@pochta.ru" method="post" enctype="text/plain">
<h3>Послать почту.</h3>
Имя:<br>
<input type="text" name="imya"
value="vvedite imya" size="20">
<br>
e-mail:<br>
<input type="text" name="pochta"
value="vvedita e-mail" size="20">
<br>
Сообщите что:<br>
```

```
<input type="text" name="pole"
value="vvedite chto hotite" size="40">
<br><br>
<input type="submit" value="Send">
<input type="reset" value="Reset">
</form>
</body>
</html>
```

Позиционирование HTML-элементов с помощью таблиц

Таблицы в HTML используются для задания общей схемы форматирования. Таблицы — это основной механизм позиционирования элементов в пределах одной страницы и одного слоя. Таблицы используются для организации пространства просматриваемой страницы в окне браузера. В таблицах можно размещать не только текстовую информацию, но и изображения.

Поскольку HTML-код не имеет возможности непосредственно определять фиксированный внешний вид выводимой в окне браузера HTML-страницы, то полезным было бы иметь хотя бы некоторые примерные ограничения на относительное местоположение элементов HTML-страницы. В этом нам помогают таблицы.

Простая таблица

Начнем с примера. Оформим в виде HTML-таблицы перечень шрифтов и задающих их тэгов. На экране вновь созданная таблица выглядит так, как показано на рис. 1.18. Поскольку ваш браузер может быть настроен несколько иначе, конкретный вид таблицы, показываемой на экране в окне браузера, может несколько отличаться, но относительное положение элементов, расположенных в таблице, останется неизменным.

Созданный нами для описания такой таблицы файл table1.htm показан в листинге 1.25.

Листинг 1.25. Файл table1.htm

```
<html>
<head>
<title>Таблица</title>
</head>
<body>
<p> Таблица. <i>Еще простая таблица</I>.</p>
```

```
<table border="1" width="100%">
<tr>
<td width="30%">Ярлык </td>
<td width="70%">Соответствующий ярлыку шрифт</td>
</tr>
<tr>
<td width="30%">&lt;TT&gt;</td>
<td width="70%"><TT>Телеграфный шрифт</tt> </td>
</tr>
<tr>
<td width="30%">&lt;I&gt;</td>
<td width="70%"><i>Курсив</i></td>
</tr>
<tr>
<td width="30%">&lt;b&gt;</td>
<td width="70%"><b>Жирный шрифт</B></td>
</tr>
<tr>
<td width="30%">&lt;big&gt;</td>
<td width="70%"><big>Крупный шрифт</big></td>
</tr>
<tr>
<td width="30%">&lt;small&gt;</td>
<td width="70%"><small>Мелкий шрифт</small><font FACE="Times New Roman">
</font></td>
</tr>
<tr>
<td width="30%">&lt;STRIKE&gt; или &lt;S&gt;</td>
<td width="70%"><s>Перечеркнутый шрифт<s></td>
</tr>
<tr>
<td width="30%">&lt;u&gt;</td>
<td width="70%"><u>Подчеркнутый шрифт</u></td>
</tr>
</table>
</body>
</html>
```

Что нового мы видим в тексте этого файла? Появился тэг `<table>`. Между этим открывающим тэгом и соответствующим ему закрывающим тэгом `</table>` располагается таблица. Таблица состоит из строк и столбцов. Каждая строка начинается тэгом `<tr>` и заканчивается тэгом `</tr>`. Каждый столбец описывается внутри строки с помощью открывающего тэга `<td>` и закрывающего тэга `</td>`. Вот и все основные компоненты таблицы.

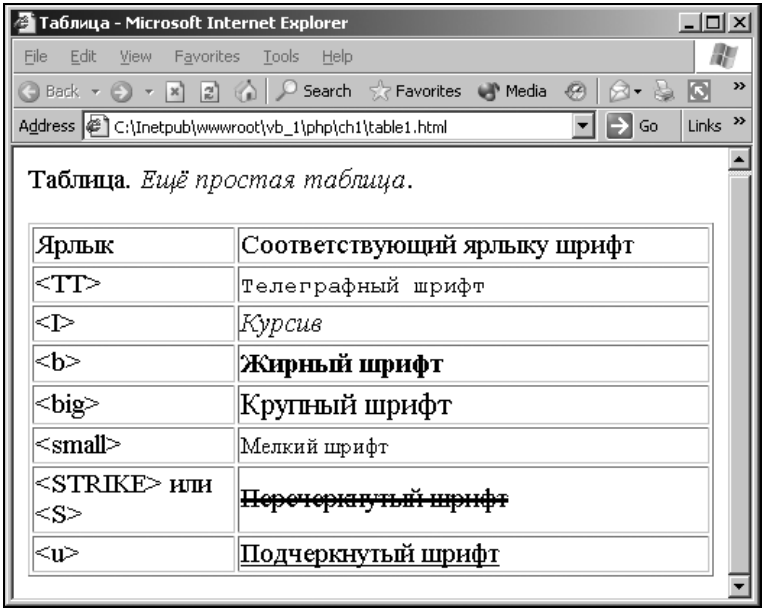

Рис. 1.18. Простая таблица

В тэгах `<table>`, `<tr>`, `<td>` можно указать свойства. В нашем случае мы задали параметр ширины таблицы в виде `width= "100%"`. Это означает, что таблица займет всю ширину окна браузера. В параметрах столбцов мы также указываем ширину столбца в процентах от общей ширины таблицы. Если изменить значения ширины столбцов, внешний вид таблицы изменится.

В тексте файла встречаются странные комбинации символов: `&lt;` и `&gt;`. Такие комбинации используются для того, чтобы дать возможность браузеру отобразить на экране специальные символы. Если в тексте файла написать специальный символ в обычном его виде, то браузер воспримет этот символ, как элемент HTML-кода и соответствующим образом обработает его. Чтобы графическое представление специального символа было выведено на экран, используются комбинации символов. В нашем случае комбинации `&lt;` соответствует левая угловая скобка <, а комбинации `&gt;` — правая угловая скобка >. Если в тексте HTML-кода записать `< ... >`, то браузер воспримет такую последовательность, как тэг. Но если вместо этого написать `&lt;...&gt;`, то браузер выведет на экран <...>. Весь список комбинаций символов, соответствующих специальным символам, можно найти в справочниках.

Вложенные таблицы

Таблицы можно вкладывать друг в друга. В качестве примера рассмотрим код файла table3.htm (листинг 1.26). Здесь мы в ячейку первой строки вто-

рого столбца первой объемлющей таблицы вставили другую таблицу. Для того чтобы одна таблица четче выделялась на фоне другой, мы задали разные значения параметров цвета фона для этих таблиц. Также мы установили параметр выравнивания второй таблицы по центру и задали ее относительную общую ширину — 70 %, поэтому вложенная таблица занимает не всю ширину удочерившей ее ячейки. Как выглядят вложенные таблицы, показано на рис. 1.19.

Листинг 1.26. Файл table3.htm

```
<html>
<head>
<title>Таблица в таблице</title>
</head>
<body>
<p> Изучаем таблицы. Таблица в таблице.</p>

<table border="1" bgcolor="#eeeeee" width="100%">
<tr>
<td width="30%">Первая таблица. Первая строка, первый столбец. </td>
<td width="70%">Первая таблица. Первая строка, второй столбец. В эту
ячейку таблицы вложена вторая таблица.
<table border="1" bgcolor="#dddddd" align="center" width="70%">
<tr>
<td width="50%">Вторая таблица. Первая строка, первый столбец. </td>
<td width="50%">Вторая таблица. Первая строка, второй столбец. </td>
</tr>
<tr>
<td width="50%">Вторая таблица. Вторая строка, первый столбец.</td>
<td width="50%">Вторая таблица. Вторая строка, второй столбец </td>
</tr>
</table></td>
</tr>
<tr>
<td width="30%">Первая таблица. Вторая строка, первый столбец.</td>
<td width="70%">Первая таблица. Вторая строка, второй столбец </td>
</tr>

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

Вложенные таблицы позволяют со значительной степенью гибкости располагать элементы на страничке. Если установить параметр толщины рамки

таблицы `border` равным 0, то рамки станут невидимы, и пользователь не будет подозревать о том, что страница скомпонована при помощи таблиц.

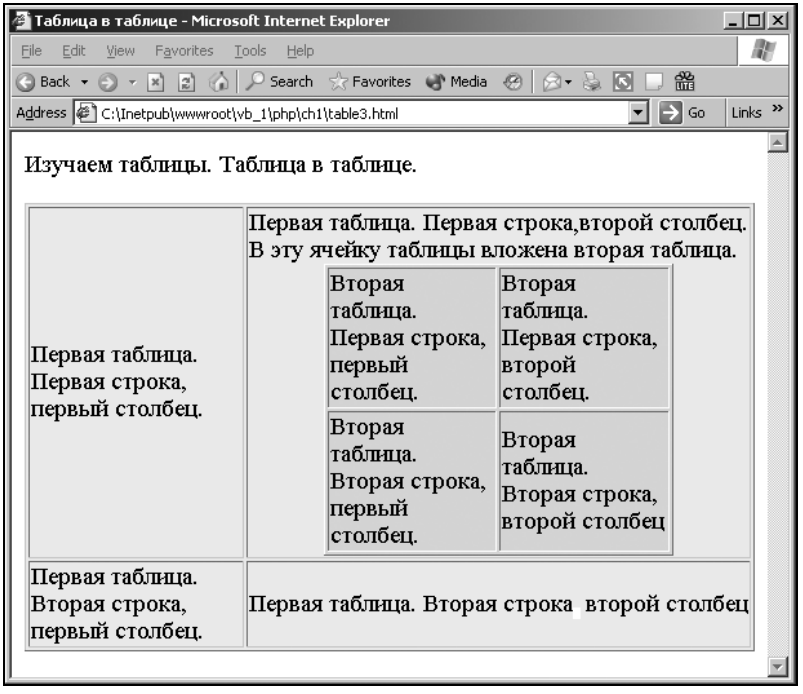

Рис. 1.19. Вложенные таблицы

Объединение ячеек в таблице

Если вам приходилось работать с текстовым процессором Word, то вы знаете, что в таблице несколько соседних ячеек можно объединить. Язык HTML позволяет производить такое объединение ячеек. Для объединения ячеек используются параметры `rowspan` и `colspan`. Их значение — это целое число, равное количеству объединяемых ячеек. Эти параметры задаются в тэге `<td>`. Параметры `rowspan` и `colspan` показывают соответственно количество строк и столбцов, которые "захвачены" данной ячейкой.

Рисунки и карты ссылок

Внедрение рисунка в страницу

Внедрить файл с изображением в HTML-страницу несложно. Для этого используется тэг `<img>`. "Изображение" по-английски — "image" ("образ"), со-

кращенный вариант этого слова используется в качестве HTML-тэга (рис. 1.20). Простейший файл, в котором используется изображение, можно написать в следующем виде (листинг 1.27).

Листинг 1.27. Файл pic1.htm

```
<html>
<head>
<title>
Вставляем рисунки - файл pic1.htm
</title>
</head>
<body bgcolor="white">
<img src="pic1.jpg">
<p>
Это простейший html-файл с рисунком.
</P>
</body>
</html>
```

Если загрузить этот файл в браузер, то в окне браузера мы увидим картинку pic1.gif (см. рис. 1.20).

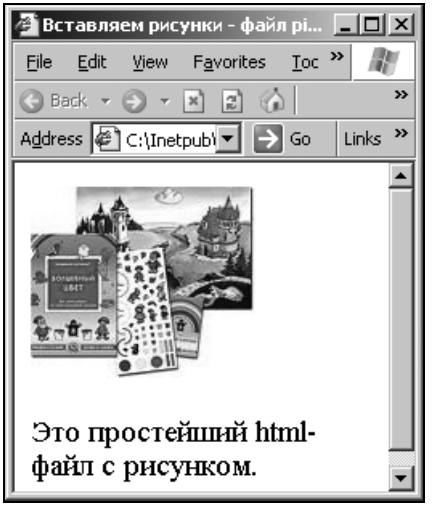

Рис. 1.20. Вставляем рисунок в HTML-страницу

В случае, если пользователь имеет очень медленную связь с Интернетом, он может отключить автоматическую загрузку изображений в браузер. Тогда вместо картинки мы можем показать пользователю текст. Для этого нужно в

тэг <img> вставить свойство Alt, которому присваиваем текстовую строку. Именно эта строка будет показана в окне браузера вместо рисунка, если рисунок по тем или иным причинам не был загружен:

```
<img src="pic1.jpg" Alt="Текст, выводимый вместо изображения">
```

Окно браузера будет выглядеть примерно так, как показано на рис. 1.21.

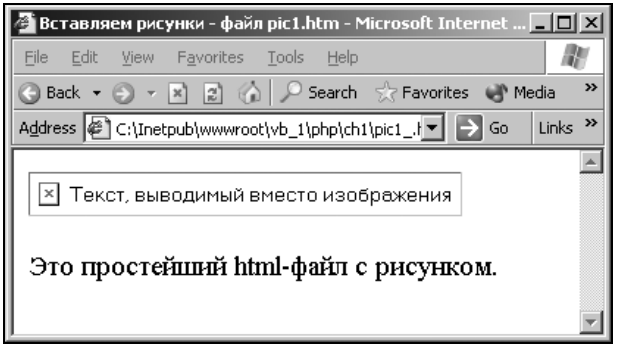

Рис. 1.21. Альтернативный текст, выводимый вместо рисунка

Рисунок фона

Рисунки можно использовать не только в качестве элементов, объектов страницы, документа. Рисунки можно использовать и как фон, на котором

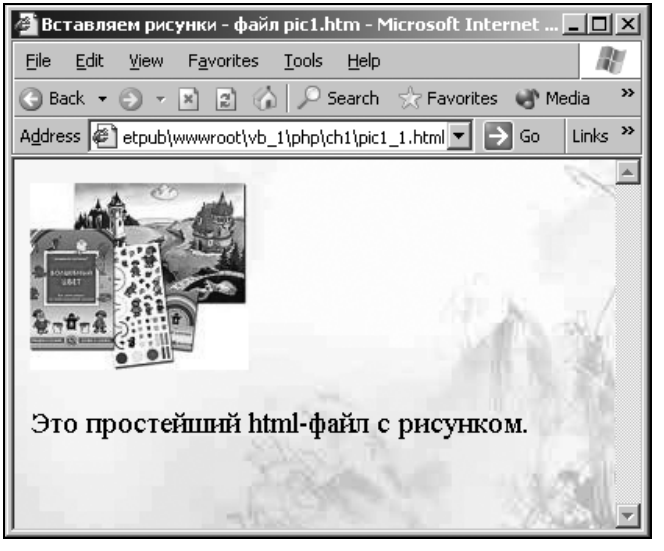

Рис. 1.22. В качестве фона использован рисунок

будет расположена страница. Файл рисунка с фоном указывается в тэге <body>:

```
<body bgcolor="white" background="back.gif">
```

Теперь, когда мы установили свойство background в тэге <body>, наша страница будет выглядеть так, как показано на рис. 1.22.

Здесь мы видим, что если одной копии рисунка не хватает, чтобы покрыть весь фон документа, т. е. если размер окна браузера больше размера рисунка, то рисунок фона воспроизводится столько раз, сколько необходимо для того, чтобы покрыть все окно браузера.

Свойства, указываемые в тэге *<img>*

В тэге <img> можно задать значения свойств (атрибутов), указанных в табл. 1.1.

Таблица 1.1. *Свойства тэга (элемента)* <img>

Свойство	Значение
Src	UDL-адрес файла, в котором хранится картинка, обязательное свойство
Alt	Текст, который будет показан вместо картинки, если картина не загружена
Align	Параметр выравнивания
Height	Высота картинки в пикселах
Width	Ширина картинки в пикселах
Border	Ширина границы картинки
Hspace	Горизонтальный отступ картинки
Vspace	Вертикальный отступ картинки
Usemap	URL-адрес карты ссылок
Ismap	Указатель использования серверной карты ссылок

Назначение свойств Src и Alt нам уже известно. Далее мы рассмотрим перечисленные здесь атрибуты более подробно.

Расположение картинки на странице

Свойство Align позволяет определить местоположение картинки в окне браузера. Данное свойство может принимать следующие значения:

- bottom
- middle

- top
- left
- right

Параметр bottom задает выравнивание по основной линии, т. е. нижний край рисунка будет располагаться по основной текущей линии (в тексте). Параметр middle соответствует выравниванию центра рисунка посередине по вертикали от текущей базовой линии. Значению top соответствует выравнивание по верхней границе рисунка с учетом верхней границы текущего текста.

Рисунки являются плавающими элементами страницы, их расположение относительно друг друга и по отношению к тексту определяется параметрами left и right.

Можно указать размер рисунка (рис. 1.23) с помощью атрибутов height и width:

```
<img src="pic1.jpg"  height="133px" width="200px">
```

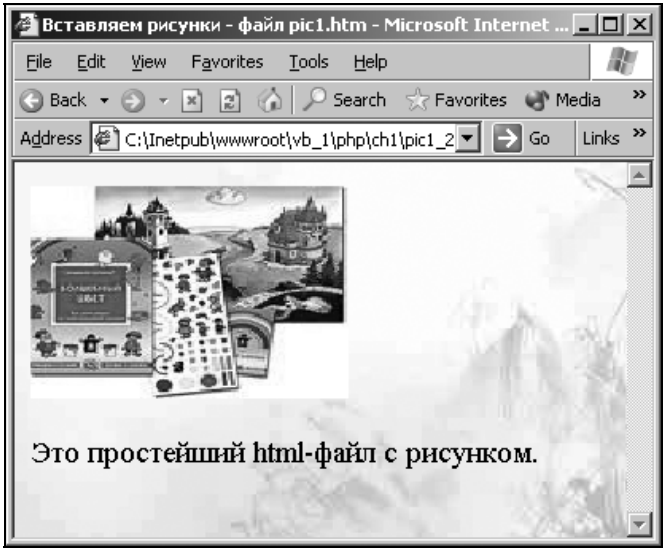

Рис. 1.23. Задаем размеры рисунка с помощью height и width

Более того, задавая искаженные параметры рисунков, можно деформировать их по своему желанию (рис. 1.24):

```
<img src="pic1.jpg" vspace="5" hspace="15" width="200" height="150">
<img src="pic1.jpg" vspace="5" hspace="15">
<img src="pic1.jpg" vspace="5" hspace="15" width="80" height="300">
```

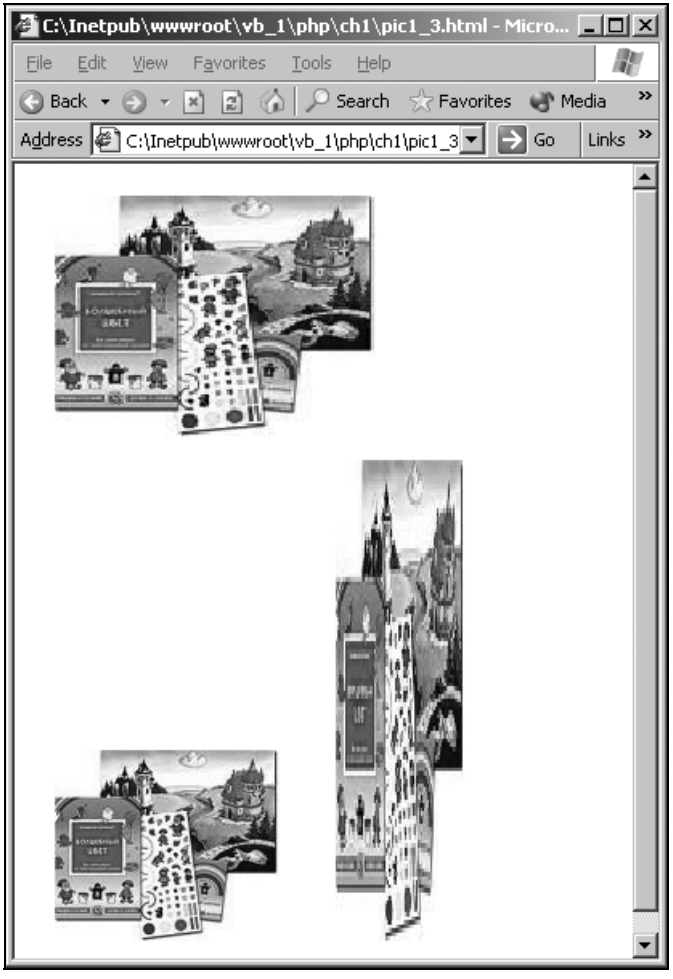

Рис. 1.24. Искажаем изображения

Форматирование изображений с использованием таблиц

Механизм таблиц предоставляет удобный способ форматирования изображений и отделения их от текста наиболее легким способом. Создадим простой файл table1.htm, содержащий HTML-код, в который будет записан текст и изображение, относительное положение их будет определено с помощью таблицы. Код файла показан в листинге 1.28, а результат — на рис. 1.25.

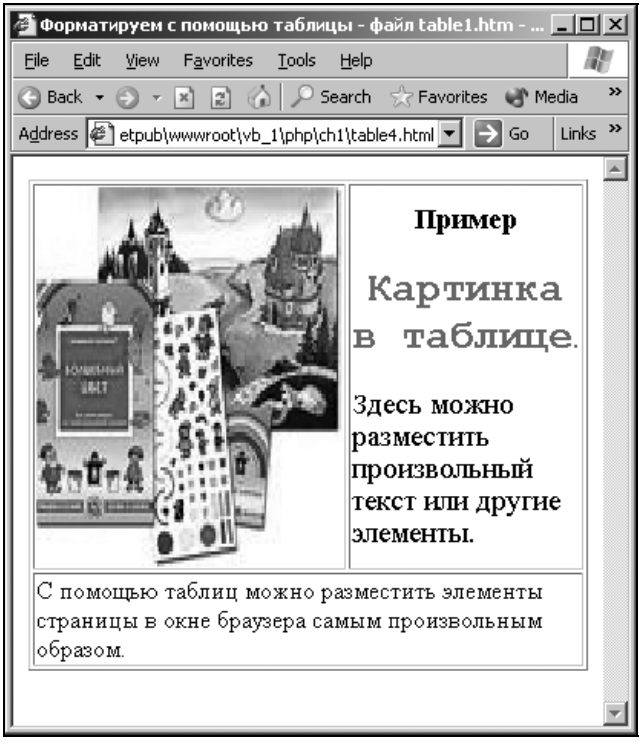

Рис. 1.25. Форматируем страницу с помощью таблиц

Листинг 1.28. Файл table1.htm

```
<html>
<head>
<title> Форматируем с помощью таблицы - файл table1.htm </title>
</head>

<body bgcolor="white">

<table border="1" width="100%">
<tr>
<td width="35%"><img src="pic1.jpg" width="200" height="248"
alt="pic1.jpg (13212 bytes)"></td>
<td width="65%"><p align="center"><strong>Пример </strong></p>
<p align="center"><font color="#808000"><font size="6"><b><tt>
Картинка в таблице</tt></b></font>. </font></p>
<p>Здесь можно разместить произвольный текст или другие элементы.</td>
</tr>
```

```
<tr>
<td width="100%" colspan="2"><font size="2">С помощью таблиц можно
разместить элементы страницы в окне браузера самым произвольным
образом.</font></td>
</tr>
</table>
</body>
</html>
```

Границы таблицы можно сделать невидимыми, тогда пользователь даже не будет подозревать о том, что вы использовали таблицу для форматирования содержимого HTML-страницы (рис. 1.26):

```
<table border="0" width="100%">
```

Здесь мы задали толщину границы — 0, теперь граница таблицы не будет видима.

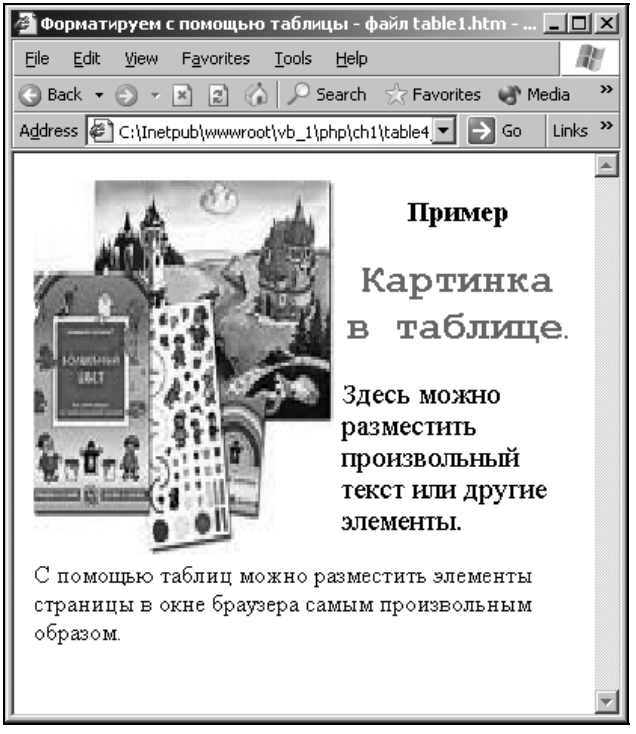

Рис. 1.26. Тот же файл с единственным изменением — отсутствуют границы в таблице

Использование изображения в качестве ссылки

Изображение можно использовать в качестве ссылки на другой ресурс Интернета или на другую страницу. Для этого рисунок необходимо поместить между тэгами якоря `<a href= "...">` и `</a>`.

```
<a href="bgif.htm"><img src="ball.gif" width="75"><a>
```

Динамический HTML

При помощи динамического HTML (или кратко DHTML) мы можем оживить статическую HTML-страничку, сделать ее интерактивной. При этом страничка будет "отвечать" на действия пользователя, производимые им в окне браузера. Многие свойства, описываемые в этом разделе, в значительной степени зависят от того, в каком браузере просматривается HTML-страничка. Это всегда следует иметь в виду при разработке динамических HTML-страниц. Желательно создавать такие странички, которые будут приемлемо отображаться в большинстве браузеров. Здесь мы рассмотрим большое количество полезных примеров, код которых может быть изменен таким образом, чтобы максимально соответствовать нуждам разработчика.

Динамический HTML основан на возможности изменения свойств объектов. Практически каждому элементу HTML соответствует объект, который создается в браузере при загрузке HTML-страницы или другого документа, например, XML-документа. Каждый объект имеет свои свойства и методы. Конкретная реализация объектов зависит от типа и версии браузера, в который загружен исходный Web-документ, поэтому возможности динамического HTML в сильной степени зависят от браузера. Доступ к свойствам и методам объектов осуществляется при помощи языков сценариев. Для работы с объектами, создаваемыми в клиентских приложениях-браузерах, используются языки JavaScript и VBScript. Мы не будем останавливаться на детальном изучении этих языков. Устройство программ-сценариев, созданных при помощи этих языков и используемых в примерах, довольно просто, поэтому примеры понятны и без предварительной подготовки.

Помимо сценариев динамический HTML основывается также на таблицах стилей CSS и особенностях того или иного браузера.

DHTML не является каким-либо стандартом. Это некий (нефиксированный раз и навсегда) набор средств, позволяющий сделать клиентский Web-документ интерактивным, живым и динамичным. DHTML — это скорее рекламный термин, вошедший в обиход с момента создания четвертых версий браузеров Internet Explorer и Netscape Navigator. Как правило, набор

программных средств, используемых в динамическом HTML, включает в себя следующее:

- язык HTML 4.0;
- стили CSS (Cascading Style Sheets — каскадные листы стилей);
- объектную модель документа DOM;
- языки сценариев (JavaScript и/или VBScript).

Объектная модель документа дополняется набором факультативных свойств и методов объектов, свойственных данному виду браузера. Реализация CSS в различных браузерах также имеет отличия (табл. 1.2).

Таблица 1.2. *Отличия браузеров при работе с DHTML*

Браузер Netscape Navigator 4.*x*	**Унифицированные стандарты DHTML**	**Браузер Internet Explorer 4.*x***
JSS (листы стилей JavaScript) Layers (слои — позиционирование и свойства блоков элементов)	CSS1 CSS2 (управление свойствами отдельных элементов) CSS Positioning (видимость элементов и позиционирование элементов) JavaScript	Visual Filters (применение визуальных эффектов для работы с текстом и графикой) Dynamic CSS (управление свойствами положения и видимости элементов страницы)

Листы стилей CSS (Cascading Style Sheets — каскадные листы стилей) используются для задания способов отображения HTML-элементов. Листы стилей используются также для задания способов отображения любых XML-элементов.

Внимание!

Отметим, что большая часть примеров требуют наличия браузера ***Internet Explorer*** версии 5.5 и выше или ***Netscape Navigator*** *6.2* и выше. Некоторые примеры могут работать с браузерами ***Internet Explorer*** *4* и ***Netscape Navigator*** *4*.

Положение элемента

При помощи свойства листов стилей `position` можно задать относительное или абсолютное положение элемента на странице в окне браузера.

Относительное положение *position:relative*

Свойство `position:relative` используется для задания положения элемента по отношению к текущему его положению (которое было бы, если бы не

было указано свойство position:relative). Пример работы с этим свойством приведен в листинге 1.29.

Листинг 1.29. Свойство position:relative

```
h1
{
position:relative;
left:10;
}
```

Данный пример задает стиль отображения элемента h1, т. е. элемента заголовка первого уровня, который описывается при помощи тэга <h1>. Заметим, что в классическом HTML не имеет значения, какие буквы мы используем — строчные или заглавные. Однако при работе с XML и XHTML следует тщательно следить за тем, чтобы правильно соблюдать регистр букв в написании элементов. В примере установлен сдвиг элемента вправо от первоначального неформатированного положения на десять пикселов (левый край сдвинут на 10 пикселов).

Чтобы задать абсолютное положение элемента, используем свойство position:absolute (листинг 1.30).

Листинг 1.30. Свойство position:absolute

```
h1
{
position:absolute;
left:10;
}
```

В качестве примеров относительного (рис. 1.27) и абсолютного (рис. 1.28) позиционирования элементов приведем файлы prel.html (листинг 1.31) и pabs.html (листинг 1.32). После сдвига влево на 20 пикселов заголовок вышел за границу окна браузера.

Листинг 1.31. Файл prel.html

```
<html>
<head>
<style>
h1.example1
{
position:relative;
left:40;
}
```

```
h1.example2
{
position:relative;
left:-20;
}
</style>
</head>
<body>
<h1>Заголовок первого уровня</h1>
<h1 class="example1">Заголовок первого уровня со сдвигом +40</h1>
<h1 class="example2">Заголовок первого уровня со сдвигом -20</h1>
<p>
Относительное позиционирование задает сдвиг относительно исходного
неизмененного положения элемента.
</p>
<p>
"left:40" прибавить 40 пикселов к положению левой границы.
</p>
<p>
"left:-20" отнять 20 пикселов от положения левой границы.
</p>
</body>
</html>
```

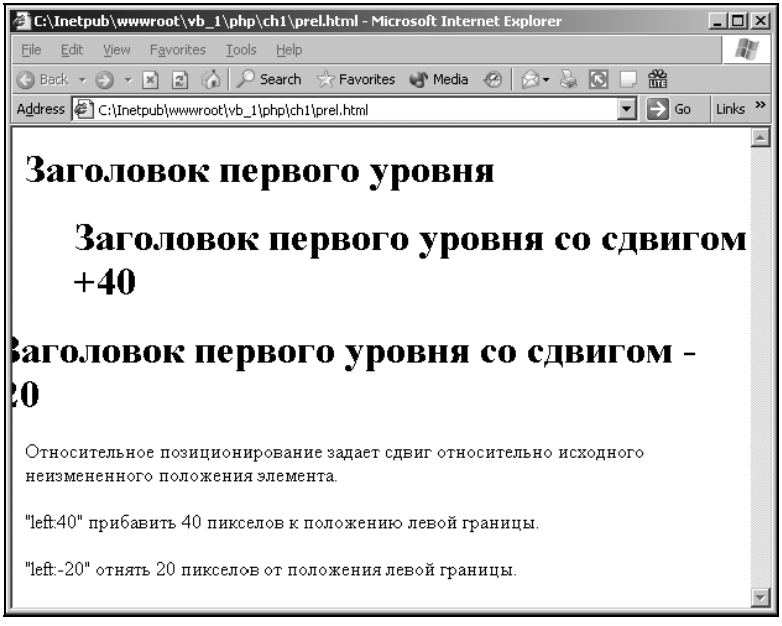

Рис. 1.27. Относительное смещение элементов

Листинг 1.32. Файл pabs.html

```
<html>
<head>
<style>
h1.x
{
position:absolute;
left:120;
top:150;
}
</style>
</head>
<body>
<h1 class="x">Заголовок первого уровня</h1>
<p>
Абсолютное позиционирование позволяет разместить элемент в любом месте
странички.
</p>
<p>
Заголовок смещен на 120 пикселов вправо и на 150 пикселов вниз.
</p>
</body>
</html>
```

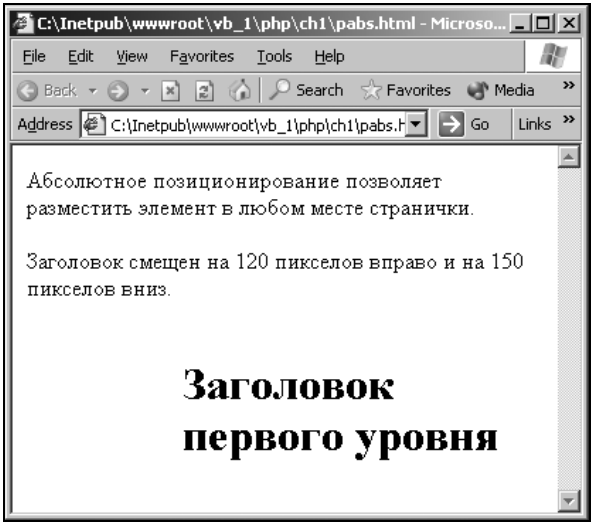

Рис. 1.28. Абсолютное позиционирование элемента

Видимость объектов

Свойство `visibility` позволяет указать, будет данный элемент видимым или нет. Допускается использование двух значений:

- `visibility:visible`
- `visibility:hidden`

Свойство `visibility` применяется так:

```
h1
{
visibility:visible;
}
```

или:

```
h1
{
visibility:hidden;
}
```

В примере в файле visibility.html свойство видимости второго заголовка задано так, что он не будет виден на HTML-страничке (листинг 1.33 и рис. 1.29).

Листинг 1.33. Файл visibility.html

```
<html>
<head>
<style>
h1.1
{
visibility:visible;
}
h1.2
{
visibility:hidden;
}
</style>
</head>
<body>
<h1 class="1">Первый заголовок</h1>
<h1 class="2">Второй заголовок</h1>
<p>Второй заголовок невидим.</p>
</body>
</html>
```

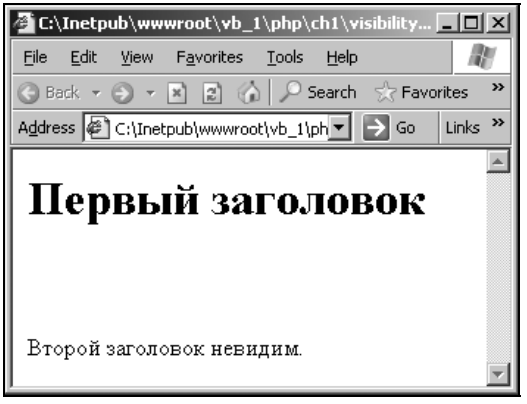

Рис. 1.29. Второй заголовок будет невидим

Свойство *z-index*

Элементы (объекты) Web-страницы можно упорядочить, расположив их "один над одним" ("один за одним"). Для этого элементы нумеруются при помощи свойства `z-index`. Чем больше значение свойства `z-index`, тем больший приоритет имеет элемент. Все значения этого свойства должны быть целыми.

Свойство `z-index` устанавливается следующим образом:

```
h1
{
z-index:1;
}
h2
{
z-index:2;
}
```

Пример работы со свойством `z-index` приведен в листинге 1.34.

Листинг 1.34. Файл zindex.html

```
<html>
<head>
<style>
img.x
{
position:absolute;
left:0;
```

```
top:0;
z-index:-1;
}
</style>
</head>
<body>
<h1>Заголовок</h1>
<img class="x" src="pic1.jpg">
<p>По умолчанию задается значение свойства z-index, равное 0.
Свойство Z-index, равное -1, имеет меньший приоритет, чем то, что задано
по умолчанию.</p>
</body>
</html>
```

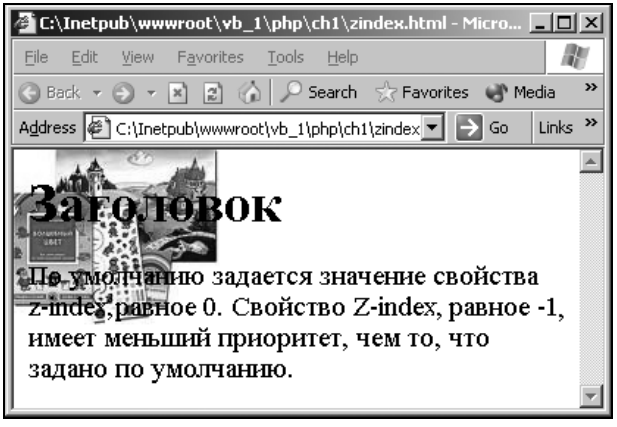

Рис. 1.30. Свойство `z-index` элементов

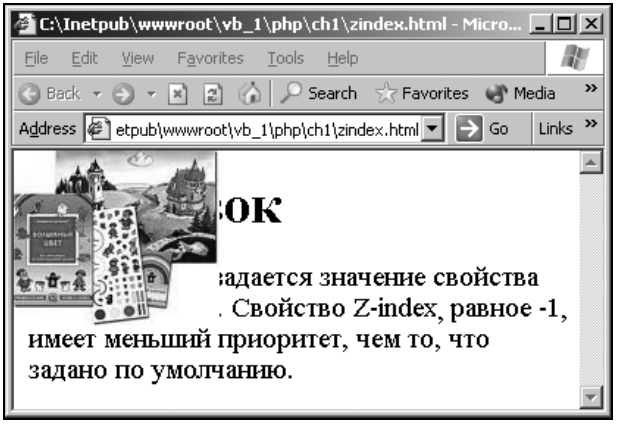

Рис. 1.31. Приоритет рисунка выше, чем приоритет текста

На рис. 1.30 мы видим, что изображение расположено на заднем плане, а текст — поверх него. Так произошло потому, что значение `z-index` для рисунка меньше, чем для текста. По умолчанию задается параметр `z-index`, равный 0. Чем больше `z-index`, тем "ближе к зрителю" расположен элемент в окне браузера. Значения свойства `z-index` не обязаны быть последовательными числами, но они должны быть обязательно целыми.

Если изменить приоритет и установить значение `z-index` равным 1, как показано ниже, то рисунок окажется на переднем плане и заслонит собой текст (рис. 1.31).

Объектная модель документа и DHTML

Благодаря существованию объектной модели документа разработчик HTML-страниц имеет возможность обращаться к свойствам и методам объектов, создаваемым на основе полученного браузером HTML-документа. Чтобы обратиться к элементу, можно указать его идентификатор. Это универсальный подход, который работает всегда, когда необходимо обратиться к тому или иному объекту. Прежде чем мы сможем обратиться к объекту по имени, необходимо задать его идентификатор (или имя). Это можно сделать, если использовать атрибут `id`, значение которого будет именем (идентификатором) соответствующего объекта. Например:

```
<html>
<body>
<h1 id="nazvanie">Заголовок</h1>
<script type="text/javascript">
header.style.color="red"
</script>
</body>
</html>
```

Если загрузить этот фрагмент кода в браузер, то будет создан объект заголовка первого уровня, идентификатором которого станет `nazvanie`. Скрипт, который расположен в этом HTML-коде, приведет к тому, что цвет заголовка изменится на красный. Скрипт может быть полезен, если требуется заменить рисунок другим рисунком, например:

```
<html>
<body>
<img id="image" src="s1.gif" width="32" height="32">
<script type="text/javascript">
image.src="s2.jpg"
</script>
```

```
Замена рисунка при помощи скрипта.</p>
</body>
</html>
```

Здесь исходный рисунок s1.gif будет заменен рисунком s2.gif.

Языки сценариев предоставляют механизм, при помощи которого мы имеем возможность обращаться к различным объектам, возникающим при загрузке страницы в браузер. В табл. 1.3 приведены основные объекты, которые существуют всегда, вне зависимости от того, какой код был получен браузером в клиентском документе.

Таблица 1.3. *Основные объекты в модели DOM*

Объект	Что содержит	Описание
`window`	Методы Свойства	Объект верхнего уровня в иерархии DHTML DOM. Объект содержит информацию об окнах и фреймах
`document`	Методы Свойства Коллекции	Объект представляет собой документ и используется для обращения к элементам, входящим в состав документа
`navigator`	Свойства	Этот объект содержит информацию о пользовательском браузере
`event`	Свойства Коллекции	Объект `event` содержит информацию о произошедшем событии

Обработка событий в DHTML

Обработка событий — это процесс выполнения тех или иных функций после того, как возникает определенное событие. Существует перечень событий, причем набор событий может меняться от одного объекта к другому, т. е. каждый объект может обладать набором событий, отличным от другого объекта. С событием можно связать ту или иную функцию или блок функций, причем при наступлении события эта функция (или блок функций) будет автоматически выполнена. Это называется *обработкой событий*. После того, как событие обработано в одном объекте, оно автоматически (если не задано обратное) переходит к объекту, который является родительским объектом по отношению к данному объекту. В родительском объекте также может быть задана функция для обработки этого события. Событие снова будет обработано в родительском объекте, и затем будет передано старшему объекту в цепочке иерархии объектов. Процесс перехода события от одного объекта к старшему родительскому объекту называется *"всплытием"* события. Всплывающее событие будет обрабатываться каждым объектом по це-

почке всплытия, если заданы соответствующие функции обработки для этого события.

Рассмотрим несколько примеров работы с событиями. В первом примере элемент будет изменять цвет при помещении на нем указателя мыши (рис. 1.33) (событие `onmouseover`). После того как курсор будет убран с элемента (событие `onmouseout`), цвет элемента возвращается к прежнему (рис. 1.32). HTML-код приведен в листинге 1.35.

Листинг 1.35. Файл onmouse.html

```
<html>
<body>
<h1 onmouseover="style.color='yellow'"
onmouseout="style.color='black'">
Поместите мышку на тексте</h1>
</body>
</html>
```

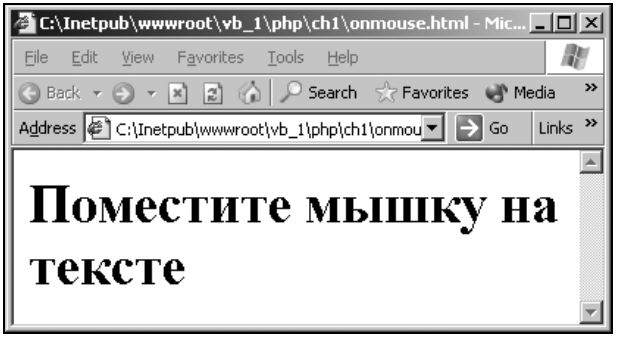

Рис. 1.32. Исходный текст

Рис. 1.33. Цвет текста поменялся

В следующем примере после двойного щелчка мышью на текстовом элементе происходит запись нового текста (рис. 1.34, 1.35). HTML-код приведен в листинге 1.36.

Листинг 1.36. Файл ondblclick.html

```
<html>
<head>
<script type="text/javascript">
function gettip(txt)
{
tip.innerHTML='Это дополнительный текст, который появился после двойного
щелчка'
}
</script>
</head>
<body>
<p>Щелкните следующий текст двойным щелчком мышки:</p>
<table>
<tr>
<th ondblclick="gettip()" valign="top">Щелкните меня.</th>
<th id="tip"> </th>
</tr>
</table>
</body>
</html>
```

Исходный текст показан на рис. 1.34. После двойного щелчка появился дополнительный текст (рис. 1.35).

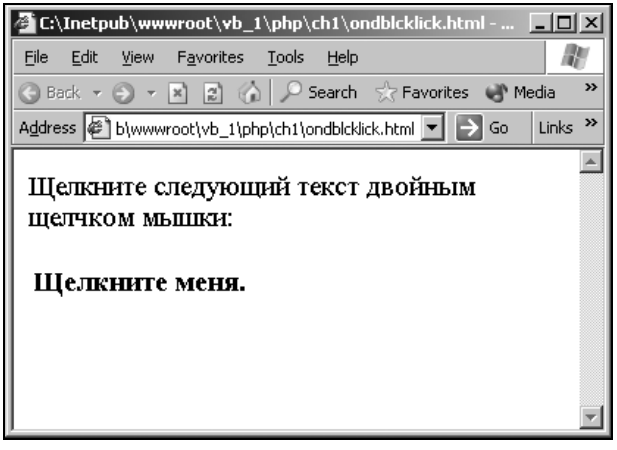

Рис. 1.34. Исходный текст

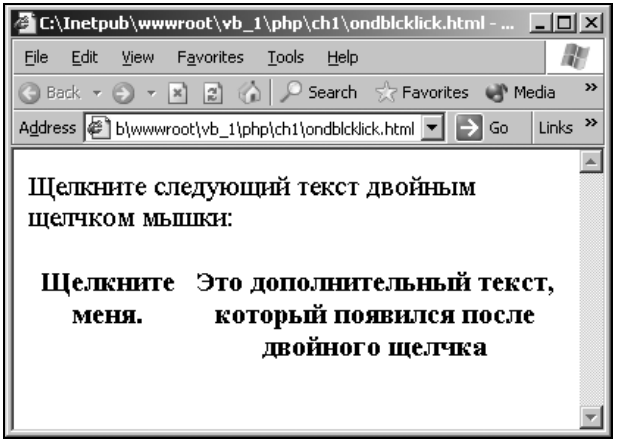

Рис. 1.35. Вид после двойного щелчка

В табл. 1.4 приведен список событий и их описания.

Таблица 1.4. *События в DHTML*

Событие	Описание
onabort	Пользователь прекратил загрузку страницы
onblur	Пользователь покидает объект
onchange	Пользователь изменяет значение объекта
onclick	Пользователь щелкает объект
ondblclick	Пользователь производит двойной щелчок мышью над объектом
onfocus	Пользователь устанавливает фокус на объекте (объект становится активным)
onkeydown	Пользователь нажимает клавишу клавиатуры (движение вниз)
onkeypress	Пользователь нажимает клавишу клавиатуры (полный цикл нажатия)
onkeyup	Пользователь отпускает клавишу клавиатуры (движение вверх)
onload	Завершение загрузки страницы
onmousedown	Пользователь нажимает кнопку мыши
onmousemove	Курсор движется над объектом
onmouseover	Курсор находится над объектом
onmouseout	Курсор покидает объект
onmouseup	Пользователь отпускает нажатую кнопку мыши
onreset	Пользователь очищает поля формы

Таблица 1.4 *(окончание)*

Событие	Описание
onselect	Пользователь выбирает часть содержимого страницы
onsubmit	Пользователь посылает форму серверу
onunload	Пользователь загружает страницу из окна браузера

Рецепты для работы с объектами DHTML

Далее мы рассмотрим некоторые основные методы работы с объектами. Эти примеры могут служить базовыми рецептами, на основе которых читатель сможет сам создавать свои собственные программы. Примеры тесно связаны со свойствами основных объектов DHTML.

Объект *window*

Поскольку этот объект содержит информацию об окнах и фреймах, то он позволяет изменять значения многих основных свойств. Большинство примеров работает с браузером Internet Explorer версии 4.7 и Netscape Navigator 6.2.

Первый пример показывает, как можно организовать работу с окнами-предупреждениями. Окно-предупреждение содержит текст-предупреждение и кнопку, нажатие которой закроет предупреждающее окно. Код примера содержится в файле alert.html (листинг 1.37 и рис. 1.36).

Листинг 1.37. Файл alert.html

```
<html>
<head>
<script type="text/javascript">
function salert()
{
alert("Я окно с предупреждением! -- Не шали!!")
}
</script>
</head>
<body>
<form>
<input type="button" onclick="salert()"
value="Показать окно-предупреждение">
</form>
</body>
</html>
```

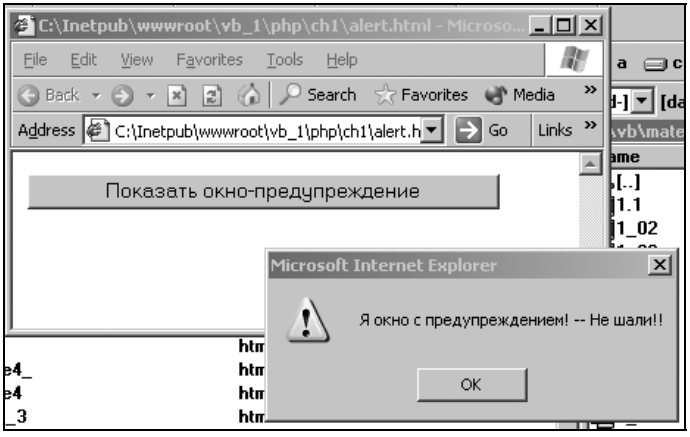

Рис. 1.36. Окно-предупреждение

Объект `window` позволяет получить свойства пользовательского экрана (рис. 1.37), как показано в файле screen.html (листинг 1.38).

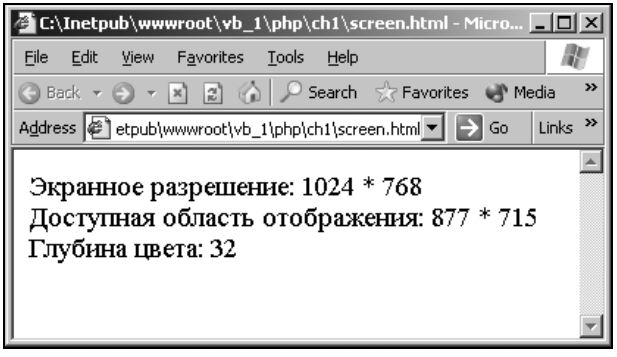

Рис. 1.37. Параметры пользовательского экрана

Листинг 1.38. Файл screen.html

```
<html>
<body>
<script type="text/javascript">
document.write("Экранное разрешение: ")
document.write(screen.width + " * ")
document.write(screen.height + "<br>")
document.write("Доступная область отображения: ")
document.write(screen.availWidth + " * ")
document.write(screen.availHeight + "<br>")
```

```
document.write("Глубина цвета: ")
document.write(screen.colorDepth + "<br>")
</script>
</body>
</html>
```

Открыть новое окно браузера можно при помощи приведенного ниже кода:

```
<html>
<head>
<script language=javascript>
function openWindow()
{
window.open("http://www.yandex.ru/")
}
</script>
</head>
<body>
<input type=button value="Open Window" onclick="openWindow()">
</form>
</body>
</html>
```

Можно задать фиксированные размеры окна, которые будут установлены после изменения размеров окна, для этого следует использовать такой фрагмент кода:

```
top.resizeTo(700,500)
```

Здесь в качестве фактических параметров указываются конечные ширина и высота окна соответственно.

Приведем пример работы с окном-приглашением, в котором пользователь может ввести произвольный текст. В отличие от форм, которые выводятся в составе основного окна браузера, это окно является самостоятельным, а не входит в браузер. После того как пользователь ввел в окно текстовую информацию, эта информация может быть обработана при помощи языка сценария и, например, отображена в основном окне браузера (рис. 1.38). Код приводится в файле prompt.html (листинг 1.39).

Листинг 1.39. Файл prompt.html

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

```
<script type="text/javascript">
var name = prompt("Введите сюда текст.","")
if (name != null && name != "")
{
document.write("Ваш ввод: " + name)
}
</script>
</body>
</html>
```

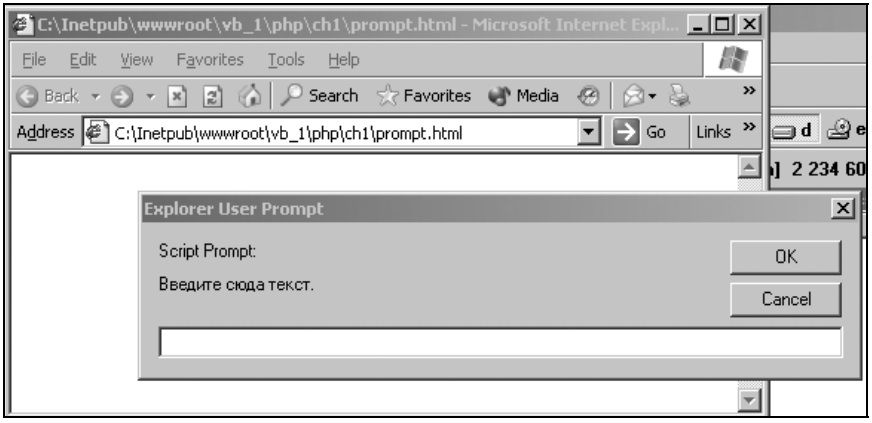

Рис. 1.38. Окно-приглашение

В табл. 1.5 приведены свойства объекта `window`, табл. 1.6 содержит описания методов объекта `window`.

Таблица 1.5. *Свойства объекта* `window`

Свойство	Назначение
`clientInformation`	Объект навигатора
`Closed`	Логическое значение, сообщает о том, закрыто окно или нет
`Complete`	Логическое значение, сообщает о том, завершен ли процесс загрузки окна
`DataTransfer`	Возвращает заранее установленные форматы, которые могут быть использованы в процессе передачи данных
`DefaultStatus`	Задает сообщение, которое будет использоваться по умолчанию для отображения в панели состояния
`DialogArguments`	Аргумент (можно читать и задавать свои) диалогового окна, создаваемого при помощи методов `showModalDialog()` и `showModelessDialog()`

Таблица 1.5 *(окончание)*

Свойство	Назначение
DialogHeight	Высота диалогового окна
DialogLeft	Левая координата диалогового окна
DialogTop	Верхняя координата диалогового окна
DialogWidth	Ширина диалогового окна
Document	Объект документа document
Event	Объект event
External	Хост, откуда получена Web-страница
History	Ранее посещенные URL
Length	Количество фреймов в окне
Location	Текущий URL
Name	Имя окна
Navigator	Объект навигатора
offscreenBuffering	Используется ли двойная буферизация, т. е. отображаются ли объекты "за экраном" перед тем, как будут показаны на экране
Opener	Объект, который открыл текущее окно
Parent	Родительский объект для текущего фрейма
ReturnValue	Возвращаемое окном-предупреждением значение
Screen	Объект с информацией о пользовательском экране
ScreenLeft	Координаты (только чтение) левого верхнего края пользовательского экрана
ScreenTop	Координаты верха пользовательского экрана
Self	Текущий объект кона
Status	Текст (чтение и запись) панели состояния
Top	Самый верхний объект окна

Таблица 1.6. *Методы объекта* `window`

Метод	Описание
alert()	Выводит окно-предупреждение с сообщением и кнопкой **OK**

Таблица 1.6 *(продолжение)*

Метод	Описание
`attachEvent()`	Связывает с событием функцию. Функция вызывается всякий раз при наступлении этого события
`Blur()`	Окно теряет фокус
`clearInterval()`	Освобождает установленные методом `setInterval()` значения
`clearTimeout()`	Освобождает установленные методом `setTimeout()` значения
`close()`	Закрывает текущее окно
`confirm()`	Выводит окно-подтверждение с сообщением и кнопками **Cancel** и **OK**
`detachEvent()`	Убирает функцию из объекта
`execScript()`	Позволяет выполнить скрипт
`focus()`	Устанавливает фокус на заданное окно
`MoveBy()`	Меняет местоположение окна по отношению к текущему положению
`MoveTo()`	Перемещает окно в заданное положение
`Navigate()`	Загружает указанный URL
`open()`	Открывает новое окно
`print()`	Выводит документ в окне на печать
`Prompt()`	Выводит окно-приглашение с сообщением и полем для ввода текста
`ResizeBy()`	Изменяет размер окна на заданные значения
`ResizeTo()`	Устанавливает размер окна в соответствии с указанными значениями
`Scroll()`	Этот метод устарел, вместо него используется метод `ScrollTo()`
`ScrollBy()`	Прокручивает окно с текущей позиции на заданную величину
`ScrollTo()`	Прокручивает окно к указанной позиции
`setInterval()`	Устанавливает интервал для выполнения функции
`setTimeout()`	Устанавливает задержку для выполнения функции
`ShowHelp()`	Выводит указанный файл подсказки
`showModalDialog()`	Выводит модальное диалоговое окно с указанным HTML-документом

Таблица 1.6 *(окончание)*

Метод	Описание
`showModelessDialog()`	Выводит немодальное диалоговое окно с указанным HTML-документом

Объект *navigator*

Объект `navigator` содержит информацию о браузере, который работает у пользователя. В качестве примера рассмотрим код, который выводит информацию о браузере непосредственно в окно браузера (рис. 1.39). Текст программы содержится в листинге 1.40.

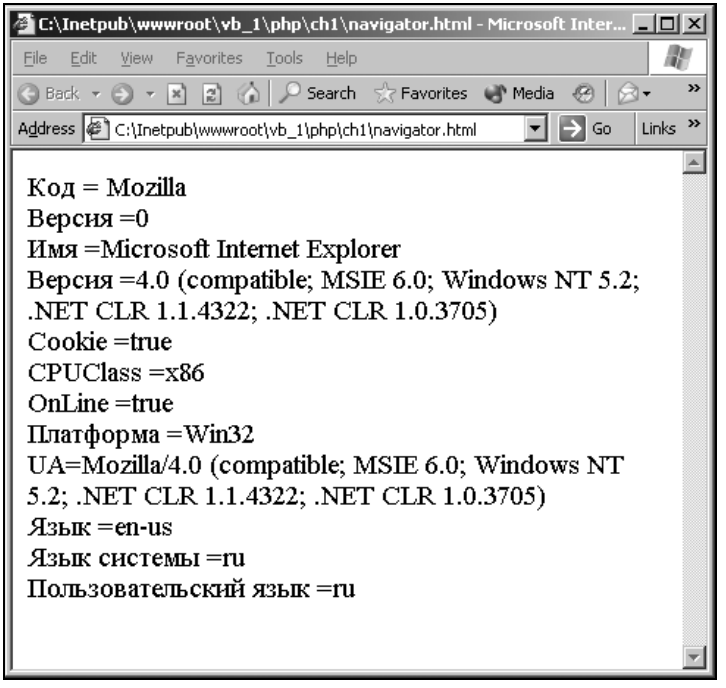

Рис. 1.39. Параметры пользовательского браузера

Листинг 1.40. Файл navigator.html

```
<html>
<body>
<script type="text/javascript">
var x = navigator
```

```
document.write("Код = " + x.appCodeName)
document.write("<br>")
document.write("Версия =" + x.appMinorVersion)
document.write("<br>")
document.write("Имя =" + x.appName)
document.write("<br>")
document.write("Версия =" + x.appVersion)
document.write("<br>")
document.write("Cookie =" + x.cookieEnabled)
document.write("<br>")
document.write("CPUClass =" + x.cpuClass)
document.write("<br>")
document.write("OnLine =" + x.onLine)
document.write("<br>")
document.write("Платформа =" + x.platform)
document.write("<br>")
document.write("UA=" + x.userAgent)
document.write("<br>")
document.write("Язык =" + x.browserLanguage)
document.write("<br>")
document.write("Язык системы =" + x.systemLanguage)
document.write("<br>")
document.write("Пользовательский язык =" + x.userLanguage)
</script>
</body>
</html>
```

В табл. 1.7 содержатся описания свойств объекта `navigator`.

Таблица 1.7. *Свойства объекта* `navigator`

Свойство	Описание
`appCodeName`	Возвращает кодированное имя браузера
`appMinorVersion`	Номер расширения версии браузера
`appName`	Имя браузера
`appVersion`	Платформа и версия браузера
`browserLanguage`	Текущий язык браузера
`cookieEnabled`	Поддерживаются ли cookie
`cpuClass`	Класс процессора
`OnLine`	Состояние системы (online или offline)

Таблица 1.7 *(окончание)*

Свойство	Описание
`Platform`	Платформа, на которой работает браузер
`systemLanguage`	Язык системы по умолчанию
`userAgent`	Пользовательский агент HTTP
`userLanguage`	Текущий язык пользователя

Объект `navigator` часто используется для того, чтобы определить, в каком окружении работает пользователь — это требуется для предоставления пользователю клиентского кода, который будет правильно работать в существующем клиентском окружении.

Объект *event*

Объект `event` содержит информацию о происходящем событии. Существуют различные типы событий. К наиболее часто используемым событиям относятся события мыши (нажатие кнопки мыши, помещение курсора над изображением того или иного объекта, перемещение курсора, нажатие клавиш клавиатуры, окончание загрузки документа, прекращение загрузки документа и т. п.). Примеры этого раздела будут работать с браузером Internet Explorer 5.5 и выше.

В первом примере мы рассмотрим код, с помощью которого можно определить, какая кнопка мыши была нажата (рис. 1.40). Этот код содержится в файле buttonm.html (листинг 1.41).

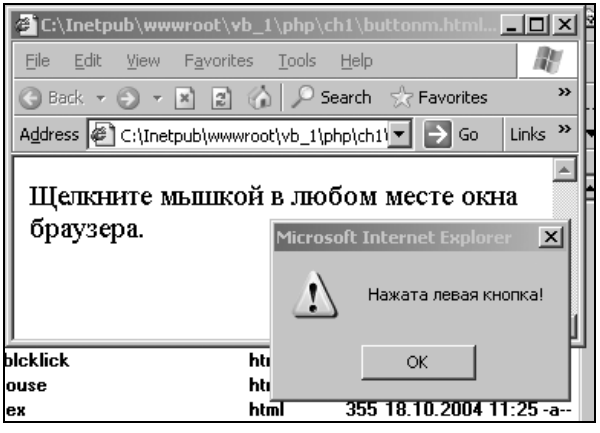

Рис. 1.40. Сообщение о кнопке мыши, которая была нажата

Листинг 1.41. Файл buttonm.html

```
<html>
<head>
<script type="text/javascript">
function whichButton()
{
if (event.button==1)
{
alert("Нажата левая кнопка!")
}
else
{
alert("Нажата правая кнопка!")
}
}
</script>
</head>
<body onmousedown="whichButton()">
<p>
Щелкните мышкой в любом месте окна браузера.
</p>
</body>
</html>
```

Для определения того, какая кнопка мыши была нажата, используется свойство `button` объекта `event`. Значению 1 соответствует левая кнопка. Иное значение будет соответствовать правой кнопке. Проверка того, какая была нажата кнопка, осуществляется функцией `whichButton()`, которая назначена функцией обработки события `onmousedown` (нажатие кнопки мыши, причем любой кнопки).

Часто бывает полезным определить координаты местоположения курсора. Объект `event` позволяет сделать это без всяких проблем. Пример приведен в файле coordxy.html (листинг 1.42), на рис. 1.41 показаны полученные координаты курсора.

Листинг 1.42. Файл coordxy.html

```
<html>
<head>
<script type="text/javascript">
function coordinates()
{
x=event.clientX
y=event.clientY
```

```
alert("X=" + x + " Y=" + y)
}
</script>
</head>
<body onmousedown="coordinates()">
<p>
Щелкните мышкой, и в окне-предупреждении появятся координаты места
щелчка.
</p>
</body>
</html>
```

Чтобы получить координаты события, используются свойства `clientX` и `clientY` (см. рис. 1.41).

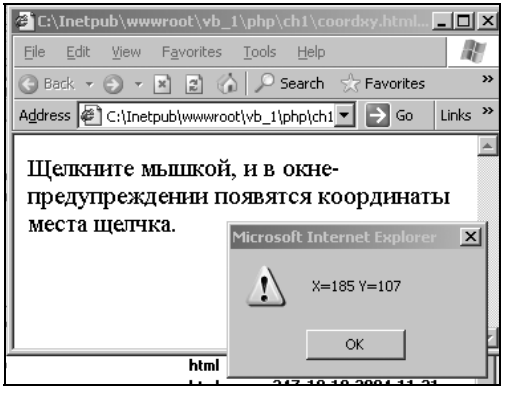

Рис. 1.41. Координаты курсора

Может оказаться весьма полезным свойство `keycode`, с помощью которого можно определить код нажатой на клавиатуре клавиши в кодировке Unicode (рис. 1.42). Пример содержится в файле key.html (листинг 1.43).

Листинг 1.43. Файл key.html

```
<html>
<head>
<script type="text/javascript">
function whichButton()
{
alert(event.keyCode)
}
</script>
```

```
</head>
<body onkeyup="whichButton()">
<p>
Нажмите клавишу на клавиатуре. В окне-предупреждении появится ее код.
</p>
</body>
</html>
```

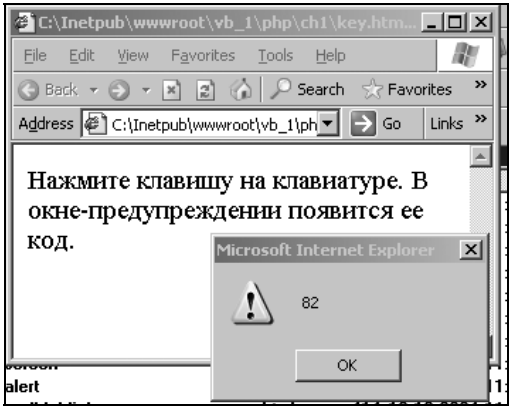

Рис. 1.42. Код клавиши на клавиатуре

Объект `event` позволяет определить тип произошедшего события (рис. 1.43), для этого используется свойство `type` (листинг 1.44).

Листинг 1.44. Файл type.html

```
<html>
<head>
<script type="text/javascript">
function whichType()
{
alert(event.type)
}
</script>
</head>
<body onmousedown="whichType()">
<p>
тип события (щелкните в окне мышкой).
</p>
</body>
</html>
```

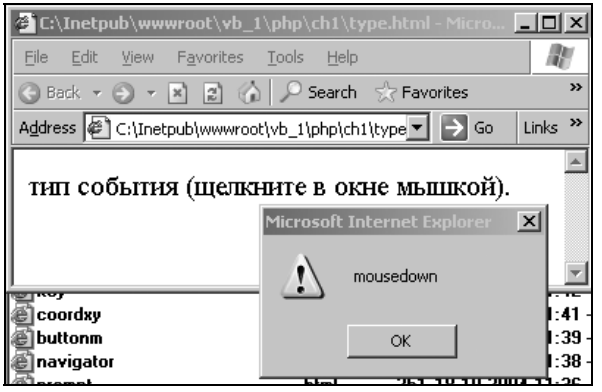

Рис. 1.43. Тип вызванного события

С помощью объекта `event` можно узнать, какой элемент был щелкнут мышью в документе. Пример приведен в файле element.html (листинг 1.45) и показан на рис. 1.44.

Листинг 1.45. Файл element.html

```
<html>
<head>
<script type="text/javascript">
function whichElement()
{
var tname
tname=event.srcElement.tagName
alert("Щелчок был на элементе " + tname + ".")
}
</script>
</head>
<body onmousedown="whichElement()">
<p>
Щелкните в окне и узнайте, где произведен щелчок.
</p>
<h3>Заголовок</h3>
<p>Текст простой.</p>
<p><img border="0" src="mik.jpg" width="40" height="55"
alt="рисунок"></p>
</body>
</html>
```

При помощи объекта `event` можно осуществить проверку того, была ли нажата та или иная дополнительная клавиша-модификатор. Например, про-

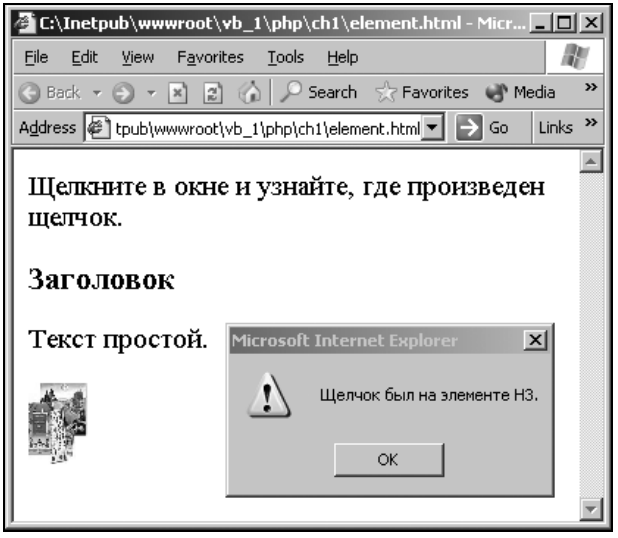

Рис. 1.44. Сообщение о элементе, на котором был произведен щелчок мышью

верим, была ли нажата во время щелчка мышью клавиша `<Shift>`. Код приведен в файле shift.html (листинг 1.46), результат показан на рис. 1.45.

Листинг 1.46. Файл shift.html

```
<html>
<head>
<script type="text/javascript">
function isKeyPressed()
{
if (event.shiftKey==1)
{
alert("Клавиша shift была нажата!")
}
else
{
alert("Клавиша shift не была нажата!")
}
}
</script>
</head>
<body onmousedown="isKeyPressed()">
<p>Щелкните в окне и узнайте, нажата ли была клавиша Shift. </p>
</body>
</html>
```

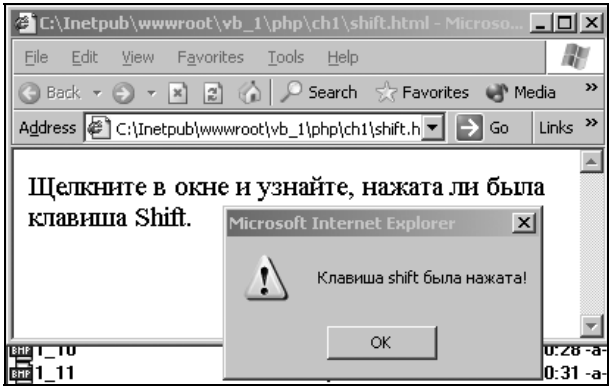

Рис. 1.45. Клавиша <Shift> была нажата

Чтобы узнать, была ли нажата клавиша <Shift>, мы воспользовались свойством `shiftKey`.

В табл. 1.8 содержатся описания свойств объекта `event`.

Таблица 1.8. *Свойства объекта* `event`

Свойство	Описание
`AltKey`	Определяет, нажата или нет клавиша <Alt>
`Button`	Определяет, какая нажата кнопка мыши
`cancelBubble`	Отменяет или оставляет в силе всплытие текущего объекта по иерархии объектов документа
`clientX`	X-координата курсора
`clientY`	Y-координата курсора
`ctrlKey`	Определяет, была ли нажата клавиша <Ctrl>
`dataFld`	Возвращает столбец, измененный при помощи события `oncellchange`
`fromElement`	Возвращает объект, из которого ушел курсор
`keyCode`	Задает или получает значение клавиши в кодировке Unicode
`offsetX`	Горизонтальная координата курсора
`offsetY`	Вертикальная координата курсора
`propertyName`	Имя измененного свойства
`Reason`	Причина завершения работы с объектом данных
`recordset`	Возвращает запись `recordset`

Таблица 1.8 (окончание)

Свойство	Описание
Repeat	Возвращает значение, соответствующее повторению (или не повторению) события
returnValue	Задает значение, которое будет возвращено событием
screenX	X-координата курсора на экране
screenY	Y-координата курсора на экране
ShiftKey	Определяет, была ли нажата клавиша <Shift>
srcElement	Объект, вызвавший событие
srcFilter	Объект фильтра, вызвавший событие onfilterchange
srcUrn	Универсальное имя ресурса, связанного с объектом behavior, вызвавшего событие
toElement	Объект, к которому перешел курсор
type	Тип события
x	X-координата курсора
y	Y-координата курсора

Объект *collection*

Объект collection — это *коллекция* элементов документа. С помощью этого объекта можно, например, подсчитать количество элементов того или иного типа, которые расположены на текущей страничке, а также осуществить доступ к любому элементу странички. Примеры этого раздела могут работать с браузерами Internet Explorer 4.7 и Netscape Navigator 6.2. Чтобы осуществить доступ к элементу, напишем код, который сохраним в файле collection.html (листинг 1.47).

Листинг 1.47. Файл collection.html

```
<html>
<body>
Осуществляем доступ к элементам форм.
<form name="Form1">
Имя: <input type="text" size="20">
</form>
<form name="Form2">
Возраст: <input type="text" size="3">
</form>
```

```
<p><b>По номеру в коллекции (указываем индекс в коллекции):</b></p>
<script type="text/javascript">
document.write("<p>Первая форма: ")
document.write(document.forms[0].name + "</p>")
document.write("<p>Вторая форма: ")
document.write(document.forms[1].name + "</p>")
</script>
<p><b>По имени формы:</b></p>
<script type="text/javascript">
document.write("<p>Имя первой формы: ")
document.write(document.forms("Form1").name + "</p>")
document.write("<p>имя второй формы: ")
document.write(document.forms("Form2").name + "</p>")
</script>
</body>
</html>
```

В этом примере используется два способа обращения: по имени элемента и по индексу в массиве `collection`. Результат показан на рис. 1.46.

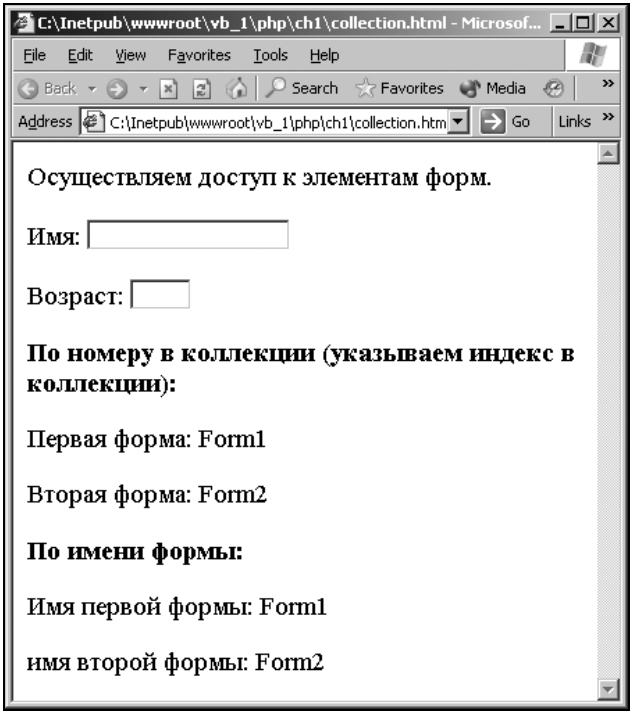

Рис. 1.46. Обращение к элементам

Второй пример посвящен решению задачи пересчета всех элементов заданного типа (листинг 1.48). Результат работы примера показан на рис. 1.47.

Листинг 1.48. Файл colel.html

```
<html>
<head>
</head>
<body>
<form name="Form1">
Имя: <input type="text" size="20">
</form>
<form name="Form2">
Возраст: <input type="text" size="3">
</form>
<script type="text/javascript">
txt="На этой страничке: " + document.forms.length + " форм."
document.write(txt)
</script>
</body>
</html>
```

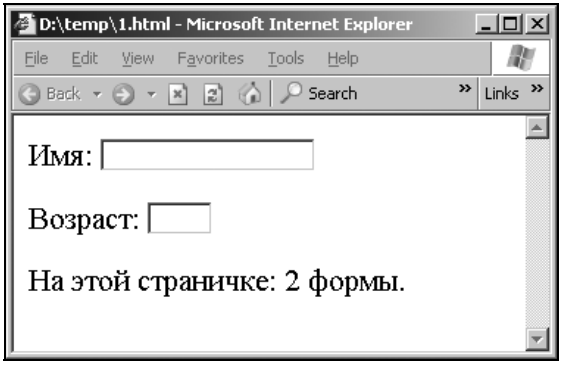

Рис. 1.47. Количество элементов в коллекции

В приведенном выше файле количество форм коллекции содержится в свойстве `length`.

В табл. 1.9 приведены свойства и методы объекта коллекции.

Таблица 1.9. Свойства и методы объекта `collection`

Свойства и методы	Описание
`Length` — свойство	Количество элементов в коллекции
`Item(index)` — метод	Возвращает элемент по индексу
`namedItem(name)` — метод	Возвращает элемент по имени

Объект *document*

Объект `document` соответствует текущему документу, загруженному в браузер.

В файле anchors.html определяется количество ссылок в документе (листинг 1.49, рис. 1.48).

Листинг 1.49. Файл anchors.html

```
<html>
<body>
<a name="firstA">первая ссылка</a><br>
<a name="secondA">Вторая ссылка</a><br>
<a name="thirdA">Третья ссылка</a><br>
В этом документе следующее число ссылок:
<script type="text/javascript">
document.write(document.anchors.length)
</script>
</body>
</html>
```

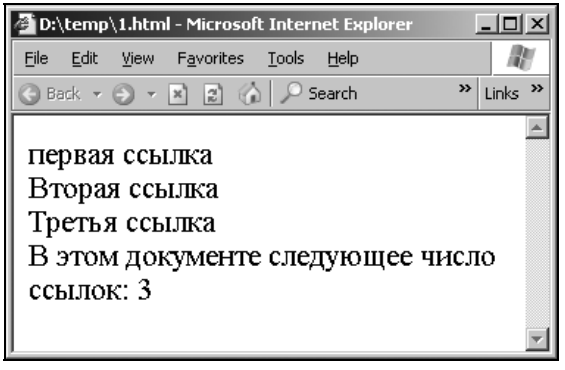

Рис. 1.48. Количество ссылок в документе

Объект `document` позволяет обращаться к входящим в его состав дочерним элементам. В следующем примере мы определяем, над каким элементом произошел щелчок мышью (листинг 1.50 и рис. 1.49).

Листинг 1.50. Файл byifd.html

```
<html>
<head>
<script type="text/javascript">
function getElement()
{
var x=document.getElementById("myHeader")
alert("I am a " + x.tagName + " element")
}
</script>
</head>
<body>
<h1 id="myHeader" onclick="getElement()">Щелкни на мне и узнай - что я за
элемент!</h1>
</body>
</html>
```

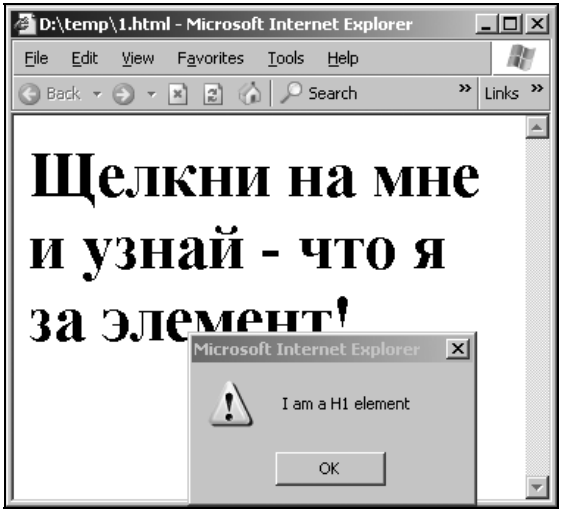

Рис. 1.49. Обращение к элементу по его имени (идентификатору)

Метод `write` позволяет осуществлять вывод HTML-кода, который отображается в браузере обычным способом. В качестве примера приведем код файла write.html (листинг 1.51). После загрузки в браузере появляется кнопка **Соз-**

дать новый документ (рис. 1.50). После нажатия этой кнопки в окне браузера отображается новый документ (рис. 1.51), код которого выглядит следующим образом:

```
<h3>Новый документ</h3>
```

Создан новый документ с помощью метода `write`.

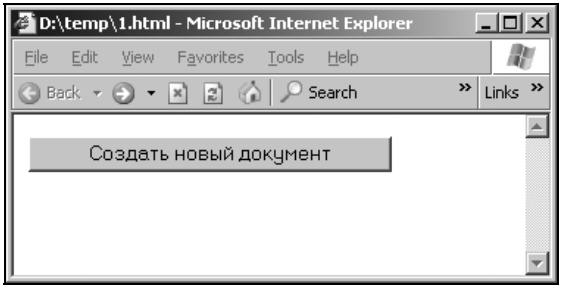

Рис. 1.50. После загрузки документа write.html

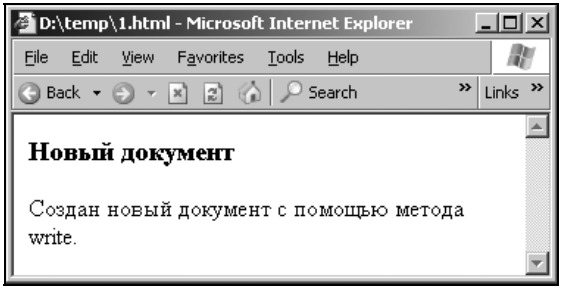

Рис. 1.51. После нажатия кнопки **Создать новый документ**

Листинг 1.51. Файл write.html

```
<html>
<head>
<script type="text/javascript">
function docOpen()
{
document.open()
document.write("<h3>Новый документ</h3> ")
document.write("Создан новый документ с помощью метода write.")
}
</script>
</head>
```

```
<body>
<form>
<input type="button" onclick="docOpen()" value="Создать новый документ">
</form>
</body>
</html>
```

Свойства объекта `document` приведены в табл. 1.10.

Таблица 1.10. *Свойства объекта* `document`

Свойства	Описание
`Anchors`	Коллекция всех ссылок (элементов `<a>`)
`Applets`	Коллекция всех апплетов
`Body`	Тело элемента, помещенного в элемент `frameset`
`Cookie`	Возвращает cookies для данного документа
`Domain`	Возвращает доменное имя полученного текущего документа
`Forms`	Возвращает коллекцию всех форм документа
`Images`	Возвращает коллекцию всех рисунков документа
`Links`	Возвращает коллекцию всех ссылок, для которых задан атрибут `href`
`Referrer`	Возвращает URL, с которой была вызвана текущая страничка
`Title`	Возвращает или устанавливает заголовок документа
`URL`	Возвращает URL текущего документа

Методы объекта `document` перечислены в табл. 1.11.

Таблица 1.11. *Методы объекта* `document`

Метод	Описание
`close()`	Закрывает документ, который был открыт при помощи метода `document.open()`
`getElementById(ID)`	Возвращает элемент по идентификатору `ID`
`getElementsByName(name)`	Возвращает коллекцию всех элементов с заданным именем
`open()`	Открывает новый документ
`write(text)`	Пишет текст в документ
`writeln(text)`	Пишет строку текста в документ

ГЛАВА 2

Основы работы с PHP

Предыдущая глава была посвящена рассмотрению возможностей создания динамических страниц, которые выполняются в браузере клиента. Настоящая глава знакомит с основами создания серверных сценариев. Серверный сценарий выполняется перед тем, как HTTP-сервер отправит документ в ответ на запрос клиента. Для создания серверных сценариев можно использовать различные языки, но преимуществом языка PHP является весьма богатый ассортимент возможностей в сочетании с простотой. Кроме того, язык PHP распространяется как бесплатное программное обеспечение с открытым кодом. Исполняемые файлы и исходный код можно получить на сайте **http://www.php.net**.

Установка довольно проста. В системе Windows можно воспользоваться программой-инсталлятором, или же просто распаковать архив в любую папку, например в папку C:\php. В администраторе вашего HTTP-сервера нужно указать, что файлы с расширением php будут обрабатываться PHP-интерпретатором (exe-модулем CGI — php.exe или ISAPI-библиотекой DLL), конкретное название файла может меняться от версии к версии.

Создадим простейший PHP-файл, который назовем test.php (листинг 2.1).

Листинг 2.1. Файл test.php

```
<?
echo "<h1>"Это язык php 5.</h1>";
?>
```

Если вы работаете в Windows, то щелкните на имени файла мышью. Если PHP зарегистрирован в системе, то наш файл автоматически будет выполнен. Если система не узнает файл по расширению, то расширение нужно зарегистрировать. Простейший способ регистрации — выбрать в диалоговом окне, появившемся после щелчка мышью на файле first.php, программу

php.exe (в каталоге, куда был установлен php), и отметить, что эта программа будет вызываться всякий раз при обращении к файлам с расширением php (установить флажок слева внизу). При запуске файла first.php будет появляться консольное окно, в котором отображается результат работы программы, записанной в файле first.php.

Разместим файл first.php в каталоге выполняемых файлов. В браузере укажем адрес **http://localhost/[ваш_путь]/test.php**. В окне браузера появится результат работы программы, записанной в файле test.php (рис. 2.1).

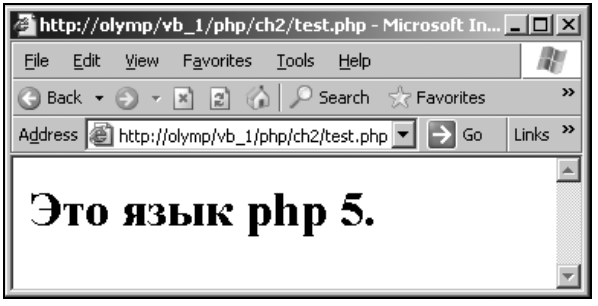

Рис. 2.1. Результат работы PHP

Язык PHP предоставляет большое количество возможностей для создания полнофункциональных серверных Web-приложений. Среди основных свойств можно назвать такие: обработка информации, введенной пользователем, посылка почты, работа с базами данных, работа с изображениями, работа с сетью, работа с объектами CORBA. В этой главе мы познакомимся с основными возможностями, без которых Web-программирование просто немыслимо, а в последующих главах рассмотрим некоторые возможности языка PHP более подробно.

Обработка HTML-форм

HTML-формы представляют собой механизм передачи серверу данных от пользователя клиентского браузера. Рассмотрим HTML-файл, содержащий форму form.html (листинг 2.2), страница с этой формой показана на рис. 2.2.

Листинг 2.2. Файл form.html

```
<html>
   <head>
      <title>
      Пример формы. Передаем пользовательскую информацию серверу.
      </title>
```

```
<head>
   <body>
      <form name="formname" method=
                  post action="http://localhost/vb_1/php/ch2/form.php">
         <P>Введите текстовую информацию: <input type="text" name="text1">
         <P><input type="submit" value="Submit the form to the server">
      </form>
   </body>
</html>
```

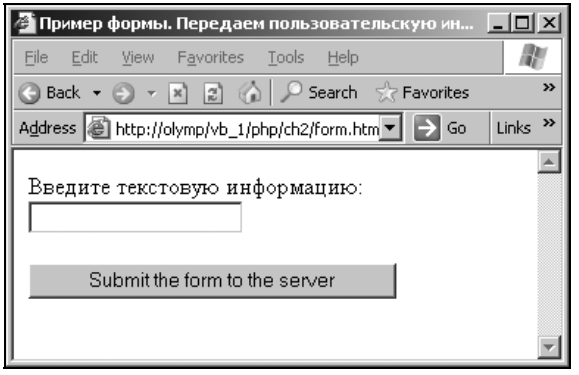

Рис. 2.2. Заполняем форму

Элемент формы начинается открывающим тэгом `<form>`. В качестве свойств элемента в тэге указаны:

- имя элемента `formname`;
- метод передачи данных `post`;
- свойство `action`, его значение — это адрес файла с программой, которая будет обрабатывать данные. В нашем случае — form.php. Адрес этот может быть как относительный (относительно местоположения данного HTML-файла), так и абсолютный.

Внутри элемента формы описано несколько компонентов. Важным для нас являются элементы `input`. Каждый элемент `input` имеет имя и значение. Именно эти пары имен и значений будут переданы серверу в виде `name=value`. В нашем случае переменной с именем `text1` будет соответствовать строковое значение, которое пользователь введет в соответствующее поле. Различные типы элементов `input`, описанные в языке HTML, соответствуют разным графическим элементам, отображаемым в окне браузера. Кнопка отправки формы имеет тип `"submit"`. Все пары `name`—`value`, содержащиеся в форме, будут переданы программе, указанной в качестве значения свойства `action`. В файле form.php обратиться к этим значениям можно,

указав имя соответствующей им переменной. Все переменные в PHP начинаются со знака $, поэтому наша переменная будет иметь вид $test1.

Напишем простой PHP-файл form.php для обработки данной формы. Мы уже видели, что текст PHP программы начинается со знаков <? и заканчивается ?> (листинг 2.3).

Листинг 2.3. Файл form.php

```
<?
Echo "<html><head><title>Forms</title></head><body>";
Echo "<h1>Пользователь ввел такую строку:</h1>";
Echo $_REQUEST["text1"];
Echo "</body></html>";
?>
```

Файл сохраняем в каталоге C:\Inetpub\wwwroot\vb_1\ch2. После нажатия кнопки отправки формы видим, что пользовательские данные возвращены сервером (рис. 2.3).

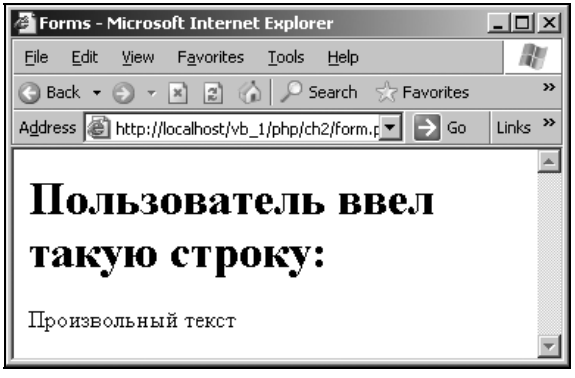

Рис. 2.3. Пользовательские данные были успешно обработаны

Приведенная в этом примере страница не имеет никакого дизайна. Коммерческие Web-приложения должны быть оформлены с учетом возможностей HTML-форматирования и удобства пользователей при работе.

Простейший счетчик посещений

Простейший *счетчик посещений* можно создать с использованием файловой системы. При посещении странички счетчик посещений каждый раз увеличивается на единицу. Для записи количества посещений используем произвольный файл count. Чтение из файла производится при помощи функции fread(), запись в файл осуществляется функцией fwrite(). Перед тем как

использовать эти функции, необходимо "открыть" файл, создать "файловый поток", "ресурс файла" при помощи функции `fopen()`.

Разместим наш простейший счетчик в файле count.php (листинг 2.4).

Листинг 2.4. Файл count.php

```
<?php
// Читаем файл=============================================================
// Открываем ресурс файла для чтения
$file = fopen("count", "r");
// Читаем
$str = fread($file, 1024);
// Закрываем ресурс файла
fclose($file);
// Увеличиваем счетчик на единицу и пишем в файл ======================
// Открываем ресурс файла для записи
$file = fopen("count", "w");
// Увеличиваем счетчик
$str++;
echo "<b>Страницу посетили ".$str." раз.</b>";
// Записываем измененный счетчик в файл
fwrite($file, $str);
// Закрываем ресурс файла
fclose($file);
?>
```

При каждом обращении к странице count.php счетчик увеличивается на единицу (рис. 2.4).

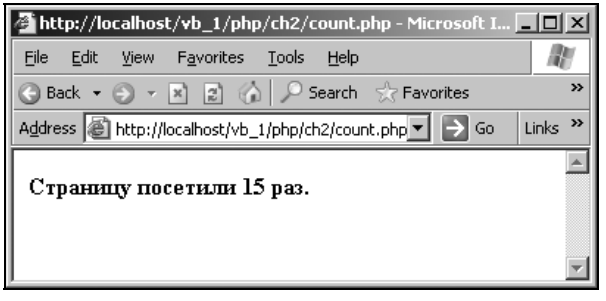

Рис. 2.4. Каждый запрос к странице увеличивает значение счетчика на 1

В качестве параметров функция `fopen()` принимает строку с именем файла и строку с ключом, который указывает права доступа и способ обращения к файлу.

Существуют следующие ключи:

- `'r'` — только чтение, указатель в начале файла;
- `'r+'` — чтение и запись, указатель в начале файла;
- `'w'` — только запись, длина файла устанавливается равной 0, указатель в начале файла, если файла нет, то предпринимается попытка его создания;
- `'w+'` — чтение и запись, длина файла устанавливается равной 0, указатель в начале файла, если файла нет, то предпринимается попытка его создания;
- `'a'` — только чтение, указатель в конце файла, если файла нет, то предпринимается попытка его создания;
- `'a+'` — чтение и запись, указатель в конце файла, если файла нет, то предпринимается попытка его создания;
- `'x'` — создание файла и только запись;
- `'x+'` — создание файла для записи и чтения.

Второй аргумент функции `fread()` — длина считываемых данных, второй аргумент функции `fwrite()` — записываемые данные, в качестве первого аргументы обе функции принимают идентификатор ресурса файла.

Отправка писем с помощью PHP

Иногда возникает необходимость обеспечить отправку писем с Web-сервера. Для этого в PHP существуют специальные функции. Если разработчик сам является администратором своего собственного сервера, то перед тем, как создавать приложение, которое посылает почту с использованием функции `mail()`, необходимо сконфигурировать PHP надлежащим образом. В файле php.ini необходимо указать адрес (имя) SMTP-сервера. Таким сервером может быть сервер, установленный на том же самом компьютере, что и Web-сервер, или этим сервером может быть другой SMTP-сервер, через который открыт доступ для посылки писем. Фрагмент файла php.ini будет выглядеть примерно так, как показано в листинге 2.5.

Листинг 2.5. Фрагмент файла php.ini

```
;;;;;;;;;;;;;;;;;;;
[mail function]
; For Win32 only.
SMTP= mail.yourname.com; for Win32 only
```

```
; For Win32 only.
sendmail_from= yourname@yourhost.com; for Win32 only

; For Unix only.  You may supply arguments as well (default: 'sendmail -t -i').
;sendmail_path=
```

В качестве примера создадим простой PHP-скрипт, который будет отправлять сообщение (листинг 2.6).

Листинг 2.6. Файл mail.php

```
<?php

function whois($domain, $server="www.geektools.com")
{
$date = date("F j, Y");
$time = date("g:i a");
$ip=$_SERVER['REMOTE_ADDR'];
$string1="Date of visit: ".$date."\r\n Time of visit: ".$time."\r\n
Visited from: ".$ip;
$fp = fsockopen ($server, 43, &$errnr, &$errstr) or die("$errno:
$errstr");
fputs($fp, "$domain\n");
$string2="";
while (!feof($fp))

$string2.= fgets($fp, 2048);
fclose($fp);
//echo $string2;
$m =  mail("vbolymp@rol.ru", "File mail.php has been visdited recently",
$string1.$string2,   "From: name@host.com\r\n"  ."Reply-To:
name@address.com\r\n"  ."X-Mailer: PHP/". phpversion());
}
whois($_SERVER['REMOTE_ADDR']);

?>
<html>
<head>
<title>Mail
</title>
<body>
<h1>Отправка почты сервером.</h1>
<P>При этом пользователь ничего не подозревает...
</body>
</html>
```

Немного поясним приведенный пример. Отправка почтового сообщения осуществляется функцией `mail()`, аргументами которой являются:

- первый аргумент: адрес получателя;
- второй аргумент — это строка — поле `subject`, в котором указывается тема сообщения;
- третьим аргументом является тело письма (в нашем случае это конкатенация двух строк с именами `$string1` и `$string2`);
- четвертым аргументом служит строка, содержащая дополнительные заголовки. В этой строке имеются управляющие последовательности символов `\r` и `\n` — переход на новую строку, а также PHP-функция определения версии языка PHP.

Стандартная функция PHP `mail()` помещена внутрь пользовательской функции `whois($ip)`, аргументом которой является строка, содержащая имя или IP-адрес хоста. Весь скрипт осуществляет определение времени обращения к нему и IP-адреса, с которого к нему обратились (встроенная переменная окружения `$REMOTE_ADDR`). Полученный IP-адрес используется для того, чтобы получить информацию о хосте, который связан с данным IP. Для этого происходит обращение к базе, расположенной на сервере **www.geektools.com** с портом 43. Для соединения используется протокол TCP, соединение осуществляется при помощи стандартной функции языка PHP `fsockopen()`. Работу с сетью мы рассмотрим немного позже. Функции связи с сервером и отсылки информации по указанному адресу организованы в самостоятельную пользовательскую функцию `whois()`.

Если пользователь зайдет на ваш сайт (предположим, что файл mail.php расположен на сайте **http://yoursite.com/cgi-bin/mail.php)**, то PHP тут же запросит базу данных whois и сгенерирует электронное сообщение, отправив его по адресу, указанному в качестве первого аргумента функции `mail()`. По этому адресу вскоре придет сообщение, содержащее примерно такую информацию в теле письма:

```
Date of visit: September 30, 2004
 Time of visit: 5:22 pm
 Visited from: 212.46.197.136 GeekTools Whois Proxy v5.0.4 Ready.
Final results obtained from whois.ripe.net.
Results:
% This is the RIPE Whois secondary server.
% The objects are in RPSL format.
%
% Rights restricted by copyright.
% See http://www.ripe.net/db/copyright.html

inetnum:   212.46.192.0 - 212.46.218.255
netname:   CITYLINERU
```

```
descr:    Cityline offers dial-up and leased line access to Internet
country:   RU
admin-c:   CTNC1-RIPE
tech-c:    CTNC1-RIPE
status:    ASSIGNED PA
notify:    noc@cityline.ru
remarks:   Please send abuse notification to abuse@cityline.ru
mnt-by:    CITYLINERU-MNT
changed:   mih@cityline.ru 20040422
source:    RIPE

route:    212.46.197.0/24
descr:    Cityline offers dial-up and leased line access to Internet
descr:    for StPete citizens
origin:    AS8755
remarks:   Please send abuse notification to abuse@cl.spb.ru
remarks:   please direct all queries to support@cl.spb.ru
notify:    noc@cityline.ru
mnt-by:    CITYLINERU-MNT
changed:   fil@cl.spb.ru 20001124
source:    RIPE

role:     cityline.ru NOC
address:   Cityline Ltd.
address:   1/2 str 6, Korobeinikov per.
address:   119034 Moscow
address:   Russia
phone:    +7 095 3633312
fax-no:    +7 095 2487848
e-mail:    noc@cityline.ru
admin-c:   MR14224-RIPE
tech-c:    MR14224-RIPE
nic-hdl:   CTNC1-RIPE
notify:    dbm@cityline.ru
mnt-by:    CITYLINERU-MNT
changed:   mih@cityline.ru 20020219
source:    RIPE
```

Отсюда мы можем узнать, что посетитель нашей странички зашел к нам с машины с IP-адресом 212.46.197.136, который принадлежит компании Cityline. Письмо содержит также другую полезную информацию.

Отметим, что вне зависимости от того, какие действия будут произведены на серверной стороне, клиентский браузер может получить любую HTML-информацию, пользователь даже не будет подозревать о том, что информа-

ция о нем была каким-то образом обработана и передана средствами электронной почты (рис. 2.5).

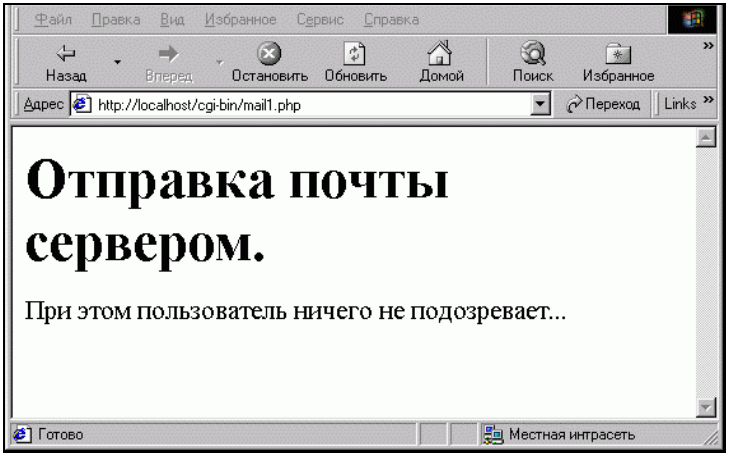

Рис 2.5. Пользователь не знает, что происходит на сервере

Передача файла серверу

Чтобы передать файл серверу вместе с HTML-формой, необходимо использовать элемент `<input type=file>`. HTML-форма должна содержать описание атрибута `enctype="multipart/form-data"`. Скрипт, который будет обрабатывать эту форму, также должен быть специальным образом подготовлен (листинг 2.7).

Листинг 2.7. Файл fileupload.html

```
<html>
<head>
<title></title>
</head>
<body bgcolor="#FEF8C0">

<h2 align="center"><font color="#EC2F97">N</font><font
color="#83E2D9">e</font><font color="#223344">w</font>
<form enctype="multipart/form-data" action="fileupload.php"
method="post">

<tr><td>Choose file here.
<input type=file name="userfile">
</td><td></td> </tr>
```

```
Введите имя файла для сохранения
<input type=text name="fname">
<input type=submit  value="Upload your file">
    </td>
  </tr>
</table>
</form>
</html>
```

Файл обрабатывающего скрипта fileupload.php приведен в листинге 2.8.

Листинг 2.8. Файл fileupload.php

```
<?php
$date = date("Y-m-d");
$time = date("G:i:s");
$ip=$REMOTE_ADDR;
if (is_uploaded_file($userfile)) {
copy($userfile, "./images/".$fname); $p=1;}
else { echo "Mistake found"; }
?>
```

Сессии

Сессии представляют собой механизм, с помощью которого происходит идентификация пользователя. Они позволяют сохранять информацию, связанную с пользователем, даже тогда, когда пользователь перемещается от одного URL к другому. В листинге 2.9 приведен пример простейшего скрипта, который увеличивает "счетчик посещений" на единицу при повторном посещении страницы session1.php пользователем.

Листинг 2.9. Файл session1.php

```
<?php
session_start();
if (!session_is_registered('count')) {
   session_register('count');
   $count = 1;
} else {
   $count++;
}
?>

<p>
Привет, уважаемый посетитель, на этой странице Вы были <?php echo $count;
?> раз.
</p>
```

Чтобы этот скрипт правильно работал, необходимо в файле конфигурации php.ini включить опцию `register globals`:

```
register_globals = On
```

О правилах хорошего тона

Существует множество полезных рекомендаций на тему о том, как правильно программировать. Этой теме посвящено немалое количество источников. Всегда оказывается полезным отделять представление от данных. Верстка HTML-странички должна быть отделена от функциональности. При работе с языком PHP это также актуально.

Отделить функциональность основного скрипта от HTML-представления можно довольно просто, если использовать функции вставки, например функцию `require_once()`. В нашем примере HTML-код и вставляемые в него переменные из фрагментов-скриптов располагаются в файле require1.php (листинг 2.10). Основная функциональность которая может быть сколь угодно сложной (в нашем же примере приводится лишь описание принципа разделения), содержится в самостоятельном файле required1.php (листинг 2.11). Файл required1.php вставляется в файл require1.php с помощью функции `require_once('required1.php')`.

Листинг 2.10. Файл require1.php

```
<?php
require_once('required1.php');
?>
<h1>Front-end</h1>
Этот файл содержит представление данных. Формирование данных, их
обработка отделены от представления -
это происходит в отдельном файле.<hr>
<p>Первая переменная: <?php echo $first; ?>.</p>
<p>Вторая переменная: <?php echo $second; ?>.</p>
<p>Третья переменная: <?php echo $third; ?>.</p>
```

Листинг 2.11. Файл required1.php

```
<?php
// Здесь располагается основная функциональность
// (во всяком случае должна быть расположена)
$first='строчка';
$second=6;
$third="<h3>Заголовок</h3>";
?>
```

Изображения в PHP

PHP предоставляет удобные средства для работы с изображениями. Файл image1.php (листинг 2.12) содержит пример отображения существующего рисунка, сохраненного в файле (рис. 2.6).

Листинг 2.12. Файл image1.php

```
<?php
// HTTP-заголовок
header("Content-type: image/png");
// Создаем изображение
$im = imagecreatefrompng("pic1.png");
// Отображаем рисунок
imagepng($im);
?>
```

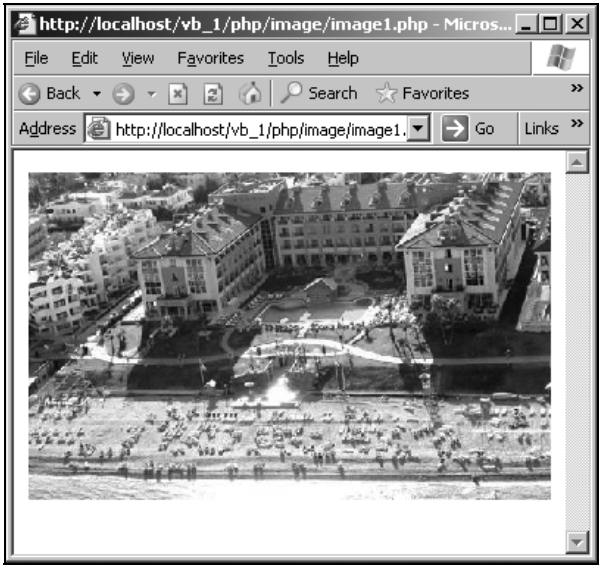

Рис. 2.6. Отображение существующего рисунка

Изменение размеров изображения и отображение в окне браузера в соответствии с указанными размерами показано на примере файла image2.php (листинг 2.13 и рис. 2.7).

Листинг 2.13. Файл image2.php

```
<?
// Посылаем заголовок
header("Content-type: image/jpeg");
```

```
// Вызываем функцию изменения размеров
resize_jpg("pic1.jpg",600,100);

function resize_jpg($img,$w,$h){
// Размеры исходного изображения
   $imagedata = getimagesize($img);
// Создание нового изображения
   $im2 = ImageCreateTrueColor($w,$h);
// Загрузка исходного изображения
   $image = ImageCreateFromJpeg($img);
// Переформатирование нового изображения с использованием старого
   imagecopyResampled ($im2, $image, 0, 0, 0, 0, $w, $h, $imagedata[0],
$imagedata[1]);
// Отображение полученного изображения
   ImageJpeg($im2);
}
?>
```

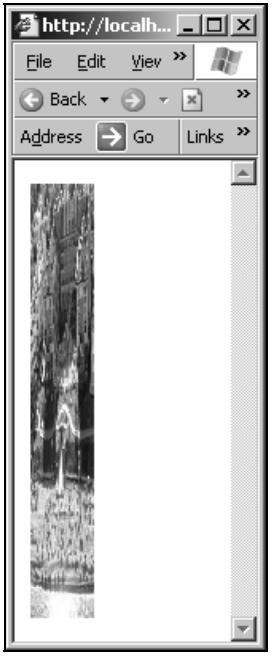

Рис. 2.7. Отображение существующего рисунка по заданным размерам

Полезно создать функцию, которая будет менять размер исходного изображения пропорционально размерам исходной картинки, так, чтобы результирующее изображение не превышало указанные размеры (листинг 2.14).

Листинг 2.14. Файл image3.php

```
<?
// Посылаем заголовок
header("Content-type: image/jpeg");
// Вызываем функцию изменения размеров
resize_jpg("pic1.jpg",600,100);

function resize_jpg($img,$w,$h){
// Размеры исходного изображения
   $imagedata = getimagesize($img);
// Масштабирование
   if ($w && ($imagedata[0] < $imagedata[1])) {
         $w = ($h / $imagedata[1]) * $imagedata[0];
   } else {
         $h = ($w / $imagedata[0]) * $imagedata[1];
   }
// Создание нового изображения
   $im2 = ImageCreateTrueColor($w,$h);
// Загрузка исходного изображения
   $image = ImageCreateFromJpeg($img);
// Переформатирование нового изображения с использованием старого
   imagecopyResampled ($im2, $image, 0, 0, 0, 0, $w, $h, $imagedata[0],
$imagedata[1]);
// Отображение полученного изображения
   ImageJpeg($im2);
}
?>
```

Следующий пример вставляет существующий рисунок с поворотом на заданный угол (листинг 2.15 и рис. 2.8).

Листинг 2.15. Файл image3a.php

```
<?
// Посылаем заголовок
header("Content-type: image/jpeg");
// Вызываем функцию изменения размеров
resize_jpg("pic1.jpg",600,100);

function resize_jpg($img,$w,$h){
// Размеры исходного изображения
   $imagedata = getimagesize($img);
```

```
// Масштабирование
   if ($w && ($imagedata[0] < $imagedata[1])) {
         $w = ($h / $imagedata[1]) * $imagedata[0];
   } else {
         $h = ($w / $imagedata[0]) * $imagedata[1];
   }
// Создание нового изображения
   $im2 = ImageCreateTrueColor($w,$h);
// Загрузка исходного изображения
   $image = ImageCreateFromJpeg($img);
// Переформатирование нового изображения с использованием старого
   imagecopyResampled ($im2, $image, 0, 0, 0, 0, $w, $h, $imagedata[0],
$imagedata[1]);
   $bg = imagecolorallocate($im2, 240, 240, 240);

    $im2 = imagerotate($im2, 10, $bg);

// Отображение полученного изображения
   ImageJpeg($im2);
}
?>
```

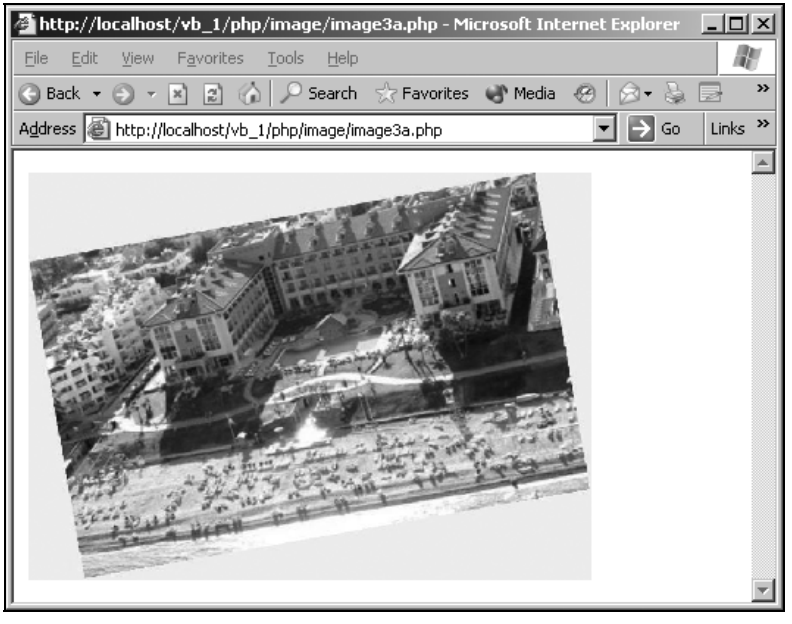

Рис. 2.8. Отображение существующего рисунка с поворотом

Еще один пример — масштабирование рисунка и вставка его в рисунок с заранее указанным размером (листинг 2.16 и рис. 2.9).

Листинг 2.16. Файл image4.php

```
<?
// Посылаем заголовок
header("Content-type: image/jpeg");
// Вызываем функцию изменения размеров
resize_jpg("pic1.jpg",1000);

function resize_jpg($img,$w){
// Размеры исходного изображения
   $imagedata = getimagesize($img);
// Масштабирование
$h=$w;
$w1=$w;
$h1=$h;
   if ($w && ($imagedata[0] < $imagedata[1])) {
         $w1 = ($h / $imagedata[1]) * $imagedata[0];
   } else {
         $h1 = ($w / $imagedata[0]) * $imagedata[1];
   }


// Создание нового изображения
   $im2 = ImageCreateTrueColor($w1,$h1);
   $im3 = ImageCreateTrueColor($w,$h);
// Загрузка исходного изображения
   $image = ImageCreateFromJpeg($img);
// Переформатирование нового изображения с использованием старого
   imagecopyResampled ($im2, $image, 0, 0, 0, 0, $w1, $h1, $imagedata[0],
$imagedata[1]);
   $bg = imagecolorallocate($im3, 240, 240, 240);
   imagefill($im3, 0,0, $bg);
//    $im2 = imagerotate($im2, 10, $bg);
 imagecopy ( $im3, $im2, 0, abs($h-$h1)/2., 0, 0, $w1, $h1);

// Отображение полученного изображения
   ImageJpeg($im3);
}
?>
```

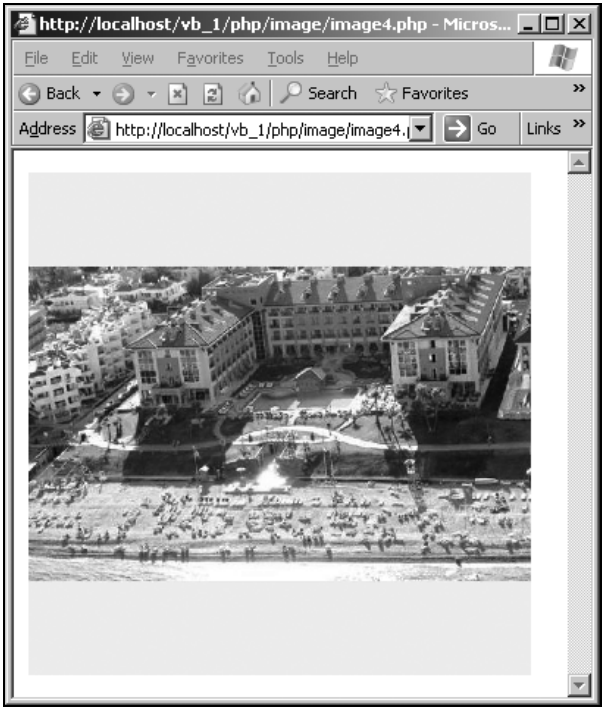

Рис. 2.9. Вставка существующего рисунка в другой рисунок с масштабированием

PHP располагает возможностями, позволяющими создавать PDF-документы (рис. 2.10), простейший пример содержится в листинге 2.17.

Листинг 2.17. Файл pdf3.php

```
<?php
Header("Content-type: application/pdf");
$cpdf = cpdf_open(0);
cpdf_page_init($cpdf, 1, 0, 595, 842, 1.0);
cpdf_add_outline($cpdf, 0, 0, 0, 1, "Page 1");
cpdf_begin_text($cpdf);
cpdf_set_font($cpdf, "Times-Roman", 70, "WinAnsiEncoding");
cpdf_set_text_rendering($cpdf, 1);
cpdf_text($cpdf, "Hello, my dear friend!", 90, 500);
cpdf_end_text($cpdf);
cpdf_stroke($cpdf);
cpdf_finalize_page($cpdf, 1);
cpdf_finalize($cpdf);
```

```
cpdf_output_buffer($cpdf);
cpdf_close($cpdf);
?>
```

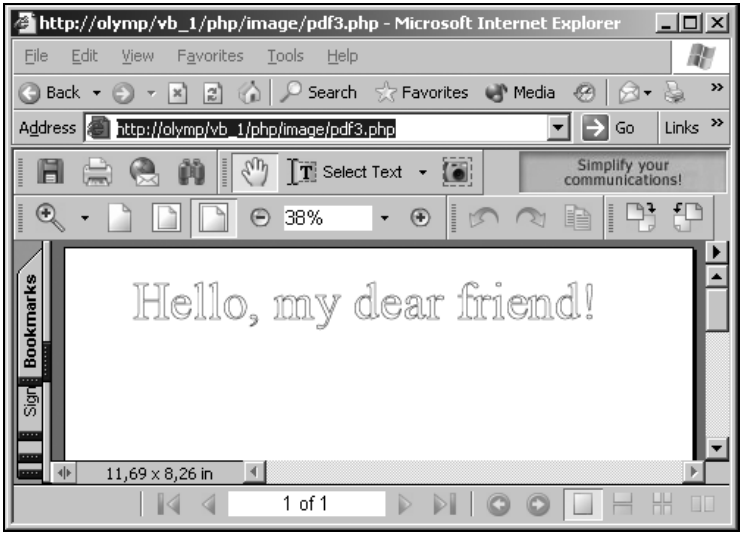

Рис. 2.10. Генерация PDF-документа

ГЛАВА 3

Основные понятия

Типы данных

PHP поддерживает 8 примитивных *типов данных*.

- Четыре скалярных типа:
  - `boolean`
  - `integer`
  - число с плавающей точкой (`float`)
  - `string`
- Два составных типа
  - `array`
  - `object`
- Два специальных типа
  - `resource`
  - `NULL`

Логический тип

Логический (булев) тип представлен двумя литералами: `TRUE` и `FALSE` (нечувствительны к регистру). При явном преобразовании к булеву типу используются конструкции `(bool)` или `(boolean)`. При этом к булеву значению `TRUE` приводятся все значения (включая ресурсы), кроме перечисленных ниже. К значению `FALSE` приводятся:

- `integer 0` (нуль);
- `float 0.0` (нуль);
- пустая строка и строка "0";

- массив с нулевым количеством элементов;
- объект с нулевым количеством элементов;
- специальный тип NULL.

Целые числа

Целые числа (элементы множества {..., -2, -1, 0, 1, 2, ...}) могут быть представлены в различном виде, например:

- `$a = 1234;` — 10-ричное число;
- `$a = -123;` — отрицательное число;
- `$a = 0123;` — 8-ричное число (эквивалентно 10-ричному 83);
- `$a = 0x1A;` — шестнадцатеричное число (эквивалентно 10-ричному 26).

Числа с плавающей точкой

Числа с плавающей точкой могут быть представлены в следующих формах:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

Формально вид числа с плавающей точкой задается следующим образом:

```
LNUM          [0-9]+
DNUM          ([0-9]*[\.]{LNUM}) | ({LNUM}[\.][0-9]*)
EXPONENT_DNUM ( ({LNUM} | {DNUM}) [eE][+-]? {LNUM})
```

Строки

Строка представляет собой последовательность символов, помещенных в кавычки. Кавычки могут быть как двойными ("), так и одинарными ('). Существует также специальный способ задания строк heredoc. Если в строке, записанной с помощью одинарных кавычек, встречается символ одиночной кавычки в последовательности прочих символов, то перед ним следует записать обратную косую черту (\).

Отметим, что переменные, входящие в состав строки, записанной при помощи одиночных кавычек, не заменяются своими значениями.

Переменные в составе строк, образованных с использованием двойных кавычек, заменяются своими значениями. Кроме этого, в строках с двойными кавычками можно использовать следующие escape-последовательности:

- `\n` — прогон строки (LF или 0x0A (10) в ASCII);
- `\r` — возврат каретки (CR или 0x0D (13) в ASCII);
- `\t` — горизонтальная табуляция (HT или 0x09 (9) в ASCII);
- `\\` — обратная наклонная черта (backslash);
- `\$` — знак доллара;
- `\"` — двойная кавычка;
- `\[0-7]{1,3}` — последовательность символов, совпадающая с регулярным выражением, символ в 8-ричной нотации;
- `\x[0-9A-Fa-f]{1,2}` — последовательность символов, совпадающая с регулярным выражением, символ в 16-ричной нотации.

Способ `heredoc` предполагает использование последовательности `'<<<'`, за которой следует идентификатор. За идентификатором записывается строка, завершающаяся этим же идентификатором, например:

```
$str = <<<EOD
Пример строки,
захватывающей несколько строчек,
с использованием синтаксиса heredoc.
EOD;
```

Массивы

Пример задания массива:

```
$a = array( 1 => 'one', 2 => 'two', 3 => 'three');
```

Здесь задается массив с тремя элементами `$a[1]`, `$a[2]`, `$a[3]` со значениями `'one'`, `'two'`, `'three'` соответственно. В качестве ключей использованы числа 1, 2, 3. В качестве ключей могут быть использованы также строки, например:

```
$a = array( 'color' => 'red'
        , 'taste' => 'sweet'
        , 'shape' => 'round'
        , 'name'  => 'apple');
```

Массив можно также задать путем присвоения значения его элементу:

```
$a['taste']='sweet';
```

или

```
$a[]='red';
```

Приведение типов

Чтобы привести значение к нужному типу, перед ним записывается имя нужного типа в скобках. Используются следующие имена типов:

- `(int)`, `(integer)` — приведение к типу `integer`;
- `(bool)`, `(boolean)` — приведение к типу `boolean`;
- `(float)`, `(double)`, `(real)` — приведение к типу `float`;
- `(string)` — приведение к типу `string`;
- `(array)` — приведение к типу `array`;
- `(object)` — приведение к типу `object`.

Приведение к типу *boolean*

Следующие значения приводятся к логическому `FALSE`:

- `boolean FALSE`;
- `integer 0` (нуль);
- `float 0.0` (нуль);
- пустая строка и строка "0";
- массив с нулевым количеством элементов;
- `object` с нулевым количеством элементов;
- специальный тип `NULL`.

Прочие значения приводятся к логическому `TRUE`.

Приведение к типу *integer*

Логическое `FALSE` приводится к нулю, `TRUE` — к единице. Числа с плавающей точкой округляются в сторону нуля.

Приведение строк к числам

Приведем примеры:

```
<?php
$foo = 1 + "10.5";                // float (11.5)
$foo = 1 + "-1.3e3";              // float (-1299)
$foo = 1 + "bob-1.3e3";           // integer (1)
$foo = 1 + "bob3";                // integer (1)
$foo = 1 + "10 Small Pigs";       // integer (11)
$foo = 4 + "10.2 Little Piggies"; // float (14.2)
$foo = "10.0 pigs " + 1;          // float (11)
$foo = "10.0 pigs " + 1.0;        // float (11)
?>
```

Переменные

Идентификатор переменной начинается со знака доллара `$`. *Имя переменной* начинается с буквы или символа подчеркивания, с последующими (в любом количестве) буквами, числами или символами подчеркивания:

```
'[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'
```

В листинге 3.1 приведен пример работы с переменными.

Листинг 3.1. Переменные

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var";      // Выводит "Bob, Joe"

$4site = 'not yet';     // Неправильно; начинается с числа
$_4site = 'not yet';    // Правильно; начинается с символа
                           Подчеркивания/underscore
$täyte = 'mansikka';    // Правильно; 'ä' это ASCII 228.
```

Константы

Константа — это идентификатор, которым обозначается простое значение. Константы определяются глобально, они видимы из любой части скрипта. Для задания константы используется функция `define()`. В константах могут содержаться только типы `boolean`, `string`, `integer`, `float`.

Отметим следующие отличия констант от переменных:

- перед именем константы нет знака доллара (`$`);
- константы могут быть определены только через использование функции `define()`, но не простым присвоением;
- константы могут быть определены, и доступ к ним может быть получен в любом месте, вне зависимости от правил области видимости переменных;
- константы не могут быть переопределены после своего определения;
- константы могут вычисляться только в скалярные значения.

Пример работы с константами приведен в листинге 3.2.

Листинг 3.2. Константы

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // Выводит "Hello world."
?>
```

Операторы

Операторы выполняют те или иные действия с одним или несколькими значениями или выражениями, результатом действия оператора является новое значение. По количеству значений, используемых оператором, операторы разделяются на три группы:

- *унарные* операторы (например, оператор отрицания `!` и оператор увеличения на единицу `++`);
- *бинарные* операторы (например, оператор сложения и большинство прочих операторов);
- *тернарные* операторы (оператор `?:`).

Бинарные операторы образуют иерархию, старшие операторы, расположенные выше в иерархии, выполняются перед тем, как будут выполнены операторы, расположенные ниже в иерархии. Иерархия операторов выглядит следующим образом:

- `new`
- `[`
- `++, --`
- `!, ~, -, (int), (float), (string), (array), (object), @`
- `*, /, %`
- `+, -, .`
- `<<, >>`
- `<, <=, >, >=`
- `==, !=, ===, !==`
- `&`
- `^`
- `|`
- `&&`
- `||`
- `?, :`
- `=, +=, -=, *=, /=, .=, %=, &=, |=, ^=, <<=, >>=`
- `and`
- `xor`
- `or`
- `,`

Арифметические операторы

Арифметические операторы — это бинарные операторы сложения, вычитания, умножения, деления и деления с остатком.

- `$a + $b` — сложение
- `$a – $b` — вычитание
- `$a * $b` — умножение
- `$a / $b` — деление (результат всегда имеет тип `float`)
- `$a % $b` — модуль (целочисленный остаток от деления `$a` на `$b`)

Операторы присваивания

Оператор присваивания задает значение переменной. Базовый оператор записывается в виде =. Пример работы с оператором присваивания приведен в листинге 3.3.

Листинг 3.3. Оператор присваивания

```
$a = ($b = 4) + 5; // $a сейчас равно 9, а $b имеет значение 4.
```

Кроме базовой операции присвоения, имеются "комбинированные операции" для всех бинарных, арифметических и строковых операций, например:

```
$a = 3;
$a += 5; // Устанавливает в $a 8, как если бы мы сказали: $a = $a + 5;
$b = "Hello ";
$b .= "There!"; // Устанавливает в $b "Hello There!", аналогично $b = $b
. "There!";
```

Побитовые операторы

Побитовые операторы предоставляют возможность работать со значениями бит.

- `$a & $b (AND)` — ненулевыми устанавливаются биты, которые не равны нулю как в `$a`, так и в `$b`;
- `$a | $b (Or)` — ненулевыми устанавливаются биты, равные 1 или в переменной `$a`, или в переменной `$b`;
- `$a ^ $b (Xor)` — ненулевыми устанавливаются биты, равные 1 или в `$a`, или `$b`, но не в обеих переменных;
- `~ $a (Not)` — ненулевыми устанавливаются биты, которые в `$a` являются нулевыми, и наоборот;

- `$a << $b` — сдвигает биты переменной `$a` на `$b` шагов влево (каждый шаг/смещение означает "умножить на 2");
- `$a >> $b` — сдвигает биты переменной `$a` на `$b` шагов вправо (каждый шаг/смещение означает "разделить на 2").

Операторы сравнения

Операторы сравнения — это бинарные операторы, значением которых является тип `Boolean`. Эти операторы помогают сравнивать значения переменных.

- `$a == $b` (равно) — `TRUE`, если `$a` равно `$b`;
- `$a === $b` (идентично) — `TRUE`, если значение переменной `$a` равно `$b`, и обе переменные одного типа;
- `$a != $b` (не равно) — `TRUE`, если `$a` не равно `$b`;
- `$a <> $b` (не равно) — `TRUE`, если `$a` не равно `$b`;
- `$a !== $b` (не идентично) — `TRUE`, если `$a` не равно `$b` или они разных типов; (только в PHP 4);
- `$a < $b` (меньше) — `TRUE`, если `$a` строго меньше `$b`;
- `$a > $b` (больше) — `TRUE`, если `$a` строго больше `$b`;
- `$a <= $b` (меньше или равно) — `TRUE`, если `$a` меньше или равно `$b`;
- `$a >= $b` (больше или равно) — `TRUE`, если `$a` больше или равно `$b`.

Условным оператором является и тернарный оператор "`?:`".

```
(expr1) ? (expr2) : (expr3);
```

Это выражение принимает значение `expr2`, если `expr1` имеет значение `TRUE`, или значение `expr3`, если `expr1` имеет значение `FALSE`.

Оператор управления ошибками

PHP поддерживает одну операцию управления ошибками: знак `@`. Если он вставлен как префикс выражения PHP, любые ошибки, которые могут генерироваться этим выражением, подавляются.

Если включена опция `track_errors`, то любые сообщения об ошибках, генерируемые выражением, будут сохраняться в глобальной переменной `$php_errormsg`. Эта переменная будет перезаписываться при возникновении каждой новой ошибки. Например:

```
<?php
/* Предполагается файловая ошибка */
$my_file = @file ('non_existent_file') or
    die ("Failed opening file: error was '$php_errormsg'");
```

```
// Работает для всех выражений, а не только в функциях:
$value = @$cache[$key];
// Уведомления не будет, если индекс $key не существует.

?>
```

Примечание

Оператор `@` работает только в выражениях. Основное правило: если можно получить значение чего-нибудь, то можно поставить в качестве префикса оператор `@`. Например, можно поставить его к переменным, функциям и вызовам `include()`, константам и т. д. Нельзя подставить этот оператор к определению функции или класса, к структурам управления, таким как `if` и `foreach`, и т. д.

Операторы увеличения и уменьшения на единицу

Следующие операторы увеличивают или уменьшают значение переменной на единицу:

- `++$a` — увеличивает `$a` на 1, затем возвращает `$a`;
- `$a++` — возвращает `$a`, затем увеличивает `$a` на 1;
- `--$a` — уменьшает `$a` на 1, затем возвращает `$a`;
- `$a--` — возвращает `$a`, затем уменьшает `$a` на 1.

Логические операторы

Логические операторы приведены в табл. 3.1.

Таблица 3.1. *Логические операторы*

Условие	Выражение	Описание
И	`$a and $b`	`TRUE`, если и `$a`, и `$b` — `TRUE`
И	`$a && $b`	`TRUE`, если и `$a`, и `$b` `TRUE`
ИЛИ	`$a || $b`	`TRUE`, если `$a` или `$b` `TRUE`
ИЛИ	`$a or $b`	`TRUE`, если `$a` или `$b` `TRUE`
Исключающее ИЛИ	`$a xor $b`	`TRUE`, если `$a` или `$b` `TRUE`, но не оба
НЕ	`! $a`	`TRUE`, если `$a` не `TRUE`

Смысл двух вариантов "И" и "ИЛИ" в том, что они работают с различными приоритетами: операторы `&&` и `||` имеют более старший приоритет, чем соответствующие им `and` и `or`.

Строковые операции

Имеются две строковые операции. Первая — операция конкатенации (`'.'`), которая возвращает объединение из правого и левого аргументов. Вторая — операция присвоения (`'.='`), которая присоединяет правый аргумент к левому аргументу. Например:

```
$a = "Hello ";
$b = $a . "World!"; // Теперь $b содержит "Hello World!"

$a = "Hello ";
$a .= "World!";     //
```

Управление ходом выполнения программы

В этом разделе представлены операторы, с помощью которых происходит управление ходом выполнения программы.

Оператор *if*

Оператор `if` позволяет проверить истинность выражения и выполнить инструкцию только в том случае, если заданное выражение принимает значение `TRUE`. Общий вид оператора таков:

```
if (expr) statement
```

В листингах 3.4 и 3.5 приведены простые примеры.

Листинг 3.4. Простое выражение

```
if ($a > $b)
    print "a is bigger than b";
```

Листинг 3.5. Составное выражение в блоке

```
if ($a > $b) {
    print "a is bigger than b";
    $b = $a;
}
```

Оператор *else*

Если в случае невыполнения условия следует выполнить какие-либо действия, то в составе оператора `if` используется условие `else`. После `else` запи-

сывается выражение, которое выполняется в том случае, если условие принимает значение false.

```
if ($a > $b) {
    print "a is bigger than b";
} else {
    print "a is NOT bigger than b";
}
```

Оператор *elseif*

Если после else следует выражение, требующее проверки какого-либо условия, и есть выражение, которое содержит if, то оба оператора можно объединить в elseif:

```
if ($a > $b) {
    print "a is bigger than b";
} elseif ($a == $b) {
    print "a is equal to b";
} else {
    print "a is smaller than b";
}
```

Оператор *while*

Оператор while используется для организации циклов. Общий вид его таков:

```
while (expr) statement
```

Выполнение выражения statement (оно может быть как простым выражением, так и блоком, заключенным в фигурные скобки и состоящим из нескольких простых выражений) происходит до тех пор, пока остается истинным условие expr. Допускается альтернативная запись, когда вместо открывающей фигурной скобки используется двоеточие, а вместо закрывающей скобки слово endwhile:

```
while (expr): statement ... endwhile;
```

Пример работы с оператором while приведен в листингах 3.6 и 3.7.

Листинг 3.6. Оператор `while`

```
$i = 1;
while ($i <= 10) {
    print $i++;  /* Будет печататься значение
                    $i до увеличения на единицу
                    (постинкремент) */
}
```

Листинг 3.7. Альтернативный вариант

```
while ($i <= 10):
    print $i;
    $i++;
endwhile;
```

Можно пользоваться альтернативной записью и для других условных операторов (листинг 3.8).

Листинг 3.8. Примеры альтернативной записи

```
if ($a == 5):
    print "a equals 5";
    print "...";
elseif ($a == 6):
    print "a equals 6";
    print "!!!";
else:
    print "a is neither 5 nor 6";
endif;
```

Оператор *do ... while*

В цикле `while` проверка условия происходит перед выполнением очередной итерации. Цикл `do ... while` проверяет условие после выполнения очередной итерации. Например, данный цикл будет выполнен только один раз:

```
$i = 0;
do {
   print $i;
} while ($i>0);
```

Этот оператор не имеет альтернативной записи.

Оператор *for*

Оператор `for` используется для организации циклов. Это наиболее сложный оператор, организующий циклы. Общий его вид такой:

```
for (expr1; expr2; expr3) statement
```

Оператор `for` работает следующим образом. Первое выражение `expr1` выполняется первый раз при входе в цикл. Второе выражение `expr2` выполняется перед каждой итерацией, если его значение `TRUE`, то цикл выполняется,

если его значение `FALSE`, то выполнение прекращается. После завершения очередной итерации выполняется выражение `expr3`.

Примеры работы с этим оператором приведены в листингах 3.9—3.12.

Листинг 3.9. Пример 1

```
for ($i = 1; $i <= 10; $i++) {
    print $i;
}
```

Листинг 3.10. Пример 2

```
for ($i = 1;;$i++) {
    if ($i > 10) {
        break;
    }
    print $i;
}
```

Листинг 3.11. Пример 3

```
$i = 1;
for (;;) {
    if ($i > 10) {
        break;
    }
    print $i;
    $i++;
}
```

Листинг 3.12. Пример 4

```
for ($i = 1; $i <= 10; print $i, $i++);
```

Оператор *foreach*

Оператор `foreach` позволяет просматривать весь массив, выполняя определенные действия с каждым его элементом.

Общий синтаксис оператора таков:

```
foreach(array_expression as $value) statement
foreach(array_expression as $key => $value) statement
```

Оператор первого вида осуществляет циклический проход по массиву, заданному в array_expression. При каждом проходе значение текущего элемента присваивается переменной $value, а внутренний указатель массива передвигается на единицу (поэтому при следующем проходе вы просмотрите значение следующего элемента). Пример работы с этим оператором приведен в листинге 3.13. Оператор второго вида позволяет также работать с ключами массива, которые будут содержаться в переменной $key (листинг 3.14).

Листинг 3.13. Пример для первого варианта

```
foreach ($arr as $value) {
    echo "Value: $value<br>\n";
}
```

Листинг 3.14. Пример для второго варианта

```
foreach ($arr as $key => $value) {
    echo "Key: $key; Value: $value<br>\n";
}
```

Эти примеры могут быть записаны в следующем эквивалентном виде:

```
reset ($arr);
while (list(, $value) = each ($arr)) {
    echo "Value: $value<br>\n";
}
reset ($arr);
while (list($key, $value) = each ($arr)) {
    echo "Key: $key; Value: $value<br>\n";
}
```

Оператор *break*

Оператор break используется для завершения выполнения циклических операторов. Оператор break позволяет прекратить выполнение циклических структур, задаваемых операторами for, foreach, while, do...while или switch. Оператору break можно передать целочисленный аргумент, который указывает, какое количество вложенных циклических структур будет завершено (листинг 3.15).

Листинг 3.15. Пример

```
$arr = array ('one', 'two', 'three', 'four', 'stop', 'five');
while (list (, $val) = each ($arr)) {
```

```
    if ($val == 'stop') {
        break;    /* Можно написать 'break 1;' */
    }
    echo "$val<br>\n";
}

/* Использование необязательного аргумента. */

$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "At 5<br>\n";
        break 1;  /* Выход только из switch. */
    case 10:
        echo "At 10; quitting<br>\n";
        break 2;  /* Выход из switch и из while. */
    default:
        break;
    }
}
```

Оператор *continue*

Оператор `continue` используется для осуществления перехода к новой итерации в цикле. Этому оператору можно передать целочисленный аргумент, который указывает количество вложенных циклов, которые следует пропустить (листинг 3.16).

Листинг 3.16. Пример

```
while (list ($key, $value) = each ($arr)) {
    if (!($key % 2)) { // Пропустить нечетные члены
        continue;
    }
    do_something_odd ($value);
}

$i = 0;
while ($i++ < 5) {
    echo "Outer<br>\n";
    while (1) {
        echo "& & Middle<br>\n";
```

```
        while (1) {
            echo "& & Inner<br>\n";
            continue 3;
        }
        echo "This never gets output.<br>\n";
    }
    echo "Neither does this.<br>\n";
}
```

Оператор *switch*

Оператор `switch` выполняет действия в зависимости от переданного ему целочисленного аргумента. Работу оператора покажем на примерах (листинг 3.17 и 3.18). Оба примера работают одинаково.

Листинг 3.17. Пример 1

```
switch ($i) {
    case 0:
        print "i equals 0";
        break;
    case 1:
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
}
```

Листинг 3.18. Пример 2

```
if ($i == 0) {
    print "i equals 0";
}
if ($i == 1) {
    print "i equals 1";
}
if ($i == 2) {
    print "i equals 2";
}
```

Оператор `break` выполняется построчно. Чтобы прекратить выполнение оператора `switch`, нужно использовать оператор `break`, иначе будут выполнены и последующие строки.

Функции

Язык PHP позволяет создавать функции. *Функция* — это набор инструкций, которые выполняются при обращении к функции. Обращение к функции осуществляется по *имени функции*. Функции могут быть переданы *параметры*. Обращаться можно только к описанной функции. Описание функции в общем имеет следующий вид:

```
function foo ($arg_1, $arg_2, ..., $arg_n)
{
    echo "Пример \n";
    return $retval;
}
```

Внутри функции может быть расположен произвольный код, в том числе могут быть описаны другие функции и классы. Аргументы функции перечисляются через запятую. Как правило (по умолчанию), аргументы передаются функции по значению, при этом значения переменных вне функции не меняются. Существует возможность передавать значения по ссылке (при этом значения переменной вне функции могут быть изменены). Чтобы передать значение по ссылке перед именем аргумента в списке параметров функции, следует поставить амперсанд (&) (листинг 3.19). Существует возможность указать значения параметров, которые будут использоваться по умолчанию (если при вызове функции параметр не будет указан) (листинг 3.20).

Листинг 3.19. Передача параметров по ссылке

```
function add_some_extra(&$string)
{
$string .= 'and something extra.';
}
$str = 'This is a string, ';
add_some_extra($str);
echo $str;// Выводит 'This is a string, and something extra.'
```

Листинг 3.20. Задание значений параметров по умолчанию

```
function makecoffee ($type = "cappuccino")
{
return "Making a cup of $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
```

Оператор `return` используется для прекращения выполнения функции. В качестве аргумента этого оператора указывается возвращаемое функцией значение. Например:

```
function square ($num)
{
    return $num * $num;
}
echo square (4);   // выводит '16'
```

Классы

Классы используются для задания пользовательских типов. В классах содержатся функции (методы) и переменные. В листинге 3.21 приведен синтаксис описания классов, а в листинге 3.22 — пример описания класса.

Листинг 3.21. Синтаксис описания класса

```
<?php

class Cart
{
    var $items;  // Элементы в нашей shopping cart

    // Добавить $num артикулов $artnr в cart

    function add_item ($artnr, $num)
    {
        $this->items[$artnr] += $num;
    }

    // Изъять $num артикулов $artnr из cart

    function remove_item ($artnr, $num)
    {
        if ($this->items[$artnr] > $num) {
            $this->items[$artnr] -= $num;
            return true;
        } else {
            return false;
        }
    }
}
?>
```

Листинг 3.22. Пример описания класса

```
class Cart {
   var $todays_date;
   var $name;
   var $owner;
   var $items = array("VCR", "TV");

   function Cart() {
       $this->todays_date = date("Y-m-d");
       $this->name = $GLOBALS['firstname'];
       /* etc. . . */
   }
}
```

Описанный тип можно использовать для создания объектов этого типа с помощью оператора `new` (листинг 3.23).

Листинг 3.23. Создание объекта типа `cart`

```
$cart = new Cart;
$cart->add_item("10", 1);
$another_cart = new Cart;
$another_cart->add_item("0815", 3);
```

Оператор *extends*

Оператор `extends` служит для создания классов, в которых используется функциональность другого класса. Таким образом осуществляется наследование классов. Например:

```
class Named_Cart extends Cart
{
    var $owner;

    function set_owner ($name)
    {
        $this->owner = $name;
    }
}
```

Здесь определен класс `Named_Cart`, который содержит все переменные и функции класса `Cart` плюс дополнительную переменную `$owner` и дополнительную функцию `set_owner()`. Пример работы с оператором `extends` приведен в листинге 3.24.

Листинг 3.24. Создание новых объектов

```
$ncart = new Named_Cart;     // создаем named cart
$ncart->set_owner("kris");   // именуем cart
print $ncart->owner;         // печатаем имя владельца cart
$ncart->add_item("10", 1);   // (функциональность, унаследованная от cart)
```

Конструкторы

Конструктором класса называется функция, которая вызывается всякий раз при создании нового объекта данного класса. Чтобы функция стала конструктором данного класса, необходимо, чтобы ее имя совпадало с именем этого класса (листинг 3.25). Конструктором также является специальный метод `construct` (листинг 3.26):

```
__construct ( [mixed args [, ...]]);
```

Листинг 3.25. Пример

```
class Auto_Cart extends Cart {
   function Auto_Cart() {
       $this->add_item("10", 1);
   }
}
class Constructor_Cart extends Cart {
   function Constructor_Cart($item = "10", $num = 1) {
       $this->add_item ($item, $num);
   }
}
// Старый вариант
$default_cart = new Constructor_Cart;
// Новый вариант
$different_cart = new Constructor_Cart("20", 17);
```

Листинг 3.26. Пример с использованием метода `construct`

```
class BaseClass {
   function __construct() {
       print "In BaseClass constructor\n";
   }
}

class SubClass extends BaseClass {
   function __construct() {
       parent::__construct();
```

```
        print "In SubClass constructor\n";
    }
}

$obj = new BaseClass();
$obj = new SubClass();
```

Деструктор

Деструктор — это метод, который вызывается тогда, когда удаляются все ссылки на конкретный объект. Деструктор создается при помощи функции `__destruct`, например:

```
class MyDestructableClass {
    function __construct() {
        print "In constructor\n";
        $this->name = "MyDestructableClass";
    }

    function __destruct() {
        print "Destroying " . $this->name . "\n";
    }
}

$obj = new MyDestructableClass();
```

Область видимости членов класса

Область видимости элементов класса задается при помощи зарезервированных слов `public`, `private` и `protected`. По умолчанию (если не указан модификатор) метод класса обладает видимостью `public` (листинги 3.27 и 3.28).

Листинг 3.27. Пример с членами класса

```
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
```

```
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Works
echo $obj->protected; // Fatal Error
echo $obj->private; // Fatal Error
$obj->printHello(); // Shows Public, Protected and Private


/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
    // We can redeclare the public and protected method, but not private
    protected $protected = 'Protected2';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj2 = new MyClass2();
echo $obj->public; // Works
echo $obj2->private; // Undefined
echo $obj2->protected; // Fatal Error
$obj2->printHello(); // Shows Public, Protected2, not Private
```

Листинг 3.28. Пример с методами класса

```
/**
 * Define MyClass
 */
class MyClass
{
    // Contructors must be public
    public function __construct() { }

    // Declare a public method
    public function MyPublic() { }
```

```
    // Declare a protected method
    protected function MyProtected() { }

    // Declare a private method
    private function MyPrivate() { }

    // This is public
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Works
$myclass->MyProtected(); // Fatal Error
$myclass->MyPrivate(); // Fatal Error
$myclass->Foo(); // Public, Protected and Private work


/**
 * Define MyClass2
 */
class MyClass2 extends MyClass
{
    // This is public
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Fatal Error
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Works
$myclass2->Foo2(); // Public and Protected work, not Private
```

Оператор *::*

Иногда удобно обращаться к функциям или переменным в базовых классах или обратиться к функциям в классах, которые еще не имеют экземпляров. Для этого используется оператор `::` (листинг 3.29).

Листинг 3.29. Пример

```
class A
{
    function example()
    {
        echo "I am the original function A::example().<br>\n";
    }
}

class B extends A
{
    function example()
    {
        echo "I am the redefined function B::example().<br>\n";
        A::example();
    }
}

// нет объекта класса A.
// будет напечатано
//   I am the original function A::example().<br>
A::example();

// создается объект класса B.
$b = new B;

// будет напечатано
//   I am the redefined function B::example().<br>
//   I am the original function A::example().<br>
$b->example();
```

Ключевое слово *static*

Если метод или переменную объявить как `static`, то к ним можно будет обращаться вне контекста объекта. Такое объявление должно следовать после объявления видимости метода или переменной. По умолчанию используется следующее описание переменных и методов: `public static`. Внутри `static`-метода нельзя использовать указатель `$this`, являющийся ссылкой на данный объект, в контексте которого он используется. В листинге 3.30 приведен пример объявления переменной как `static`, а в листинге 3.31 — пример объявления метода.

Листинг 3.30. Пример с переменными

```
class Foo
{
   public static $my_static = 'foo';

   public function staticValue() {
       return self::$my_static;
   }
}

class Bar extends Foo
{
   public function fooStatic() {
       return parent::$my_static;
   }
}


print Foo::$my_static . "\n";

$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static . "\n";      // Undefined "Property" my_static

// $foo::my_static is not possible

print Bar::$my_static . "\n";
$bar = new Bar();
print $bar->fooStatic() . "\n";
```

Листинг 3.31. Пример с методами

```
class Foo {
   public static function aStaticMethod() {
       // ...
   }
}
Foo::aStaticMethod();
```

Абстрактные классы

Абстрактные классы могут содержать только сигнатуры методов, и не содержат их имплементации. Такие методы также называются *абстрактными методами*. Если класс содержит хотя бы один абстрактный метод, он дол-

жен быть объявлен как абстрактный класс. Абстрактные классы не имеют экземпляров классов, на их основе нельзя создать объект. Класс, имплементирующий абстрактный класс, должен описывать абстрактные методы имплементируемого класса с более доступной видимостью. Так, если базовый класс содержит `protected`-метод, то имплементирующий класс может определять этот метод как `protected`, или как `public` (листинг 3.32).

Листинг 3.32. Пример

```
abstract class AbstractClass
{
   // Force Extending class to define this method
   abstract protected function getValue();

   // Common method
   public function printOut() {
       print $this->getValue();
   }
}

class ConcreteClass1 extends AbstractClass
{
   protected function getValue() {
       return "ConcreteClass1";
   }
}

class ConcreteClass2 extends AbstractClass
{
   protected function getValue() {
       return "ConcreteClass2";
   }

}

$class1 = new ConcreteClass1;
$class1->printOut();

$class2 = new ConcreteClass2;
$class2->printOut();
```

Интерфейсы

Помимо абстрактных классов в PHP предусмотрена возможность работы с *интерфейсами*. Интерфейсы содержат только сигнатуры методов. Все методы интерфейса должны быть объявлены в виде `public`.

Функциональность методов интерфейса описывается в классе, имплементирующем данный интерфейс. Имплементирующий класс должен содержать имплементацию всех методов имплементируемого интерфейса (листинг 3.33).

Листинг 3.33. Пример

```
// Описание интерфейса 'iTemplate'
interface iTemplate
{
   public function setVariable($name, $var);
   public function getHtml($template);
}

// Имплементация интерфейса
// Работающий вариант
class Template implements iTemplate
{
   private $vars = array();

   public function setVariable($name, $var)
   {
       $this->vars[$name] = $var;
   }

   public function getHtml($template)
   {
       foreach($this->vars as $name => $value) {
           $template = str_replace('{' . $name . '}', $value, $template);
       }

       return $template;
   }
}

// Неработающий вариант
// Fatal error: класс BadTemplate содержит один абстрактный метод,
// следовательно должен быть объявлен как абстрактный
class BadTemplate implements iTemplate
{
   private $vars = array();

   public function setVariable($name, $var)
   {
       $this->vars[$name] = $var;
   }
}
```

Оператор *final*

Оператор `final` защищает методы класса и не позволяет переопределять их в дочерних классах. Если класс описан как `final`, то на его основе нельзя создать дочерние классы (листинги 3.34 и 3.35).

Листинг 3.34. Пример с методами

```
    public function test() {
        echo "BaseClass::test() called\n";
    }

    final public function moreTesting() {
        echo "BaseClass::moreTesting() called\n";
    }
}

class ChildClass extends BaseClass {
    public function moreTesting() {
        echo "ChildClass::moreTesting() called\n";
    }
}
// Results in Fatal error: Cannot override final method
BaseClass::moreTesting()
```

Листинг 3.35. Пример с классами

```
final class BaseClass {
    public function test() {
        echo "BaseClass::test() called\n";
    }

    // Here it doesn't matter if you specify the function as final or not
    final public function moreTesting() {
        echo "BaseClass::moreTesting() called\n";
    }
}

class ChildClass extends BaseClass {
}
// Results in Fatal error: Class ChildClass may not inherit from final
class (BaseClass)
```

ГЛАВА 4

Работаем с данными

Данные в PHP

Язык PHP поддерживает работу со многими базами данных. Часто вместе с PHP используется база данных MySQL. MySQL, подобно PHP, является продуктом с открытым кодом. Установка MySQL предельно проста: в системе Windows достаточно распаковать дистрибутив в нужную папку. Запустить сервер MySQL можно из командной строки командой `mysqld` или `mysqld-nt` (в Windows NT). Проверим, работает ли сервер, выполнив команду `mysqlshow`. Если сервер работает, то в результате получим примерно такой ответ:

```
--------------
Databases
-------------
mysql
test
```

Запускаем клиент MySQL (команда `mysql`), чтобы потренироваться в основных командах языка SQL. Появляется приглашение в форме стрелки:

```
->
```

Чтобы начать работу, нужно выбрать базу данных:

```
-> use test;
```

Каждая инструкция MySQL кончается точкой с запятой.

Для работы с базами данных из PHP необходимо знать, как создавать запросы на языке SQL. Приведем список основных функций языка PHP, предназначенных для работы с MySQL:

- `mysql_affected_rows` — выдает количество строк, полученных предыдущим оператором;

- `mysql_close` — закрывает соединение;
- `mysql_connect` — открывает соединение с сервером MySQL;
- `mysql_create_db` — создает базу данных MySQL;
- `mysql_data_seek` — перемещает внутренний указатель в полученном результате;
- `mysql_db_query` — посылает MySQL-запрос;
- `mysql_drop_db` — удаляет базу данных MySQL;
- `mysql_fetch_array` — подбирает результат запроса в виде ассоциативного или нумерованного массива или в виде того и другого;
- `mysql_fetch_assoc` — подбирает результат запроса в виде ассоциативного массива;
- `mysql_fetch_field` — возвращает информацию о колонке и возвращает объект;
- `mysql_fetch_lengths` — возвращает длину каждого результата;
- `mysql_fetch_object` — возвращает строку результата в виде объекта;
- `mysql_fetch_row` — возвращает строку результата в виде нумерованного массива;
- `mysql_field_name` — возвращает имя поля в результате;
- `mysql_field_len` — возвращает длину поля;
- `mysql_field_seek` — устанавливает указатель в результате на заданную позицию;
- `mysql_field_table` — возвращает имя таблицы, в которой находится результат;
- `mysql_field_type` — возвращает тип поля в результате;
- `mysql_insert_id` — возвращает идентификатор `id`, сгенерированный предыдущей операцией `INSERT`;
- `mysql_list_dbs` — возвращает список баз данных, доступных на сервере MySQL;
- `mysql_list_tables` — получает список таблиц базы MySQL;
- `mysql_num_fields` — возвращает количество полей результата;
- `mysql_num_rows` — возвращает количество строк результата;
- `mysql_pconnect` — открывает постоянное соединение с сервером MySQL;
- `mysql_query` — посылает запрос серверу MySQL;
- `mysql_unbuffered_query` — посылает SQL-запрос серверу MySQL без буферизации (и возможности дальнейшего подбора) строк результата;

- `mysql_result` — получает данные результата;
- `mysql_select_db` — выбирает базу данных MySQL;
- `mysql_get_host_info` — информация о хосте базы MySQL;
- `mysql_get_proto_info` — информация о протоколе MySQL;
- `mysql_get_server_info` — информация о сервере MySQL.

Это не полный перечень PHP-функций, предназначенных для работы с базами данных MySQL. PHP имеет готовые к применению функции, разработанные для различных баз данных, включая Informix, Oracle, MS SQL и др. При работе с конкретной базой данных необходимо учитывать ее особенности. Важно правильно сконфигурировать PHP. Мы не будем подробно останавливаться на деталях работы с каждым типом баз данных. Сейчас мы перейдем к рассмотрению основных примеров работы на языке PHP с базой данных MySQL.

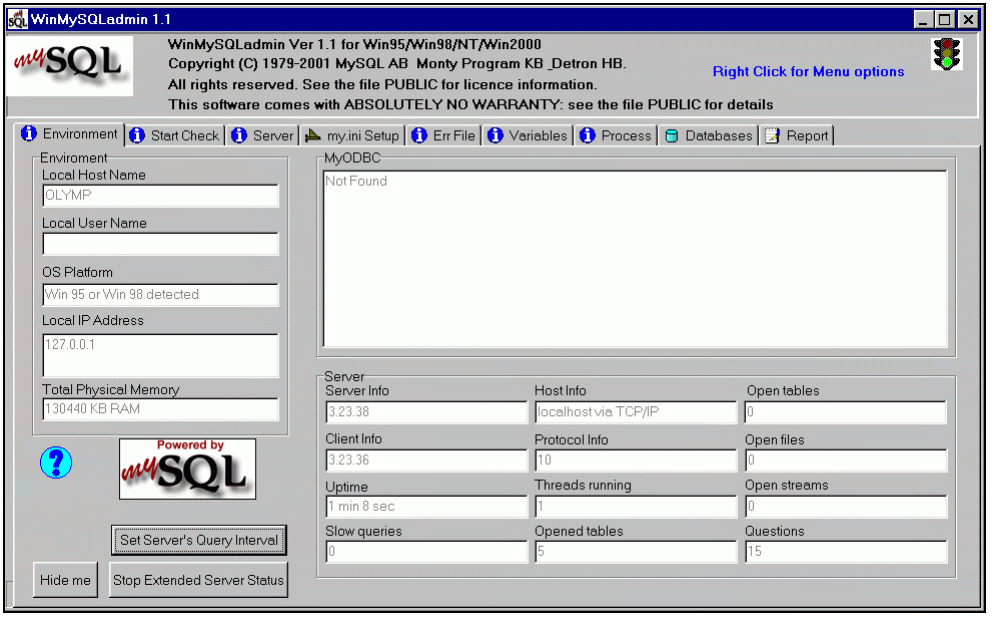

Рис. 4.1. Панель администратора

Еще раз убедимся в том, что сервер MySQL работает. Для этого в каталоге mysql\bin выполним команду `mysql`. В консольном окне должно появиться приглашение `mysql>`. Если приглашение не появилось, то в среде Windows сервер удобно запустить директивой `winmysqladmin`, расположенной в том же каталоге. На дисплее появится основное окно панели администратора (рис. 4.1), которое тут же исчезнет, если сервер удачно стартует. Если в сис-

теме нет файла my.ini, то появится приглашение зарегистрировать пользователя (рис. 4.2).

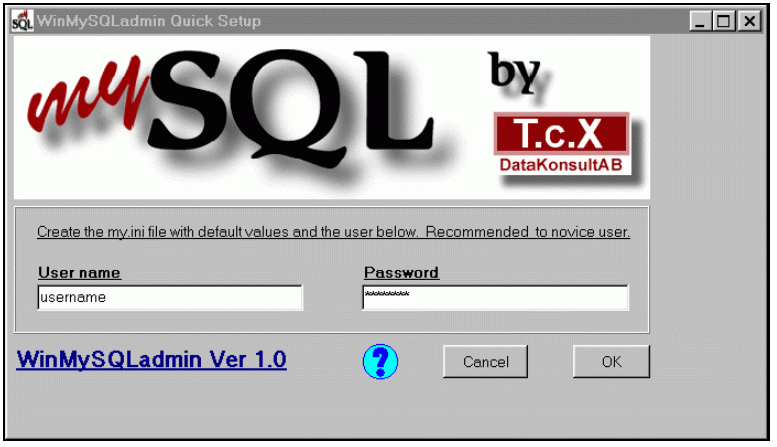

Рис. 4.2. Регистрируем пользователя сервера MySQL

Убедимся, что сервер работает. Если запущена программа WinMySQLadmin, то на панели состояния системы появится изображение светофора. При работающем MySQL-сервере цвет светофора будет зеленый (рис. 4.3).

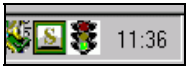

Рис. 4.3. Светофор показывает, что сервер работает

Попытаемся обратиться к серверу из консоли, для этого вызовем клиент командой `mysql`. На рис. 4.4 показана ситуация, когда сервер работает.

Создадим простой PHP-файл и обратимся к MySQL-серверу через CGI-приложение из клиентского браузера.

Файл mysql1.php разместим в каталоге \cgi-bin Web-сервера (листинг 4.1).

Листинг 4.1. Запрос к серверу

```
<?php
$c=mysql_connect("localhost:3306", "username", "password");
mysql_select_db("test",$c);
$stroka="show tables";
$result = mysql_query($stroka);
if($result) echo $result;
?>
```

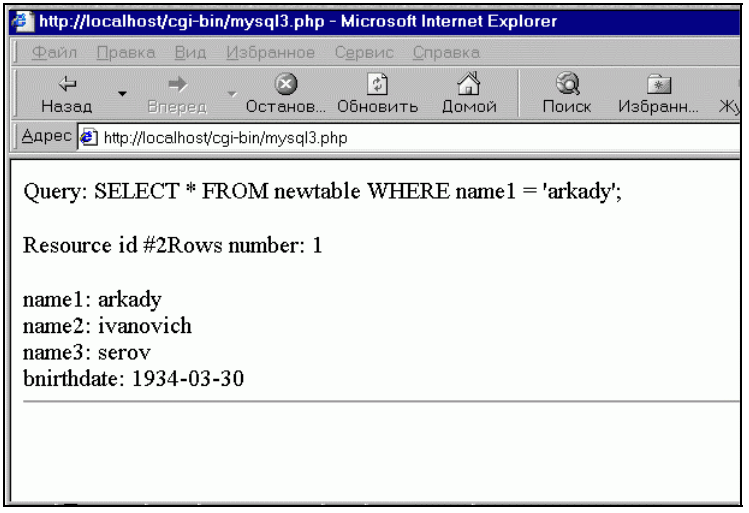

Рис. 4.4. Консольный клиент благополучно общается с сервером

Эта программа осуществляет соединение с сервером MySQL (первый оператор), выбирает базу данных (стандартную базу с именем test) и выполняет запрос. Если запрос выполняется, то функция `mysql_query` возвращает значение, равное идентификатору ресурса и отличное от `FALSE`. Если значение `$result` не ложно (т. е. запрос выполнен), то программа передает браузеру через сервер идентификатор ресурса, связанного с запросом `$stroka`. На рис. 4.5 показано, что запрос выполнен и наша программа успешно работает.

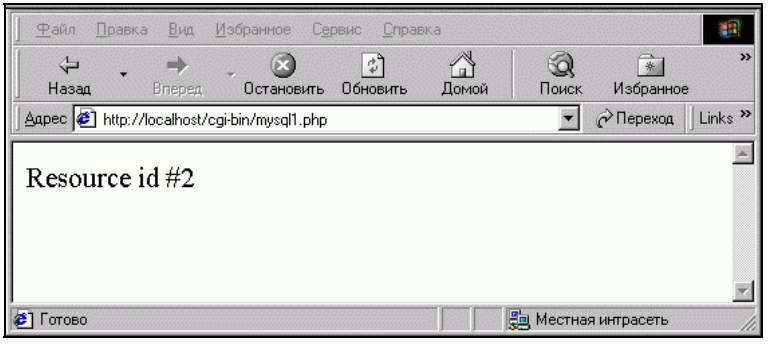

Рис. 4.5. MySQL-запрос обработан сервером

Создадим простейший запрос к базе данных, с помощью которого узнаем версию MySQL и текущую дату, и выведем результат в окно браузера (листинг 4.2).

Листинг 4.2. Запрос версии MySQL и текущей даты

```
<h3>
Запрос: "SELECT VERSION(), CURRENT_DATE;";
</h3>
Результат:
<b>
<?php
/* Connecting, selecting database */
$link = mysql_connect("localhost", "", "")
   or die("Could not connect : " . mysql_error());
echo "<script language=javascript>alert('Connected
successfully')</script>";
//mysql_select_db("my_database") or die("Could not select database");

/* Performing SQL query */
$query = "SELECT VERSION(), CURRENT_DATE;";
$result = mysql_query($query) or die("Query failed : " . mysql_error());

/* Printing results in HTML */
echo "<table>\n";
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
   echo "\t<tr>\n";
   foreach ($line as $col_value) {
       echo "\t\t<td>$col_value</td>\n";
   }
   echo "\t</tr>\n";
}
echo "</table>\n";

/* Free resultset */
mysql_free_result($result);

/* Closing connection */
mysql_close($link);
?>
</b>
```

При работе этот PHP-скрипт устанавливает соединение с базой данных, и в случае успешного соединения средствами JavaScript выводит окно-предупреждение, сообщающее о том, что соединение установлено. После закрытия этого модульного окна в основном окне браузера появляется информация о версии MySQL и текущем времени (рис. 4.6).

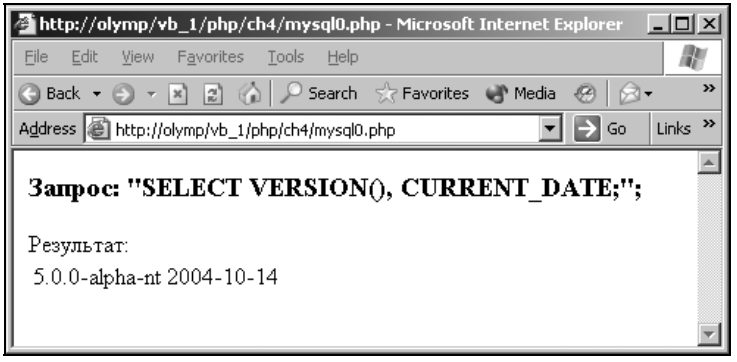

Рис. 4.6. Информация о версии сервера MySQL и текущем времени

Рассмотрим более интересный для практики пример.

Создадим таблицу newtable в существующей базе данных test:

```
mysql> CREATE TABLE newtable (name1 VARCHAR(20), name2 VARCHAR(20), name3
VARCHAR(20), sex CHAR(1), birth DATE);
```

Посмотрим, какие имеются сейчас таблицы в нашей базе данных:

```
Mysql> show tables;
```

Вставим вручную данные в таблицу (два ряда):

```
Mysql> INSERT INTO newtable VALUES
('Arkady','Ivanovich','Serov','m','1934-03-30');
```

и

```
Mysql> INSERT INTO newtable VALUES ('Max','I','Klein','m','1964-08-23');
```

Сейчас посмотрим, что содержат наша таблица:

```
Mysql> select * from newtable;
```

Результат показан на рис. 4.7.

Сейчас мы научимся читать данные из базы данных средствами PHP. Создадим простой файл mysql.php (листинг 4.3).

Листинг 4.3. Файл mysql.php

```
<?php
$c=mysql_connect("localhost:3306", "olymp", "vad");
echo mysql_select_db("newdb",$c);

$stroka = "select * from newtable;";
$result = mysql_query($stroka);
```

```
$noro=mysql_num_rows($result);
echo "Rows number: ".$noro."<P>";
while ($row = mysql_fetch_array ($result))
{

    echo "name1: ".$row["name1"]."<br>\n";
    echo "name2: ".$row["name2"]."<br>\n";
    echo "name3: ".$row["name3"]."<br>\n";
    echo "bnirthdate: ".$row["birth"]."<br>\n";
    echo "<hr>";
}
```

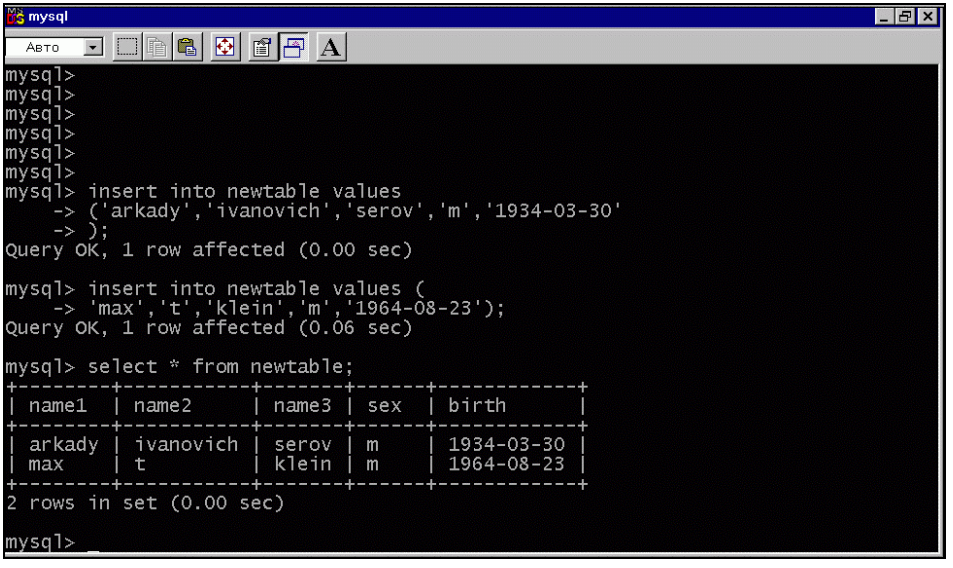

Рис. 4.7. Заполнили таблицу newtable

Здесь первая строка предназначена для соединения с базой данных, которая установлена на локальном компьютере со стандартным портом для MySQL — 3306, указаны пользователь и пароль, для проверки выводится идентификатор ресурса. В переменной `$stroka` формируется SQL-запрос, который передается функции `mysql_query`. Для контроля далее определяется, сколько строк содержится в результате. Затем цикл выводит полученный результат в окно браузера.

Для успешной работы с базами данных необходимо знать язык запросов к базам данных.

Работаем с MySQL

Выбор отдельного ряда

Мы имеем возможность выбрать ряды на основе указанных условий. Для этого нам надо написать запрос, указав эти условия, например:

```
mysql> SELECT * FROM newtable WHERE name1 = "arkady";
```

PHP-запрос будет выглядеть таким образом (листинг 4.4).

Листинг 4.4. Выбор отдельного ряда

```
<?php
$c=mysql_connect("localhost:3306", "", "");
mysql_select_db("test",$c);
$stroka = "SELECT * FROM newtable WHERE name1 = 'arkady';";
echo "Query: ".$stroka."<P>";
echo $result = mysql_query($stroka);
$noro=mysql_num_rows($result);
echo "Rows number: ".$noro."<P>";
while ($row = mysql_fetch_array ($result))
{

    echo "name1: ".$row["name1"]."<br>\n";
    echo "name2: ".$row["name2"]."<br>\n";
    echo "name3: ".$row["name3"]."<br>\n";
    echo "bnirthdate: ".$row["birth"]."<br>\n";
    echo "<hr>";
}
?>
```

В дальнейшем мы будем изменять только значение переменной `$stroka`. Будьте внимательны к пробелам. Лишние пробелы внутри строки запроса (в ее конце) могут привести к ошибке запроса. Если запрос составлен верно, получаем ответ, страница с которым показана на рис. 4.8. Если мы вставим лишний пробел в строке перед закрывающей кавычкой, MySQL-сервер будет работать неправильно, и определить такую ошибку нелегко (рис. 4.9).

Ошибки при обращении к серверу могут происходить многократно. Удаление одной ошибки не гарантирует работоспособность программы в целом (рис. 4.10).

Расширим нашу таблицу, для чего используем такие данные:

```
$string="insert into newtable values
('nikolay','stepanovich','ilyin','m','1978-08-23')";
$res = mysql_db_query("test", $string);
```

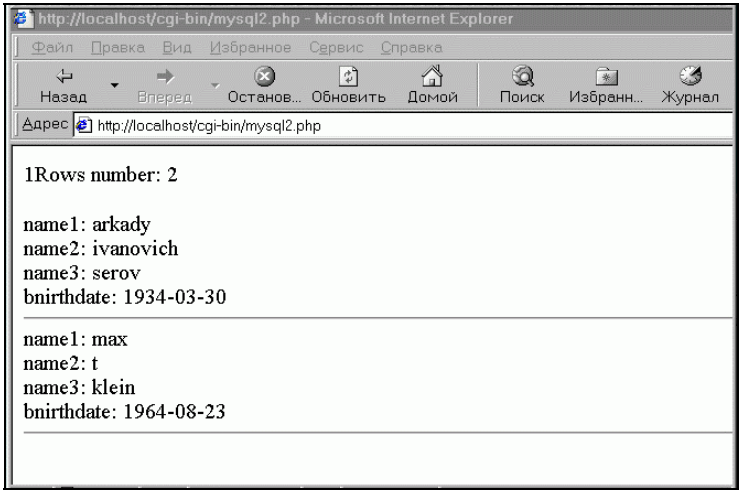

Рис. 4.8. Ответ сервера в случае верно составленного запроса

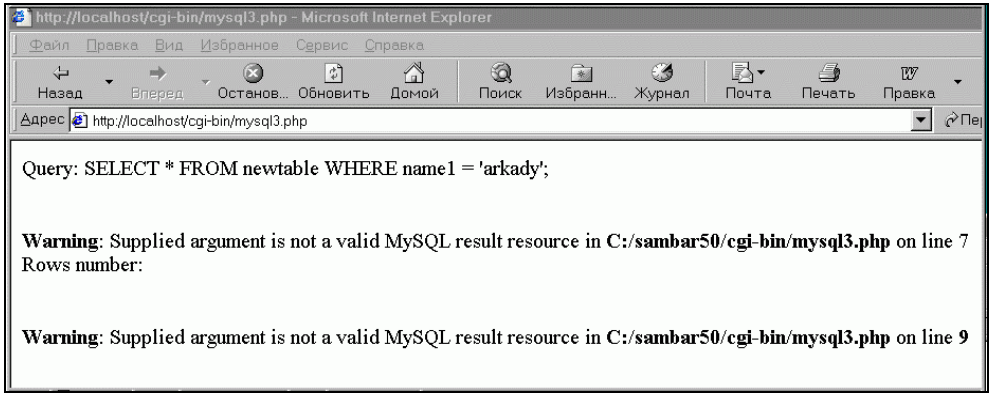

Рис. 4.9. Ошибка в работе MySQL-сервера из-за неправильного запроса

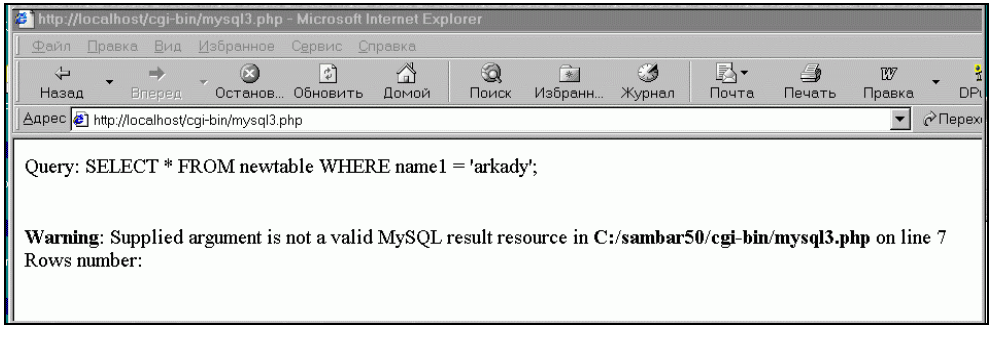

Рис. 4.10. Ответ сервера при возникновении ошибки

Несколько раз варьируем данные и вводим их по очереди:

```
$string="insert into newtable values
('nikolay','stepanovich','karaev','m','1979-03-13')";
$res = mysql_db_query("test", $string);
$string="insert into newtable values
('nikolay','grigorievich','hajt','m','1963-02-03')";
$res = mysql_db_query("test", $string);
```

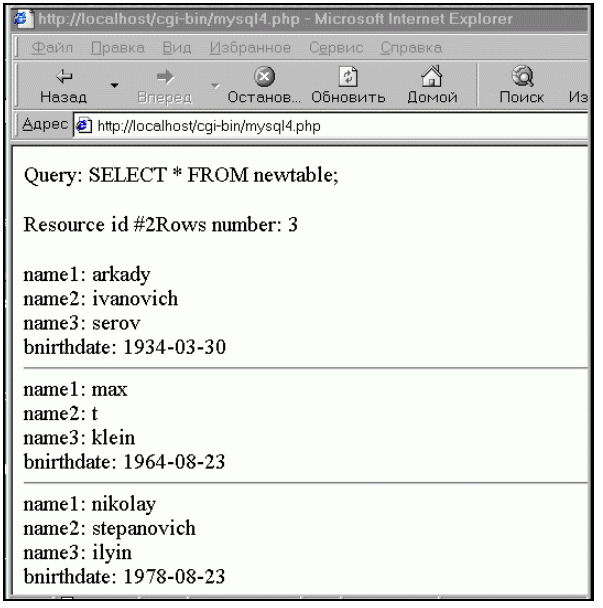

Рис. 4.11. Таблица newtable содержит три записи

Сейчас мы можем усложнить условие запроса — пусть будет выведен список лиц, родившихся до 1 января 1960 года:

```
$stroka = "SELECT * FROM newtable where birth < '1960-01-01';";
```

Из таблицы newtable этому условию соответствует только одна запись, которая и будет выведена (рис. 4.12).

Еще более сложное условие — пусть будет выведена запись о лицах с именем "nikolay", родившихся после 1 января 1970 года:

```
$stroka = "SELECT * FROM newtable where name1='nikolay' and birth >
'1970-01-01';";
```

Результат показан на рис. 4.13.

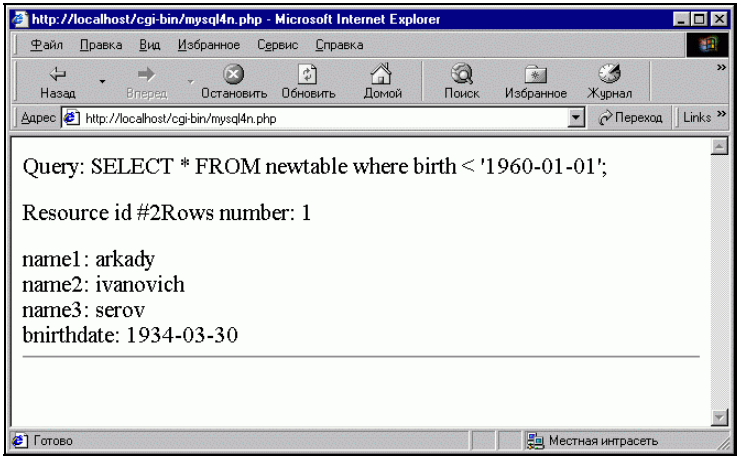

Рис. 4.12. Результат выполнения сложного условия запроса

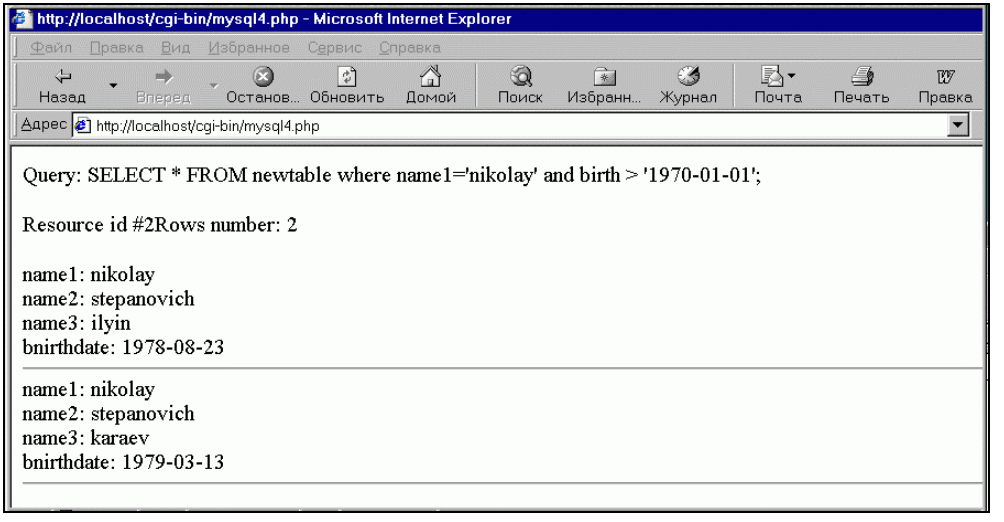

Рис. 4.13. Немного более сложное условие

Логические операторы `OR` и `AND` можно комбинировать, например:

```
$stroka = "SELECT * FROM newtable where name1='nikolay' or name2='t' and
birth > '1960-01-01';";
```

Результат показан на рис. 4.14.

Выбор столбца

Для того чтобы выбрать столбец, можно составить такой запрос, представленный в листинге 4.5.

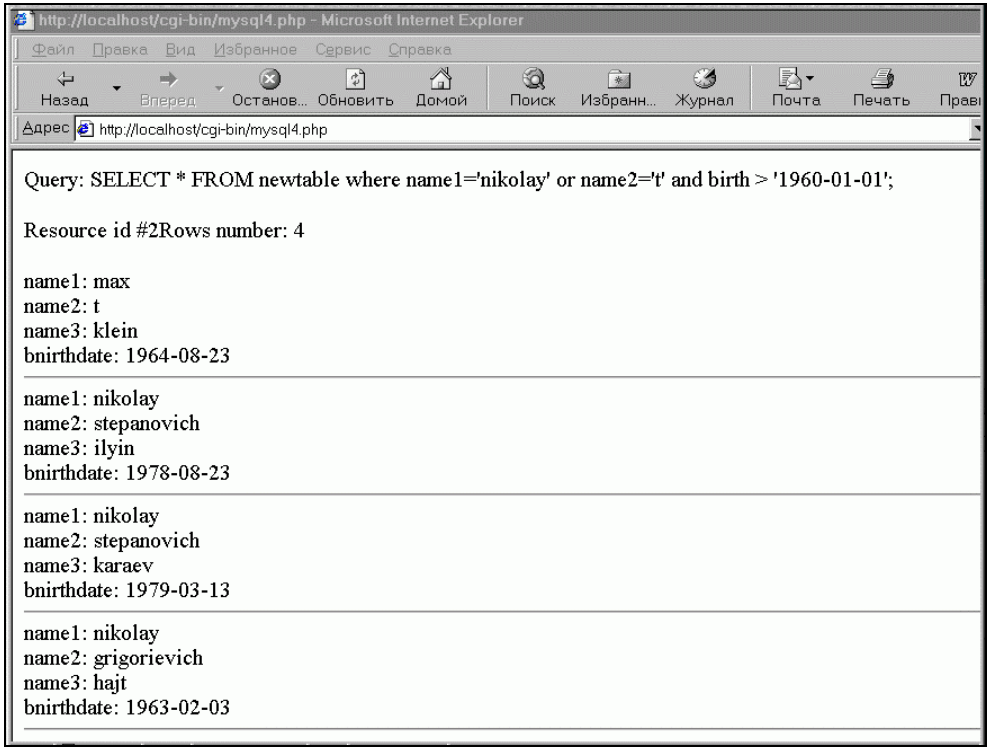

Рис. 4.14. Комбинация логических операторов

Листинг 4.5. Выбор столбца

```
<?php
$c=mysql_connect("localhost:3306", "", "");
mysql_select_db("test",$c);

$stroka = "SELECT name1, birth FROM newtable;";
echo "Query: ".$stroka."<P>";
echo $result = mysql_query($stroka);
$noro=mysql_num_rows($result);
echo "Rows number: ".$noro."<P>";
while ($row = mysql_fetch_array ($result))
{
for ($l=0; $l < 2; $l++)
{
echo $row[$l]."   ";
}
```

```
echo "<br>\n";
}
?>
```

Результат работы показан на рис. 4.15.

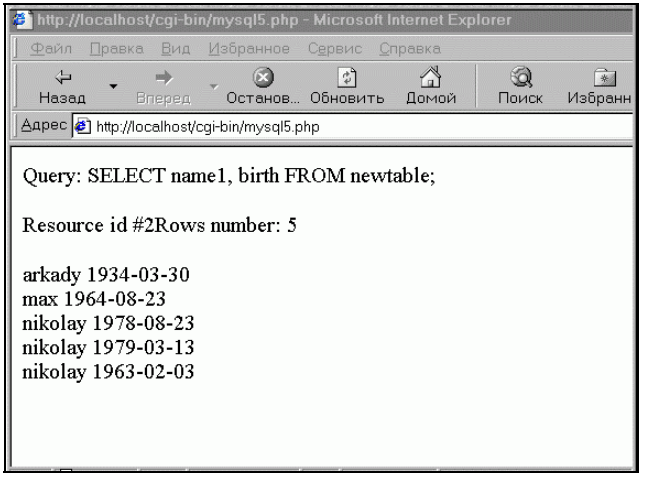

Рис. 4.15. Выбор отдельных столбцов

Чтобы выбрать столбцы только с уникальными значениями, делаем такой запрос:

```
$stroka = "SELECT DISTINCT name1 FROM newtable;";
```

При этом не забудем изменить значение параметра `$l=1`.

Результат мы видим на рис. 4.16.

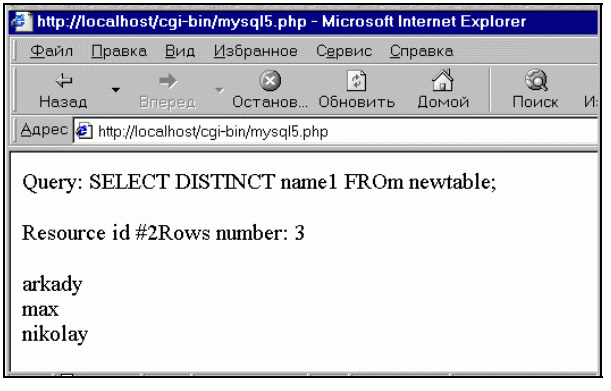

Рис. 4.16. Только уникальные значения (нет повторов)

Сортировка строк результата

MySQL располагает встроенными возможностями сортировки результата, полученного по SQL-запросу. Отсортируем записи по значению поля `birth` (по дате рождения).

```
$stroka = "SELECT  name1, birth FROM newtable order by birth;";
```

Результат показан на рис. 4.17.

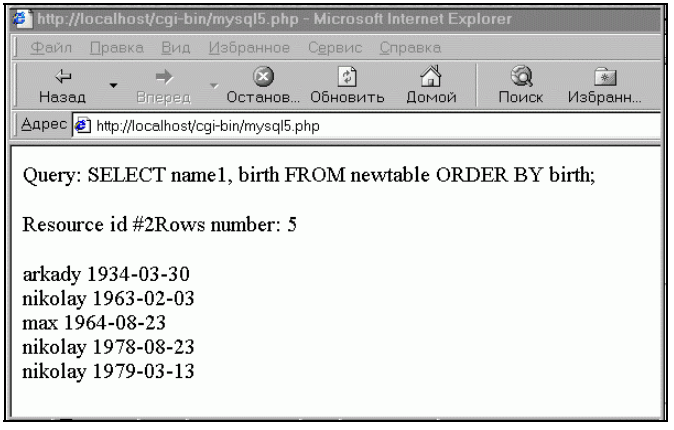

Рис. 4.17. Сортировка строк результата

Сортировка может быть организована по нескольким индексам:

```
$stroka = "SELECT  * FROm newtable order by name3, birth;";
```

Результат показан на рис. 4.18.

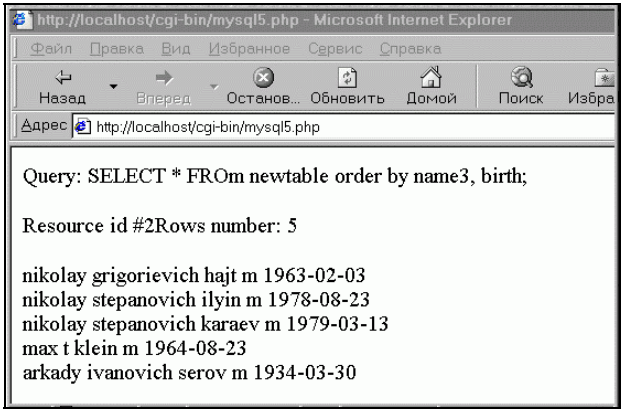

Рис. 4.18. Сортировка по двум параметрам

MySQL позволяет производить математические вычисления с данными. Можно, например, вычислить возраст персонажей из таблицы newtable, выразив его в количестве дней:

```
$stroka = "SELECT name3, (TO_DAYS(NOW())-TO_DAYS(birth)) FROM newtable;";
```

Результат показан на рис. 4.19.

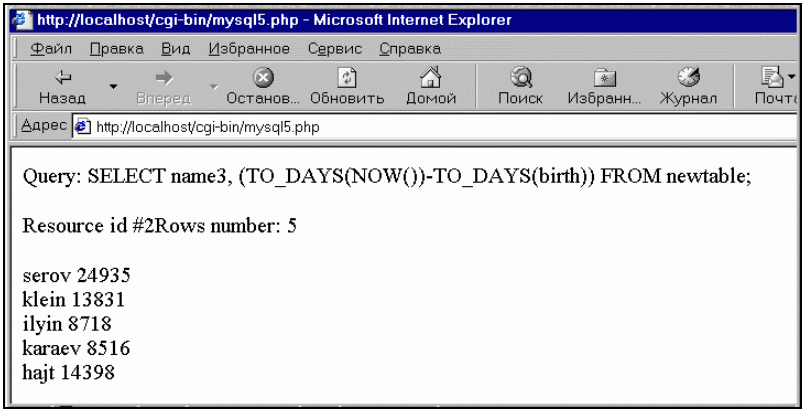

Рис. 4.19. Математические вычисления, производимые средствами MySQL

Несколько более усовершенствованный вариант этого запроса может выглядеть так:

```
$stroka = "SELECT name3, (TO_DAYS(NOW())-TO_DAYS(birth))/365 as age FROM
newtable order by age;";
```

В результате получим возраст, выраженный в годах (рис. 4.20).

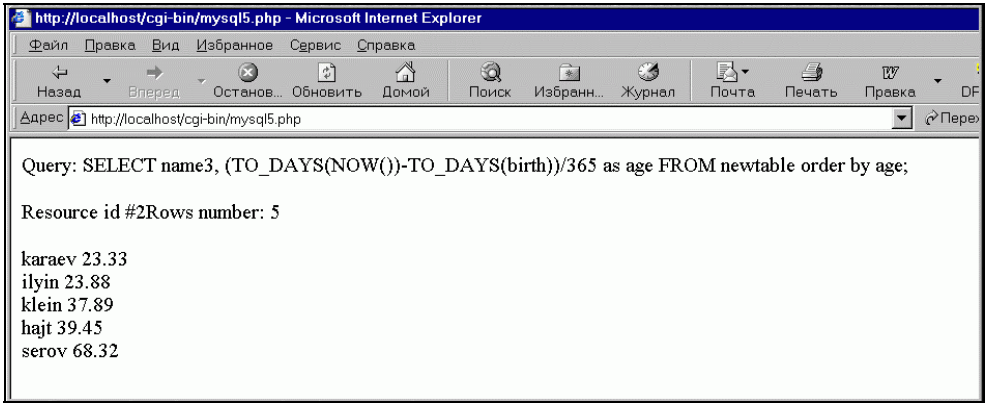

Рис. 4.20. Усовершенствованные условия вычислений и сортировки

Рассмотрим работу с текстовыми моделями. Запрос для выбора строк, начинающихся с буквы k (знак % соответствует произвольному количеству любых символов, знак _ соответствует одному произвольному символу).

```
$stroka = "SELECT * FROM newtable WHERE name3 LIKE 'k%';";
```

Все имена, начинающиеся на k, будут выбраны (результат показан на рис. 4.21).

```
$stroka = "SELECT * FROM newtable WHERE name3 LIKE '%in';";
```

Выбираются все имена, оканчивающиеся на in (результат мы видим на рис. 4.22).

```
$stroka = "SELECT * FROM newtable WHERE name3 LIKE '%a%';";
```

Выбираются имена, содержащие букву a (результат приведен на рис. 4.23).

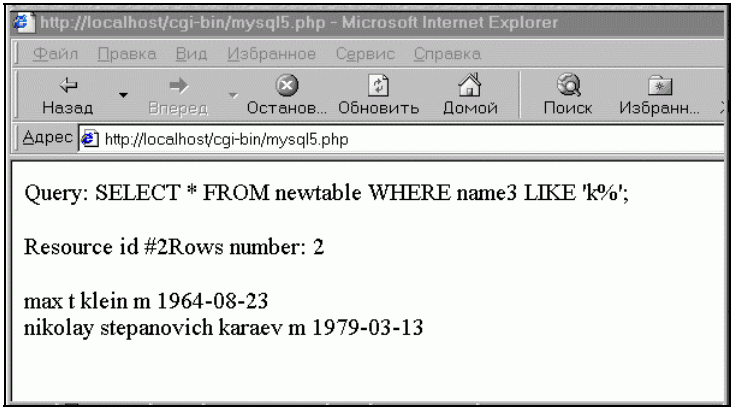

Рис. 4.21. Выбор фамилии на букву k

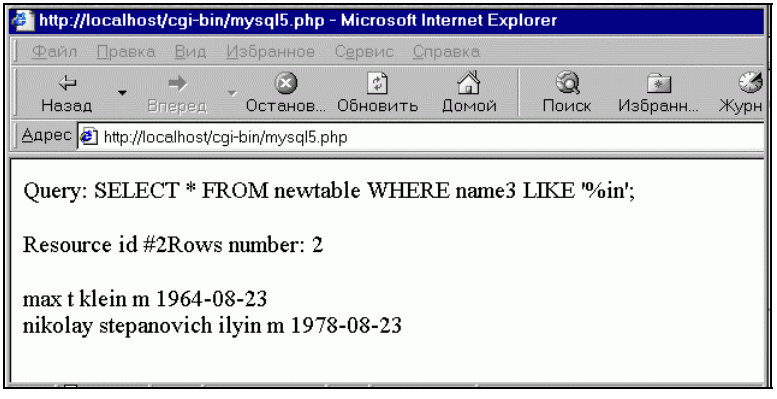

Рис. 4.22. Выбор фамилии, оканчивающейся на in

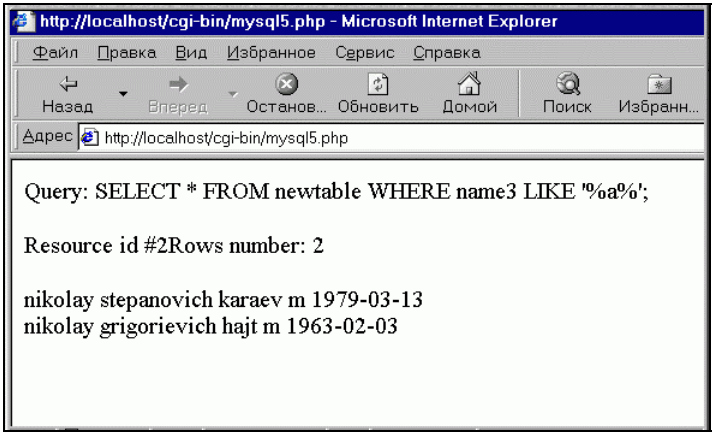

Рис. 4.23. Выбор фамилии, содержащей букву a

Запрос:

```
SELECT * FROM newtable WHERE name3 LIKE "_____";
```

выводит все имена, состоящие из 5 символов (рис. 4.24).

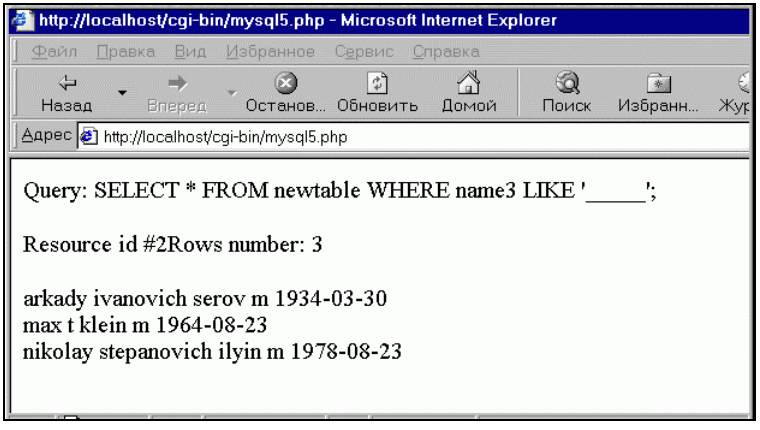

Рис. 4.24. Выбор имен по длине

Смысл некоторых стандартных моделей такой:

- `'.'` — один любой символ;
- `'[...]'` — любой символ из тех, что перечислены между квадратными скобками. Например, записи `'[abc]'` соответствует либо `'a'`, либо `'b'`, либо `'c'`. Можно указать диапазон символов, например, запись `'[a-z]'` соответствует любому строчному символу `'a'`, а запись `[0-9]'` — любой цифре;

- `'*'` — соответствует нескольким (в том числе и 0) символам, стоящим перед знаком звездочки. Например, запись `'x*'` соответствует произвольному количеству букв x, `'[0-9]*'` — произвольному количеству цифр, а запись `'.*'` соответствует произвольному количеству любых символов;
- следует иметь в виду, что модели регулярных выражений являются чувствительными к регистру. Это можно учесть, указав, например, `'[aA]'`, что соответствует как прописной, так и строчной букве a. Конструкция `'[a-zA-Z]'` соответствует всем буквам — и строчным, и прописным;
- паттерн (шаблон для поиска) соответствует модели, если он встречается в любом месте. Чтобы ограничить себя поиском только тех паттернов, что располагаются в начале, используется символ `'^'`, для тех паттернов, что располагаются в конце, следует использовать символ `'$'`.

В запросах при использовании моделей регулярных выражений нужно вставлять слово `REGEXP`.

```
$stroka = "SELECT * FROM newtable WHERE name3 REGEXP '^k';";
```

На рис. 4.25 показан пример работы с регулярными выражениями.

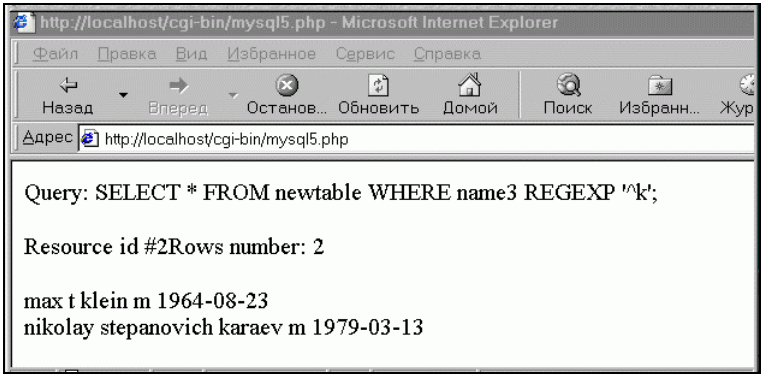

Рис. 4.25. Пользуемся регулярными выражениями при запросах к MySQL

Использование нескольких таблиц одновременно

Допустим, необходимо собирать информацию, касающуюся тех персонажей, что описаны в таблице newtable, и хранить эту информацию в новой таблице. Это может быть, например, информация об определенных событиях в жизни лиц, перечисленных в таблице newtable. Новая таблица может содержать имя человека, к которому относится событие, дату события, описание события, тип события (для категоризации событий).

Создадим новую таблицу event:

```
mysql> CREATE TABLE event (name3 VARCHAR(20), date DATE,
type VARCHAR(15), remark VARCHAR(255));
```

Заполним таблицу такими событиями (табл. 4.1).

Таблица 4.1. *Заполняем базу данных*

Name3	Date	Type	Remark
Klein	2002-07-15	First	Arrived
Klein	2002-07-16	Second	Departed alone
Klein	2002-07-17	First	Arrived with cargo
Karaev	2002-07-16	Third	Awaiting
Hajt	2002-07-17	Second	Departed with cargo
Serov	2002_07-10	Other	
Ilyin	2002-07-16	Other	
Ilyin	2002-07-17	Fourth	admitted

Сохраним эту информацию в текстовом файле (листинг 4.6).

Листинг 4.6. Файл event.txt

```
Klein      2002-07-15     First      Arrived
Klein      2002-07-16     Second     Departed alone
Klein      2002-07-17     First      Arrived with cargo
Karaev     2002-07-16     Third      Awaiting
Hajt       2002-07-17     Second     Departed with cargo
Serov      2002_07-10     other
Ilyin      2002-07-16     other
Ilyin      2002-07-17     Fourth     admitted,
```

Назовем файл event.txt. Вставим данные в базу данных их консольного клиента:

```
mysql> LOAD DATA LOCAL INFILE "event.txt" INTO TABLE event;
```

Сейчас мы несколько усовершенствуем интерфейс, сделаем вывод немного более красивым (листинг 4.7).

Листинг 4.7. Усовершенствованный вывод информации

```
<?php
$c=mysql_connect("localhost:3306", "olymp", "vad");
mysql_select_db("newdb",$c);
```

```
$stroka = "SELECT * FROM event";
echo "Query: ".$stroka."<P>";
echo $result = mysql_query($stroka);
$noro=mysql_num_rows($result);
echo "Rows number: ".$noro."<P>";

echo '<table border=1>';

while ($row = mysql_fetch_array ($result))
{

for ($l=0; $l < 4; $l++)
{
echo '<tr>';
echo '<td>';
echo $row[$l]."   </td>";
echo "</tr>";
}

}
echo '</table>';
?>
```

Результат ввода данных в новую таблицу виден на рис. 4.26.

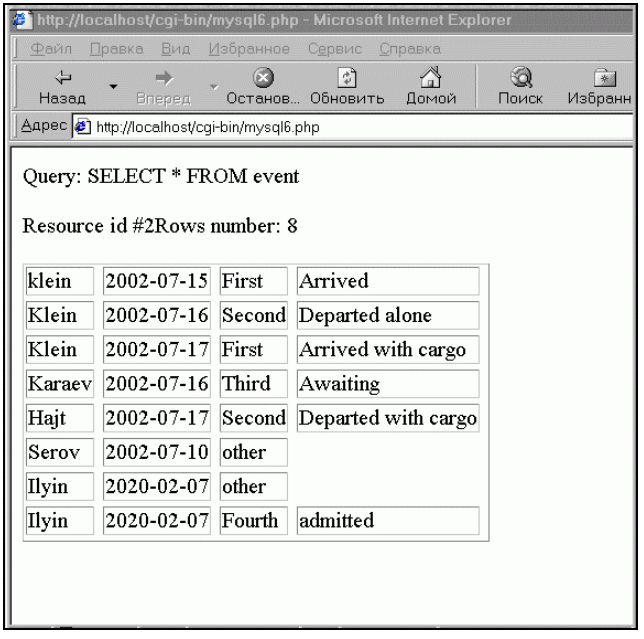

Рис. 4.26. Вывод таблицы из базы данных

Поиск по нескольким таблицам можно организовать с использованием таких запросов:

```
$stroka = "SELECT newtable.name3, name1, data, rematk FROM newtable,
event;";
```

Результат показан на рис. 4.27.

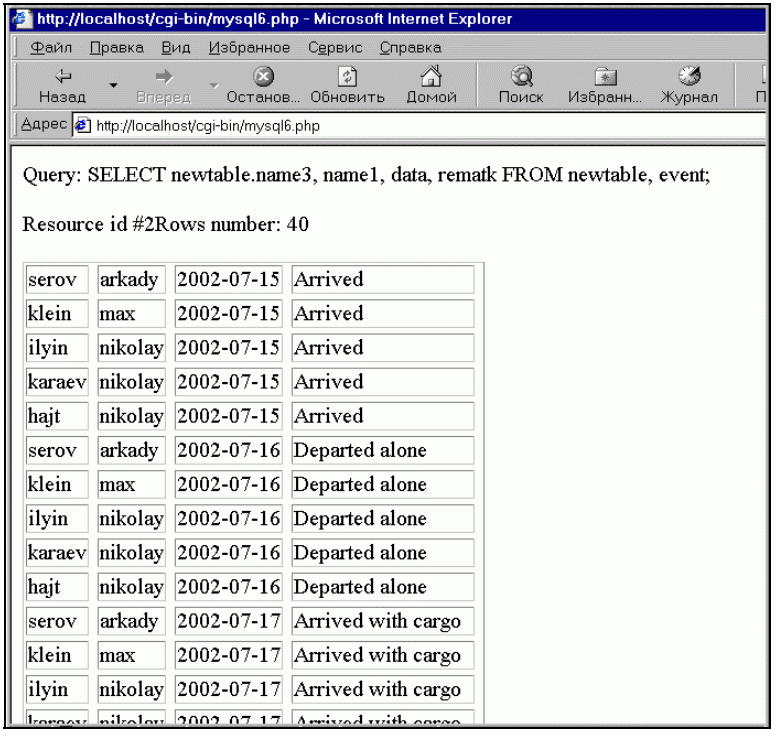

Рис. 4.27. Одновременный запрос к нескольким таблицам

Получаем информацию о базах данных и таблицах

Для того чтобы, в случае необходимости (например, если вы забыли имена баз данных), узнать, какие базы данных существуют на сервере MySQL, можно воспользоваться таким запросом:

```
$stroka = "show databases;";
```

Результат показан на рис. 4.28.

Можно воспользоваться функцией `database()`, которая выдаст название текущей базы данных:

```
$stroka = "SELECT database();";
```

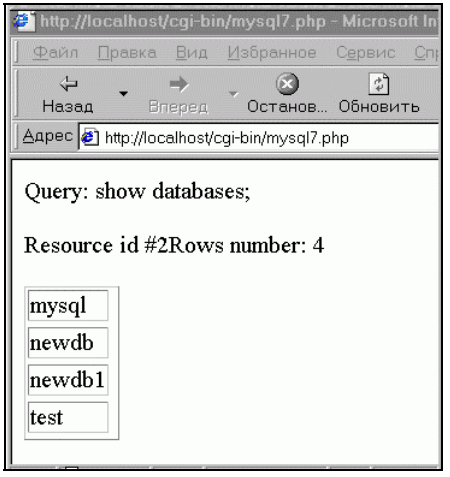

Рис. 4.28. Базы данных на сервере MySQL

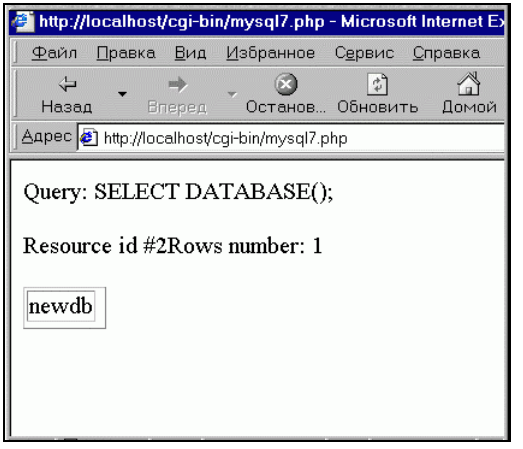

Рис. 4.29. Результат работы функции `database()`

На рис. 4.29 показан результат, выведенный в окно браузера.

Если на момент выполнения функции `database()` база данных еще не выбрана, то результат выполнения такой функции будет пустым.

Для того чтобы узнать, какие таблицы включены в выбранную базу данных, используем запрос:

```
$stroka = "SHOW TABLES;";
```

Выведенная информация представлена на рис. 4.30.

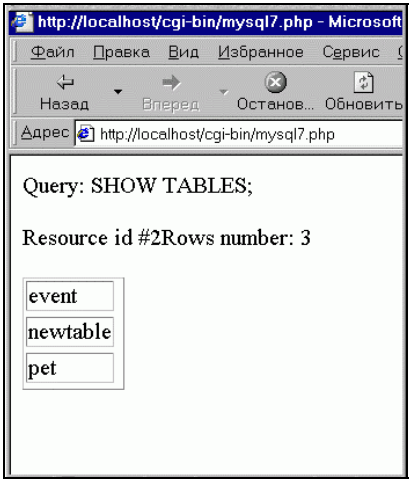

Рис. 4.30. Структура базы данных newdb

Чтобы получить описание таблицы, воспользуемся таким запросом (листинг 4.8).

Листинг 4.8. Запрос на описание таблицы

```
<?php
$c=mysql_connect("localhost:3306", "", "");
mysql_select_db("test",$c);

$stroka = "DESCRIBE event;";
echo "Query: ".$stroka."<P>";
echo $result = mysql_query($stroka);
$noro=mysql_num_rows($result);
$nofields=mysql_num_fields($result);
echo "Rows number: ".$noro."<P>";

echo '<table border=1>';

while ($row = mysql_fetch_array ($result))
{
 echo '<tr>';
for ($l=0; $l < $nofields; $l++)
{   echo '<td>';
echo $row[$l]."   </td><td>";
}
echo "</tr><tr>";
}
echo '</table>';
?>
```

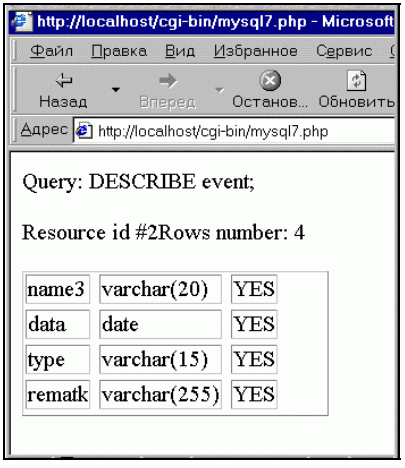

Рис. 4.31. Структура таблицы

Здесь мы слегка изменили файл, вставив функцию `mysql_num_fields()`, чтобы далее нам не пришлось уделять специальное внимание тому, сколько будет столбцов в результате. Результат показан на рис. 4.31.

Некоторые классические примеры

Классический пример создания таблицы из пособия, приведенного в документации по MySQL:

```
CREATE TABLE shop (
 article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
 dealer  CHAR(20)                 DEFAULT ''     NOT NULL,
 price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
 PRIMARY KEY(article, dealer));

INSERT INTO shop VALUES
(1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),(3,'C',1.69),
(3,'D',1.25),(4,'D',19.95);
```

Обычный MySQL запрос выглядит так:

```
SELECT * FROM shop;
```

В PHP-варианте мы его запишем так:

```
$stroka = "CREATE TABLE shop (article INT(4) UNSIGNED ZEROFILL DEFAULT
'0000' NOT NULL, dealer  CHAR(20) DEFAULT '' NOT NULL, price DOUBLE(16,2)
DEFAULT '0.00' NOT NULL, PRIMARY KEY(article, dealer));";
echo "Query: ".$stroka."<P>";
echo $result = mysql_query($stroka);
$stroka = "INSERT INTO shop VALUES
(1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),(3,'C',1.69),
(3,'D',1.25),(4,'D',19.95);";
echo "Query: ".$stroka."<P>";
echo $result = mysql_query($stroka);
$stroka = "SELECT * FROM shop;";
```

Результат этих запросов показан на рис. 4.32.

Выбор максимального значения

```
$stroka = "SELECT MAX(article) AS article FROM shop";
```

Результат можно увидеть на рис. 4.33.

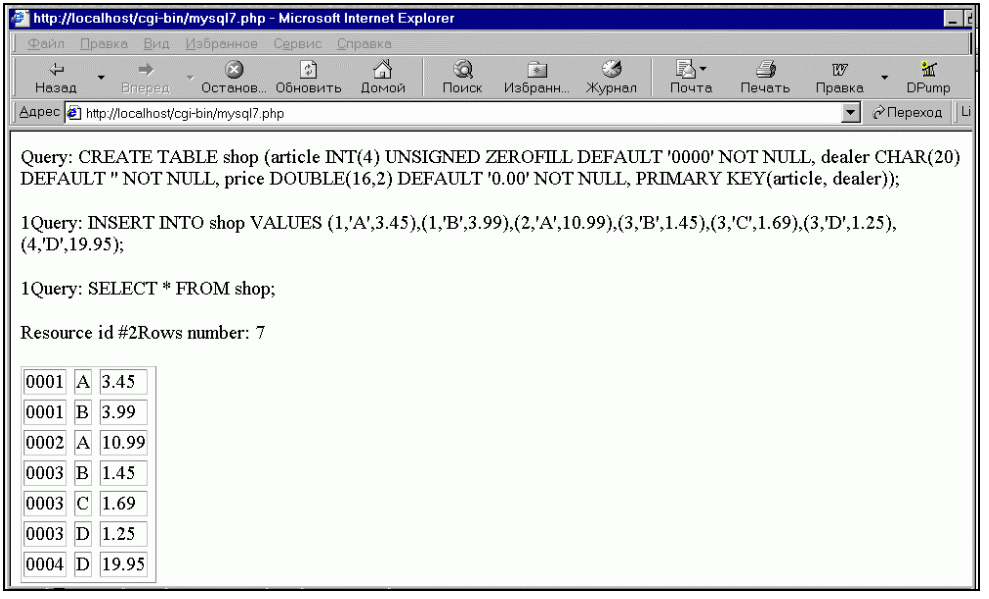

Рис. 4.32. Результаты запросов

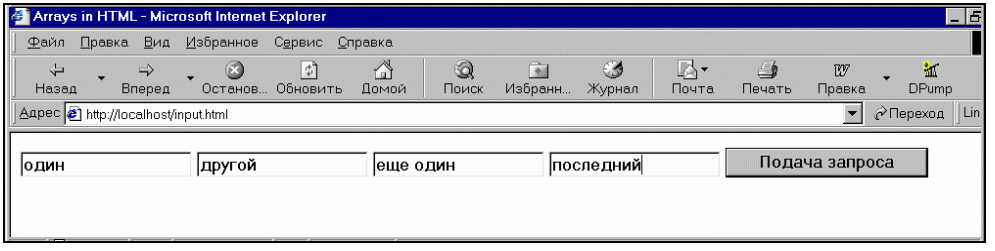

Рис. 4.33. Выбираем максимальное значение

Строка, содержащая максимальное значение из определенного столбца

Вывод строки, содержащей максимальное значение, может быть полезен, например, при поиске наиболее дорогого товара. В MySQL такая задача распадается на два шага.

1. Нахождение максимальной цены с использованием запроса `SELECT`.
2. Использование полученного значения цены в новом запросе, например:

```
SELECT article, dealer, price
  FROM   shop
  WHERE  price=19.95
```

Другой способ решения задачи — отсортировать в убывающем порядке результат по цене и взять первую строку результата с использованием условия LIMIT:

```
SELECT article, dealer, price
  FROM   shop
  ORDER BY price DESC
  LIMIT 1
```

В PHP-варианте этот запрос выглядит так:

```
$stroka = "SELECT article, dealer, price FROM   shop ORDER BY price DESC
LIMIT 1";
```

Если есть несколько единиц товара с одинаковой максимальной ценой, то этот запрос вернет только один из них.

Результат выполнения запроса показан на рис. 4.34.

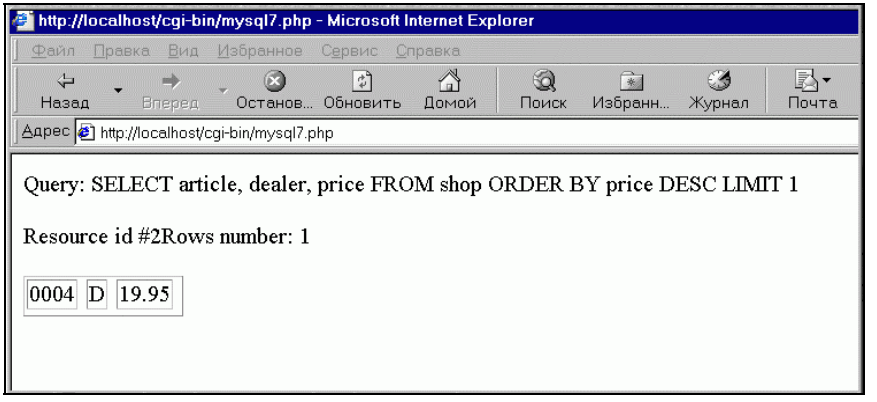

Рис. 4.34. Выбираем самый дорогой товар

Еще один вариант запроса.

```
 $stroka = "SELECT article, MAX(price) AS price FROM   shop GROUP BY
article;";
```

Результат показан на рис. 4.35.

MySQL позволяет создавать *пользовательские переменные*. В таких переменных можно хранить промежуточные значения, не прибегая к необходимости хранить их где-то средствами PHP-программы или другими способами.

```
$stroka0 = "select @min_price:=min(price),@max_price:=max(price)
from shop;";
$stroka = "select * from shop where price=@min_price or
price=@max_price;";
```

На рис. 4.36 показан результат выполнения такого запроса.

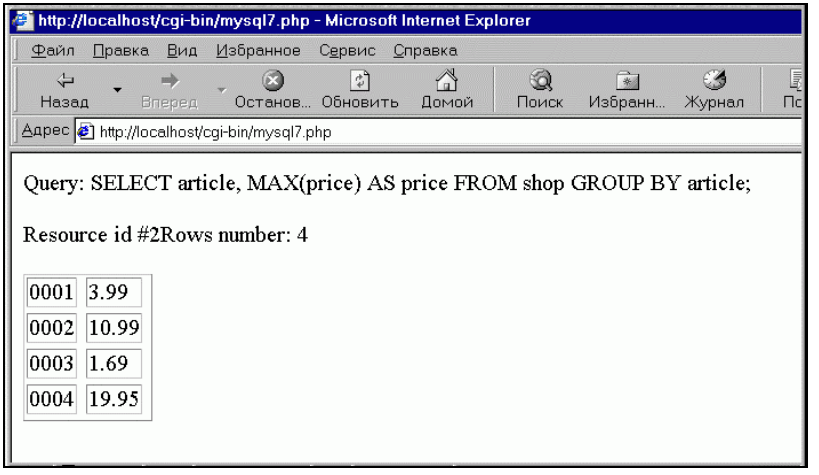

Рис 4.35. Результат выведен в окне браузера

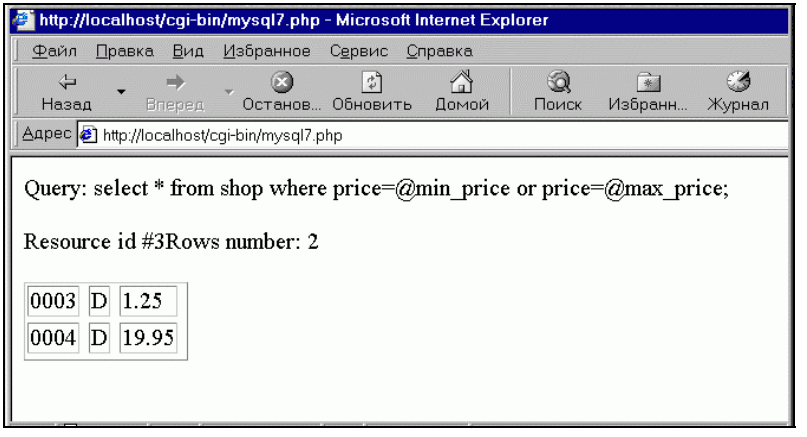

Рис 4.36. Результат запроса с пользовательскими переменными

На протяжении всего раздела мы поступали не самым оптимальным способом — каждый раз меняли содержимое PHP-файла вручную. Чтобы оптимизировать процесс, удобнее создать HTML-файл с формой, куда записывается запрос, а затем передать запрос основному PHP-файлу нажатием кнопки. Простейшая форма может быть описана в виде файла mysqlquery.html (листинг 4.9).

Листинг 4.9. Файл mysqlquery.html

```
<html>
<head>
```

```
<title>MySQL Query From </title>
<body>
<h1>My SQL Query GORM.</H1>
<p>Пожалуйста, введите ваш запрос
<P><form action ='http://localhost/cgi-bin/mysql8.php'>
<input type=text name="query">
<input type=submit>
</form>
</body>
</html>
```

В окне браузера мы увидим картинку, показанную на рис. 4.37.

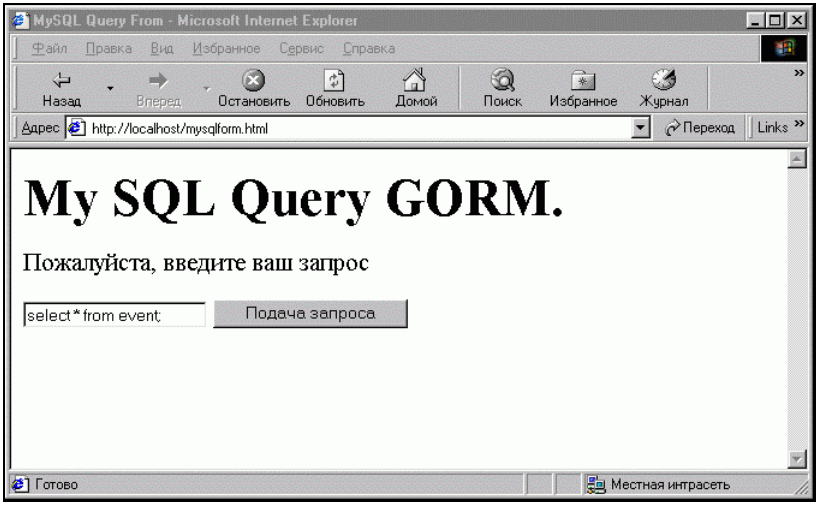

Рис. 4.37. Запрос передается серверу посредством HTML-форм

Файл с PHP-программой немного будет изменен (листинг 4.10).

Листинг 4.10. Файл mysql8.php

```
<?php
$c=mysql_connect("localhost:3306", "", "");
mysql_select_db("test",$c);
echo "Query: ".$query."<P>";
echo $result = mysql_query($query);
$noro=mysql_num_rows($result);
$nofields=mysql_num_fields($result);
echo " Rows number: ".$noro."<P>";
echo '<table border=1>';
```

```
while ($row = mysql_fetch_array ($result))
{
 echo '<tr>';
for ($l=0; $l < $nofields; $l++)
{   echo '<td>';
echo $row[$l]."   </td><td>";
}
echo "</tr><tr>";
 }
echo '</table>';
?>
```

После отправки запроса сервер через секунду-другую возвратит результат (рис. 4.38).

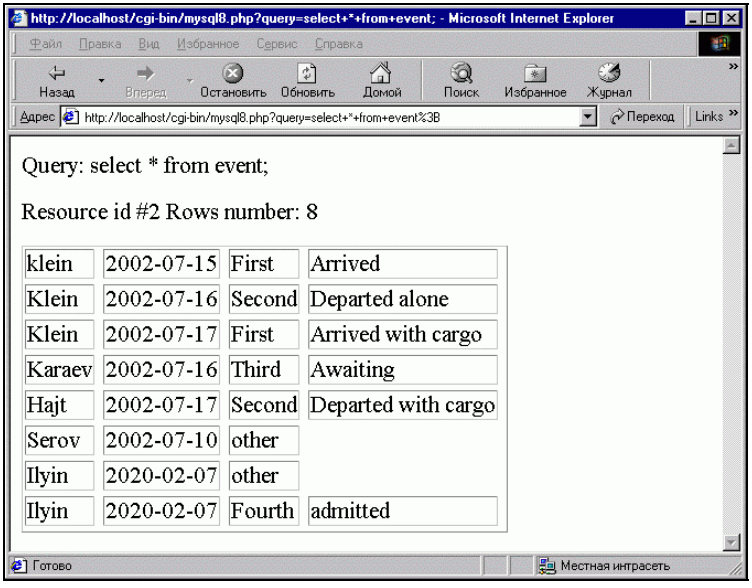

Рис. 4.38. Скрипт выполнен

Язык XML

Язык XML (Extensible Markup Language — расширяемый язык разметки) был создан для описания данных. Этим он сильно отличается от языка HTML, который используется для описания того, как следует отобразить (разместить) данные.

В этом языке, как и в языке HTML, используются тэги, но набор тэгов не является фиксированным: разработчик сам определяет, какие тэги он будет

использовать. Для описания данных применяются языки описания типа документа DTD (Document Type Definiltion) или язык XML Schema.

Язык XML создан не для того, чтобы прийти на смену языку HTML. Сам по себе язык XML не используется для выполнения каких-либо функций, единственная его задача — описание данных. При помощи XML создается структура документа, содержащего данные. Это могут быть, например, файлы, хранящие почтовые сообщения. Пример текста на языке XML приведен в листинге 4.11.

Листинг 4.11. Пример XML-текста

```
<soobschenie>
<to>Petya</to>
<from>Vasya</from>
<heading>Reminder</heading>
<body>
Don't forget me this weekend!
</body>
</soobchenie>
```

В этом примере сообщение имеет тело (body), заголовок (header), а также содержит информацию об отправителе и получателе сообщения. Еще раз отметим, что использованная разметка сама по себе не будет выполнять никаких действий. Информация о структуре документа и назначении данных, хранящихся в нем, записана при помощи произвольных тэгов. Чтобы обработать информацию, необходимо предварительно позаботиться о соответствующем программном обеспечении, которое будет понимать назначение тех или иных используемых в тексте XML-элементов, и соответствующим образом обрабатывать содержащуюся в них информацию.

Набор тэгов XML не является заранее определенным. Разработчик может изобретать любые тэги, какие ему придутся по вкусу. Автор сам создает тэги и конструирует структуру документа. Тэги, которые мы использовали в листинге 4.11, например `<to>` и `<from>`, не определены в стандарте XML. Мы выбрали такие тэги только потому, что нам так понравилось.

Популярность XML растет. С помощью XML мы имеем возможность отделить данные от описания способов представления этих данных. При этом данные можно хранить отдельно от HTML в XML-файлах. XML-данные также могут быть расположены и внутри HTML-файлов. Но сам HTML попрежнему концентрируется на описании способов отображения данных.

Формат XML может быть использован для передачи данных между несовместимыми системами. Кроме того, это наиболее удобный формат для обмена коммерческой информацией между предприятиями. Он полезен при

работе с данными общего доступа, при необходимости длительного хранения данных. При этом одни и те же данные могут быть представлены поразному, в зависимости от требований конкретного пользователя.

Важное достоинство XML состоит в том, что на его основе могут быть созданы другие языки. В частности, язык WML (Wireless Markup Language — язык разметки для беспроводных систем), который используется для создания приложений, работающих с мобильными телефонами и другими портативными устройствами, написан с применением правил XML.

Правила синтаксиса языка XML очень просты, но одновременно с этим строги. Правила эти легко запомнить и несложно соблюдать.

Начнем с примера. Фрагмент XML-кода, приведенный выше, не является законченным XML-документом. Его необходимо немного изменить. Исправленный текст представляет собой законченный документ XML (листинг 4.12).

Листинг 4.12. Пример XML-документа

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soobschenie>
<to>Petya</to>
<from>Vasya</from>
<heading>Napominanie</heading>
<body>
Ne zabyd!
</body>
</soobschenie>
```

Первая строка сообщает о том, что перед нами XML-документ. В следующей строке использован тэг `<soobschenie>`, откуда становится понятно, что перед нами файл с сообщением. Это корневой элемент документа. Последующие четыре тэга описывают четыре дочерних элемента (по отношению к корневому элементу):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>
Don't forget me this weekend!
</body>
```

Последним в файле используется закрывающий тэг `</soobschenie>`. Каждый элемент должен иметь закрывающий тэг. В HTML мы можем написать, например, так:

```
<p>Это параграф
<p>А это другой параграф
```

В XML такая запись невозможна. Мы должны использовать закрывающие тэги, поэтому с точки зрения правил языка XML следует писать так:

```
<p>Это параграф
</p>
<p>А это другой параграф
</p>
```

В предыдущем примере первый тэг, который сообщает о том, что перед нами XML-документ, не имеет парного закрывающего тэга. Это не ошибка. Этот тэг, вообще говоря, не является частью XML-документа, он только сообщает о том, что файл содержит XML-документ.

Тэги XML чувствительны к регистру, скажем, тэг `<tag>` отличается от тэга `<Tag>`. Открывающие и закрывающие тэги должны использовать буквы, написанные в одном регистре, например:

```
<Stroka>Это ошибка</stroka>
<stroka>А это правильно</stroka>
```

Еще одно правило XML гласит о том, что элементы XML должны быть правильно вложены в соответствующие родительские элементы. В HTML мы могли бы написать, скажем,

```
<b><i>Текст полужирным курсивом</b></i>
```

Но в XML такое непозволительно, следует писать так:

```
<b><I>Текст полужирным курсивом</I></b>
```

Все XML-документы обязаны иметь корневой тэг. Каждый документ должен иметь пару тэгов, которые соответствуют корневому элементу. Все прочие элементы должны быть вложены в этот корневой элемент. Любой элемент XML может иметь собственные дочерние элементы.

```
<kornevoj>
<dochernij>
<dochernijdiochernego>
. . .
</dochernijdochernego>
</dochernij>
</kornevoj>
```

Если в тэге содержится атрибут, то не допускается опускание кавычек при задании значения этого атрибута (что допускается в HTML). В листингах 4.13 и 4.14 приведены примеры верного и неверного задания значения атрибута.

Листинг 4.13. Значение атрибута указано неверно

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soobschenie data=10/10/2002>
<to>Petya</to>
<from>Vasya</from>
<heading>Napominanie</heading>
<body>
Ne zabyd!
</body>
</soobschenie>
```

Листинг 4.14. Верно записанное значение атрибута

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soobschenie data="10/10/2002">
<to>Petya</to>
<from>Vasya</from>
<heading>Napominanie</heading>
<body>
Ne zabyd!
</body>
</soobschenie>
```

Здесь, вероятно, будет полезным отметить, что при отображении HTML-документа все пробелы, кроме одного, игнорируются. XML-файл содержит данные, при его обработке пробелы не будут автоматически удаляться. При этом в XML-файле все символы, приводящие к переходу на новую строку, например комбинация `CRLF`, преобразуются в `LF`.

Комментарии в XML оформляются так же, как и в HTML, например:

```
<!-- Это комментарий -->
```

Элементы XML

XML устроен так, что элементы XML можно расширять, при этом старая структура документа будет восприниматься без изменений. Допустим, у нас есть документ, содержащий такой текст:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soobschenie data=10/10/2002>
<to>Petya</to>
<from>Vasya</from>
```

```
<body>
Ne zabyd!
</body>
</soobschenie>
```

Этот текст может быть использован какой-либо программой для отображения содержащихся в нем данных. Расширим исходный файл следующим образом:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<soobschenie data=10/10/2002>
<to>Petya</to>
<from>Vasya</from>
<heading>Napominanie</heading>
<body>
Ne zabyd!
</body>
</soobschenie>
```

Несмотря на изменения, отображение не должно измениться. Иными словами, от внесенных в файл изменений не должен измениться метод использования старых элементов.

Тот факт, что XML-документ может быть расширен, оказывается весьма полезным.

Чтобы адекватно описывать элементы, необходимо договориться о терминах. Предположим, что у нас есть книга, структуру которой можно описать с помощью XML следующим образом (листинг 4.15).

Листинг 4.15. Структура книги

```
<book>
<title>Что такое XML</title>
<prodid="1234567889" media="bumaga"></prod>
<chapter>Введение в XML
<para>Что такое HTML</para>
<para>Что такое XML</para>
</chapter>
<chapter>Синтаксис XML
<para>Элемент должен содержать закрывающий тэг</para>
<para>Элементы должны быть корректно вложены</para>
</chapter>
</book>
```

Родительские элементы имеют дочерние элементы, вложенные в родительские элементы. Дочерние элементы имеют родительские элементы, которые

их содержат. Такие элементы, как, например, <title>, <prod>, <chapter> являются *смежными* элементами, они имеют один и тот же родительский элемент.

Элемент — это все, что начинается с открывающего тэга и заканчивается закрывающим тэгом. В состав элемента входит открывающий тэг, закрывающий тэг и все, что расположено между этими тэгами. То, что расположено между тэгами, называется *содержимым тэга*. В качестве содержимого тэга могут выступать другие элементы. Кроме того, содержимое тэга может быть: *смешанным содержимым*, *простым содержимым* или быть *пустым* (содержимое отсутствует). Элемент может иметь *атрибуты*, значение которых, как правило, задается в открывающем элемент тэге.

В приведенном выше примере элемент <book> имеет непустое содержимое, поскольку в него вложены другие элементы. Элемент <chapter> имеет смешанное содержимое, т. к. он содержит как текст, так и другие элементы. Элементы <para> имеют простое содержимое (или текстовое содержимое), поскольку в этих элементах содержится только текст. Элемент <prod> не имеет содержимого. Только один элемент содержит атрибуты. Это элемент <prod>. Атрибут prodid имеет значение 1234567889, атрибут media — значение "bumaga".

Имена элементов

Имена элементов должны подчиняться определенным правилам:

- имена могут содержать буквы, числа и другие символы;
- имена не могут начинаться с цифры или знака препинания;
- имена не могут начинаться с последовательности букв xml, XML или Xml;
- Имена не могут содержать пробелы.

При создании имен полезно выбирать такие имена, которые несут в себе поясняющий смысл. Рекомендуется избегать применения знаков – и ., т. к. эти символы могут приводить к ошибкам, когда имена анализируются при помощи программ. Имена могут быть сколь угодно длинными, но не следует использовать слишком сложные имена. Если в XML используются символы букв, не входящих в английский алфавит, то необходимо быть уверенным в том, что используемое программное обеспечение может работать с такими символами. Не следует использовать в именах символ двоеточия ":", т. к. он применяется при работе с пространствами имен.

Атрибуты XML

Как и в HTML, в XML разработчик имеет возможность задавать атрибуты. Атрибуты используются для задания дополнительной информации, связан-

ной с тем элементом, к которому они относятся. Так, например, в тэге `<img>` мы можем указать значение атрибута `src`.

```
<img src="picture.jpg">
```

Еще один простой пример:

```
<a href="ssylka.php">
```

Информация, содержащаяся в атрибутах, не является частью данных. Так, в приведенном далее примере указанный тип никак не связан с самими данными, тип может быть использован для вызова соответствующего программного обеспечения, работающего с указанными данными:

```
<file type="jpg">picture.jpg</file>
```

Значение атрибута всегда должно быть помещено в кавычки. При этом полезно помнить, что тип кавычек (одиночные или двойные) не имеет значения, но тип закрывающих кавычек должен соответствовать типу открывающих кавычек. Запись

```
<tag atribut='znachenie'>
```

эквивалентна записи

```
<tag atribut="znachenie">
```

Кавычки можно вкладывать друг в друга, например:

```
<Mafiosi imya='Pasha "Kosoy" Svistunov'>
```

Информация, которая указывается в качестве значений атрибутов, может быть вложена в тело элемента с использованием соответствующего дочернего элемента. Можно написать, например, так:

```
<chelovek pol= "жен">
<imya>Фекла</imya>
<familiya>Метелкина</familiya>
</chelovek>
```

Эту информацию можно передать, используя дочерний элемент:

```
<chelovek>
<pol>жен</pol>
<imya>Фекла</imya>
<familiya>Метелкина</familiya>
</chelovek>
```

В первом случае пол — это атрибут, а во втором случае — это дочерний элемент. Оба примера, тем не менее, содержат одну информацию. Некоторые считают, что использование атрибутов более свойственно HTML, а в XML удобнее пользоваться дочерними элементами.

Это значит, что, например, вместо

```
<pismo date="12/11/02">
<to>Tebe</to>
<from>Otmenya</from>
<heading>Napominanie</heading>
<body>Ne zabud o vstreche zavtra!</body>
</pismo>
```

порой удобнее писать так:

```
<pismo>
<date>12/11/02</date>
<to>Tebe</to>
<from>Otmenya</from>
<heading>Napominanie</heading>
<body>Ne zabud o vstreche zavtra!</body>
</pismo>
```

Можно записать еще более подробно:

```
<pismo>
<date>
<day>12</day>
<month>11</month>
<year>02</year>
</date>
<to>Tebe</to>
<from>Otmenya</from>
<heading>Napominanie</heading>
<body>Ne zabud o vstreche zavtra!</body>
</pismo>
```

Если все же используются атрибуты, то следует помнить, что:

- ❒ атрибуты не могут содержать несколько значений (но дочерние элементы могут);
- ❒ значения атрибутов в дальнейшем неудобно изменять;
- ❒ атрибутам сложнее управлять из программ;
- ❒ атрибуты сложнее в работе, если применяется DTD.

Вряд ли будет полезен набор атрибутов элемента, когда вся информация "упаковывается" внутри тэга в виде набора атрибутов, как в примере ниже:

```
<pismo
day="12"
month="11"
```

```
year="02"
to="Tebe"
from="Otmenya"
heading="Napominanie"
body="Ne zabud o vstreche zavtra!">
</pismo>
```

В качестве атрибутов полезно указывать идентификаторы:

```
<pismo id="pismo401">
<date>
<day>12</day>
<month>11</month>
<year>02</year>
</date>
<to>Tebe</to>
<from>Otmenya</from>
<heading>Napominanie</heading>
<body>Ne zabud o vstreche zavtra!</body>
</pismo>
 <pismo id="pismo402">
<date>
<day>12</day>
<month>11</month>
<year>02</year>
</date>
<to>Mne</to>
<from>Ottebya</from>
<heading>Re: Napominanie</heading>
<body>Ne zabudu!</body>
</pismo>
```

В этом примере `id` — это идентификатор, счетчик сообщений.

Корректно составленный XML-документ

Если XML-документ соответствует всем правилам синтаксиса XML, то такой документ называется *корректно составленным*. Если корректно составленный документ соответствует заданному описанию типа документа, то такой документ называется *действительным*. Описание типа документа (DTD) хранится в файле DTD.

Вот пример корректно составленного XML-документа:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<pismo id="pismo402">
```

```
<date>
<day>12</day>
<month>11</month>
<year>02</year>
</date>
<to>Mne</to>
<from>Ot tebya</from>
<heading>Re: Napominanie</heading>
<body>Ne zabudu!</body>
</pismo>
```

Действительный документ должен быть не только корректно составленным, но и соответствовать описанию типа документа, на которое имеется ссылка в самом документе:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE pismo SYSTEM "vnutrenniepisma.dtd">
<pismo id="pismo402">
<date>
<day>12</day>
<month>11</month>
<year>02</year>
</date>
<to>Mne</to>
<from>Ottebya</from>
<heading>Re: Napominanie</heading>
<body>Ne zabudu!</body>
</pismo>
```

Описание типа XML-документа

Описание типа XML-документа может быть задано при помощи формата DTD или формата XML Schema.

Начиная с версии 5.0, браузер MS Internet Explorer поддерживает работу с XML. Кроме того, браузер MS Internet Explorer версии 5.0 и выше обладает следующими возможностями:

- поддерживает объектную модель DOM XML;
- позволяет просматривать код XML-документа;
- полностью поддерживает стандарт DTD;
- работает с "островами данных" XML (Data Islands);
- осуществляет XSL-преобразования XML-кода;

- способен отображать XML с применением CSS;
- поддерживает технологию компании Microsoft для работы с моделями Behaviors. Модели Behaviors позволяют отделять скрипты от статической части HTML-страниц.

XML-документ может быть отображен в окне браузера Internet Explorer, при этом тэги элементов будут показаны красным цветом, а весь документ будет отображен в виде дерева, где вложенные элементы будут немного сдвинуты влево по отношению к родительским элементам (рис. 4.39).

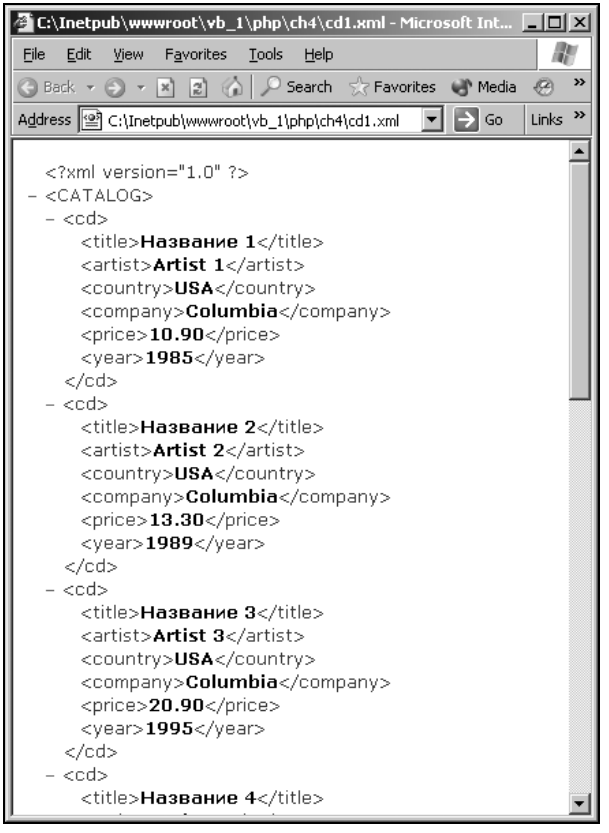

Рис. 4.39. Internet Explorer отобразил корректно составленный XML-документ

Если XML-документ содержит ошибки, то Internet Explorer выдаст сообщение об ошибке (рис. 4.40).

В представленном виде XML-файл вряд ли будет полезен. В качестве примера рассмотрим файл cd1.xml — каталог компакт-дисков (листинг 4.16).

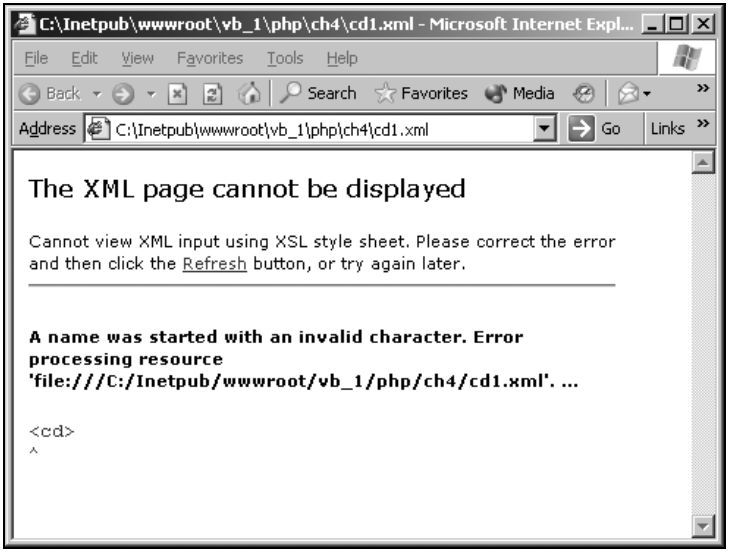

Рис. 4.40. Сообщение об ошибке

Листинг 4.16. Файл cd1.xml

```
<?xml version="1.0"?>
<CATALOG>
<CD>
<TITLE>Название 1</TITLE>
<ARTIST>Артист 1</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Название 2</TITLE>
<ARTIST>Артист 2</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
<CD>
<TITLE>Название 3</TITLE>
<ARTIST>Артист 3</ARTIST>
<COUNTRY>USA</COUNTRY>
```

```
<COMPANY>RCA</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1982</YEAR>
</CD>
<CD>
<TITLE>Название 4</TITLE>
<ARTIST>Артист 4</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>Virgin records</COMPANY>
<PRICE>10.20</PRICE>
<YEAR>1990</YEAR>
</CD>
. . .
. . .
. . .
<CD>
<TITLE>Название 26</TITLE>
<ARTIST>Артист 26</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>EMI</COMPANY>
<PRICE>8.20</PRICE>
<YEAR>1987</YEAR>
</CD>
</CATALOG>
```

Если загрузить этот файл в браузер, то получится простое дерево (см. рис. 4.39).

Информацию, содержащуюся в каталоге cd1.xml, можно представить в гораздо более удобочитаемом виде, если применить листы стилей CSS. Создадим файл cd1.css (листинг 4.17).

Листинг 4.17. Файл cd1.css

```
CATALOG
{background-color: #sbcdef;
width: 100%;}
CD
{display: block;
margin-bottom: 30pt;
margin-left: 0;}
TITLE
{color: #FF1234;
font-size: 20pt;}
```

```
ARTIST
{color: #1234FF;
font-size: 20pt;}
COUNTRY,PRICE,YEAR,COMPANY
{Display: block;
color: #333333;
margin-left: 20pt;}
```

Чтобы отобразить файл cd1.xml с использованием этих стилей, необходимо немного изменить исходный файл cd1.xml, вставив в него всего одну строку, указывающую на файл, содержащий описание стилей:

```
<?xml-stylesheet type="text/css" href="cd1.css"?>
```

Назовем новый файл cd1_css.xml (листинг 4.18).

Листинг 4.18. Файл cd1_css.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="cd1.css"?>
<CATALOG>
<CD>
<TITLE>Nazvanie 1</TITLE>
<ARTIST>Artist 1</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
<CD>
<TITLE>Nazvanie 2</TITLE>
<ARTIST>Artist 2</ARTIST>
<COUNTRY>UK</COUNTRY>
<COMPANY>CBS Records</COMPANY>
<PRICE>9.90</PRICE>
<YEAR>1988</YEAR>
</CD>
. . .
. . .
. . .
<CD>
<TITLE>Nazvanie 26</TITLE>
<ARTIST>Artist 26</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>EMI</COMPANY>
```

```
<PRICE>8.20</PRICE>
<YEAR>1987</YEAR>
</CD>
</CATALOG>
```

Сейчас при загрузке файла cd1.xml в браузер будут применены стили (рис. 4.41). Все примеры созданы для работы с браузером Internet Explorer версии 5 или 6.

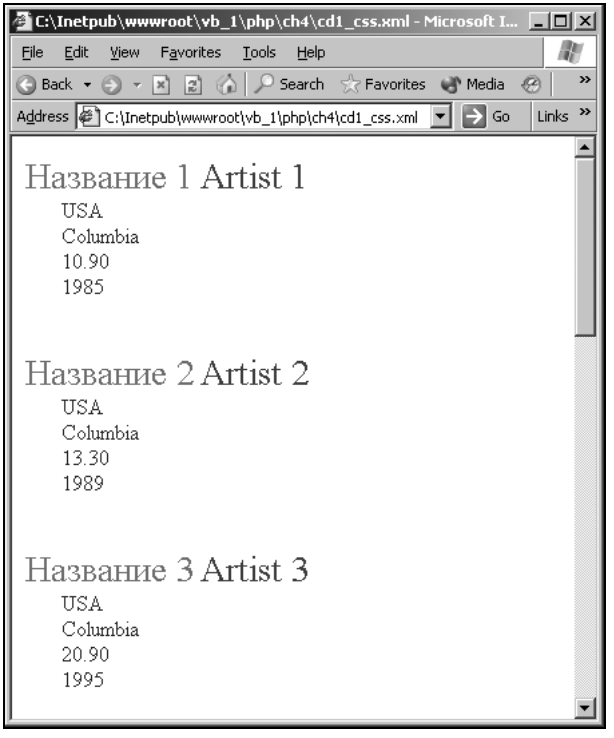

Рис. 4.41. При отображении XML-файла использованы стили CSS

XML-файл может быть отображен браузером Internet Explorer с использованием средств языка XSL. Создадим XML-файл, содержащий фрагмент ресторанного меню (листинг 4.19).

Листинг 4.19. Файл menu.xml

```
<?xml version="1.0"?>
<breakfast_menu>
<food>
<name>Oladya Prostaya</name>
```

```
<price>105.90</price>
<description>Tri oladyi s maslom i syrnymi struzhkami</description>
<calories>650</calories>
</food>
<food>
<name>Blinchiki s ruzhikami</name>
<price>176.50</price>
<description>Dva blinchika, nachinennyh ryzhikami, so smetanoj ruchnogo
izgotovleniya</description>
<calories>900</calories>
</food>
<food>
<name>Buterbrod s semgoj</name>
<price>120.70</price>
<description>Buterbrod s semgoy</description>
<calories>900</calories>
</food>
<food>
<name>Chai</name>
<price>34.70</price>
<description>Chai kitajskij dieticheskij</description>
<calories>600</calories>
</food>
<food>
<name>Kofe</name>
<price>70.95</price>
<description>Espresso capuccino s "medvezhjej ikroj"</description>
<calories>950</calories>
</food>
</breakfast_menu>
```

Простая загрузка в браузер приведет к отображению древовидной структуры XML-файла (рис. 4.42).

Чтобы "оживить" отображение, сделать его более интересным, используем файл XSL, содержащий описание расширяемых стилей (листинг 4.20).

Листинг 4.20. Файл menu.xsl

```
<?xml version="1.0"?>
<html xsl:version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/TR/xhtml1/strict">
<body style="font-family:Arial,helvetica,sans-serif;font-size:12pt;
background-color:#BCBCBC">
```

```
<xsl:for-each select="breakfast_menu/food">
<div style="background-color:teal;color:white;padding:6px">
<span style="font-weight:bold;color:white">
<xsl:value-of select="name"/></span>
- <xsl:value-of select="price"/>
</div>
<div style="margin-left:40px;margin-bottom:1em;font-size:10pt">
<xsl:value-of select="description"/>
<span style="font-style:italic">
(<xsl:value-of select="calories"/> Kalorij na odnu portsiyu)
</span>
</div>
</xsl:for-each>
</body>
</html>
```

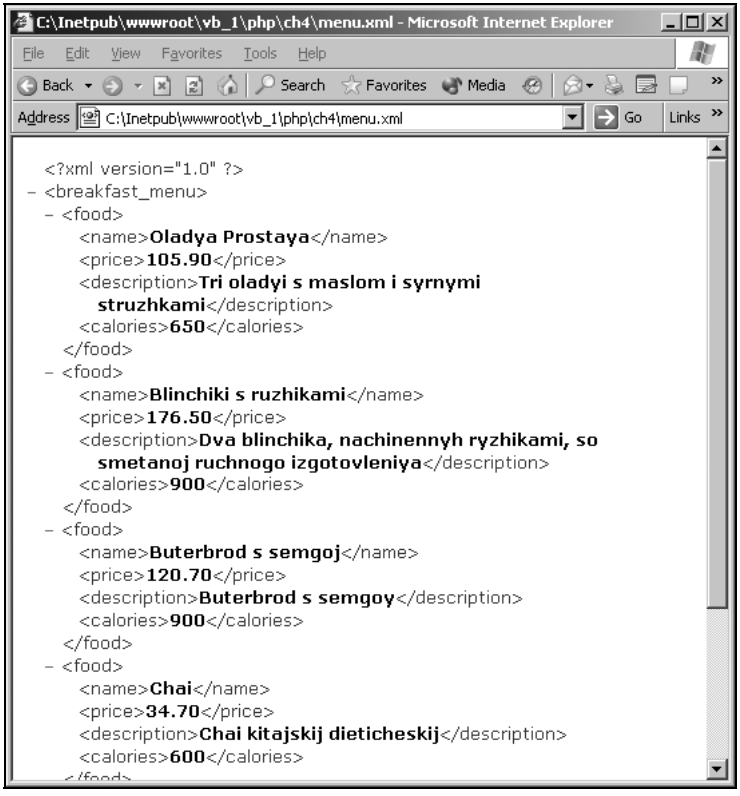

Рис. 4.42. Файл menu.xml

Чтобы XML-файл был отображен с использованием описанных стилей XSL, необходимо вставить в исходный XML-файл ссылку на файл стилей:

```
<?xml-stylesheet type="text/xsl" href="menu.xsl" ?>
```

Сохраним файл под именем menuxsl.xml (листинг 4.21).

Листинг 4.21. Файл menuxsl.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="menu.xsl" ?>
<breakfast_menu>
<food>
<name>Oladya Prostaya</name>
<price>105.90</price>
<description>Tri oladyi s maslom i syrnymi struzhkami</description>
<calories>650</calories>
</food>
<food>
<name>Blinchiki s ruzhikami</name>
<price>176.50</price>
<description>Dva blinchika, nachinennyh ryzhikami, so smetanoj ruchnogo
izgotovleniya</description>
<calories>900</calories>
</food>
<food>
<name>Buterbrod s semgoj</name>
<price>120.70</price>
<description>Buterbrod s semgoy</description>
<calories>900</calories>
</food>
<food>
<name>Chai</name>
<price>34.70</price>
<description>Chai kitajskij dieticheskij</description>
<calories>600</calories>
</food>
<food>
<name>Kofe</name>
<price>70.95</price>
<description>Espresso capuccino s "medvezhjej ikroj"</description>
<calories>950</calories>
</food>
</breakfast_menu>
```

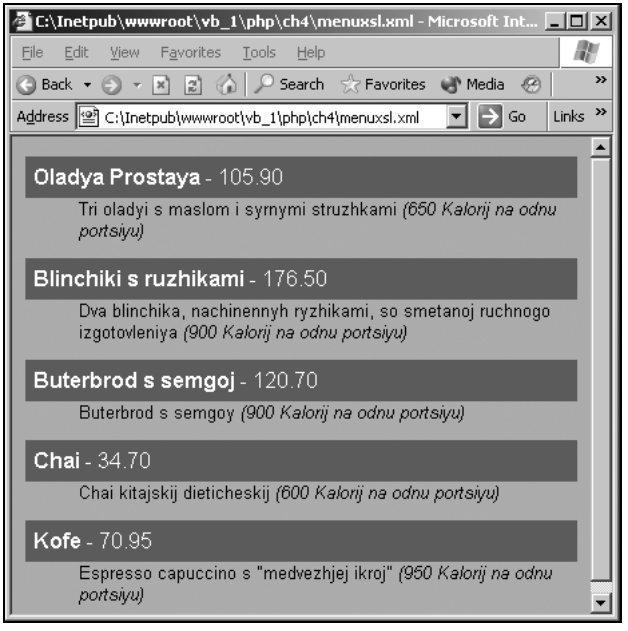

Рис. 4.43. Применение стилей XSL

Загрузка этого файла в браузер приведет к использованию стилей XSL в применении к файлу menuxsl.xml (рис. 4.43).

В предыдущем примере для отображений XML-данных с использованием XSL-стилей мы воспользовались возможностями, встроенными в браузер Internet Explorer. Это легкий, но не всегда удобный способ. Разумнее было бы преобразовывать XML-данные и создавать клиентский документ на стороне сервера. Посмотрим, как XSL-преобразования могут быть произведены средствами языка PHP.

Немного изменим исходные XML-данные, уберем из файла menuxsl.xml строку:

```
<?xml-stylesheet type="text/xsl" href="menu.xsl" ?>
```

Сохраним новый файл под именем menu.xml. Создадим файл menu.php, который будет обрабатывать XML-данные с использованием стилей XSL (листинг 4.22).

Листинг 4.22. Файл menu.php

```
<?php
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('menu.xml');
```

```
$xsl = new DomDocument;
$xsl->load('menu.xsl');

// Создаем XSL-процессор
$proc = new xsltprocessor;
// Указываем файл XSL-преобразований
$proc->importStyleSheet($xsl);
// Производим преобразования и отображаем результат
echo $proc->transformToXML($xml);
?>
```

На рис. 4.44 показан результат преобразования XML-документа в HTML-текст.

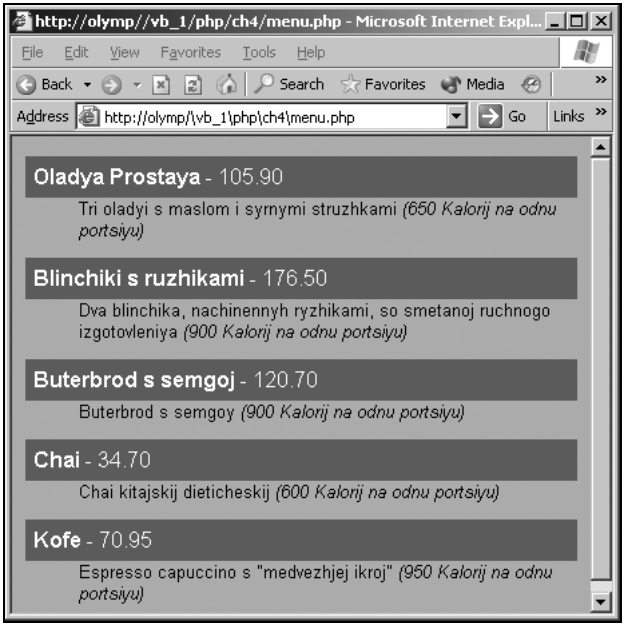

Рис. 4.44. Обработка XML-данных на сервере

В соответствии с заданным файлом XSL, PHP-скрипт направляет браузеру HTML-код (листинг 4.23).

Листинг 4.23. Результат работы скрипта menu.php

```
<body>
<div style="PADDING-RIGHT: 6px; PADDING-LEFT: 6px; PADDING-BOTTOM: 6px;
COLOR: white; PADDING-TOP: 6px; BACKGROUND-COLOR: teal">
```

```
  <span style="FONT-WEIGHT: bold; COLOR: white">Oladya Prostaya</span> -
105.90
</div>
<div style="FONT-SIZE: 10pt; MARGIN-BOTTOM: 1em; MARGIN-LEFT: 40px">
  Tri oladyi s maslom i syrnymi struzhkami<span style="FONT-STYLE:
italic"> (650
  Kalorij na odnu portsiyu)</span>
</div>
<div style="PADDING-RIGHT: 6px; PADDING-LEFT: 6px; PADDING-BOTTOM: 6px;
COLOR: white; PADDING-TOP: 6px; BACKGROUND-COLOR: teal">
  <span style="FONT-WEIGHT: bold; COLOR: white">Blinchiki s
ruzhikami</span> -
  176.50
</div>
<div style="FONT-SIZE: 10pt; MARGIN-BOTTOM: 1em; MARGIN-LEFT: 40px">
  Dva blinchika, nachinennyh ryzhikami, so smetanoj ruchnogo
izgotovleniya<span style="FONT-STYLE: italic">
  (900 Kalorij na odnu portsiyu)</span>
</div>
<div style="PADDING-RIGHT: 6px; PADDING-LEFT: 6px; PADDING-BOTTOM: 6px;
COLOR: white; PADDING-TOP: 6px; BACKGROUND-COLOR: teal">
  <span style="FONT-WEIGHT: bold; COLOR: white">Buterbrod s semgoj</span> -
  120.70
</div>
<div style="FONT-SIZE: 10pt; MARGIN-BOTTOM: 1em; MARGIN-LEFT: 40px">
  Buterbrod s semgoy<span style="FONT-STYLE: italic"> (900 Kalorij na
odnu
  portsiyu)</span>
</div>
<div style="PADDING-RIGHT: 6px; PADDING-LEFT: 6px; PADDING-BOTTOM: 6px;
COLOR: white; PADDING-TOP: 6px; BACKGROUND-COLOR: teal">
  <span style="FONT-WEIGHT: bold; COLOR: white">Chai</span> - 34.70
</div>
<div style="FONT-SIZE: 10pt; MARGIN-BOTTOM: 1em; MARGIN-LEFT: 40px">
  Chai kitajskij dieticheskij<span style="FONT-STYLE: italic">
(600 Kalorij na
  odnu portsiyu)</span>
</div>
<div style="PADDING-RIGHT: 6px; PADDING-LEFT: 6px; PADDING-BOTTOM: 6px;
COLOR: white; PADDING-TOP: 6px; BACKGROUND-COLOR: teal">
  <span style="FONT-WEIGHT: bold; COLOR: white">Kofe</span> - 70.95
</div>
<div style="FONT-SIZE: 10pt; MARGIN-BOTTOM: 1em; MARGIN-LEFT: 40px">
  Espresso capuccino s &amp;quot;medvezhjej ikroj&amp;quot;<span style=
"FONT-STYLE: italic">
  (950 Kalorij na odnu portsiyu)</span>
```

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

Стили XSL

Язык XSL (Extensible Stylesheet Language) — это расширяемый язык листов стилей, применяемых совместно с XML-документами. XSL состоит из трех языков, которые являются своеобразными составными "подъязыками" языка XSL. Это XSLT, XPath и XSL Formatting Objects (XSL FO).

Язык XSL в чем-то похож на каскадные листы стилей CSS, используемые совместно с HTML. Но в CSS набор тэгов строго определен заранее, в то время как набор тэгов, применяемых в XSL, не фиксирован. Так, в HTML каждый тэг соответствует заранее оговоренному элементу, о котором известно, как он будет представлен. Например, `<p>` используется для выделения абзаца, `<h1>` соответствует заголовку первого уровня и т. д. При этом браузер заранее знает, как следует отображать такие элементы. Существует возможность задания параметров отображения элементов в окне браузера. Для этого можно использовать листы стилей CSS, которые определяют способы отображения элементов, причем описание, используемое в стилях CSS, понятно для браузера.

Аналогично XSL — это стили для языка XML. Но в случае с XML браузер не имеет никакого представления о том, как следует отобразить XML-документ. Более того, не все XML-документы предназначены для того, чтобы быть отображенными в браузере.

XML обладает следующими возможностями:

- XML-данные легко преобразовывать из одного вида в другой. Для этого используется язык XSLT (XSL Transforms);
- XML-данные удобны для доступа к частям, составляющим XML-документ. Для этого используется язык XPath;
- форматировать XML-данные можно при помощи языка XSL Formatting Objects (XSL FO).

Язык XSL используется для описания преобразований исходного XML-документа в другой документ, который может быть документом XML или XHTML. При этом происходит "фильтрация" данных, сортировка исходных XML-данных, преобразование данных. Язык XSL представляет собой стандарт, разработанный консорциумом W3C. Мы остановимся на языках XSLT и XPath.

Язык XSLT

Язык XSLT — наиболее важная часть стандарта языка XSL. С помощью этого языка описывают преобразования, используемые для отображения

исходного документа XML в конечный документ, в частности в документ XML, или в документ другого типа. В клиентском Web-программировании с использованием браузеров в качестве клиентских агентов наиболее распространенными типами конечных документов являются документы XML, HTML, XHTML. В простейшем случае язык XSLT применяется для задания преобразования XML-документа, когда каждый элемент XML преобразуется в соответствующий ему элемент XHTML. При этом может происходить вставка новых элементов, удаление тех или иных элементов, сортировка элементов и другие действия. Широко распространенный подход состоит в том, что при помощи XSLT происходит преобразование древовидной структуры исходного XML-документа в древовидную структуру конечного XML-документа.

В процессе преобразования используется язык XPath — при его помощи XSLT получает информацию о частях исходного документа, которые должны соответствовать одному или нескольким заранее определенным шаблонам. Когда соответствие части документа шаблону обнаружено, XSLT приступает к преобразованию этой части во фрагмент результирующего документа. Если часть документа не будет соответствовать ни одному шаблону, то такая часть будет оставлена без изменений.

Корневым элементом XSL является элемент `xsl:stylesheet` или элемент `xsl:transform`. Оба эти элемента равнозначны и являются полными синонимами. Декларация XSL-файла имеет такой вид:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Эквивалентная запись:

```
<xsl:transform version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Запись `xmlns:xsl="http://www.w3.org/1999/XSL/Transform"` определяет имя для официально зарегистрированного пространства имен. Вместе с именем пространства имен (если оно указано) необходимо задать атрибут версии языка:

```
version="1.0".
```

Старый вариант декларации XSL-файла имел такой вид:

```
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

Такое описание использовалось для работы с браузером Internet Explorer 5.

Рассмотрим пример. Преобразуем файл cd1.xml (каталог CD) в документ XHTML (листинг 4.24).

Листинг 4.24. Файл cd1.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CATALOG>
<CD>
<TITLE>Nazvanie 1</TITLE>
<ARTIST>Artist 1</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>Columbia</COMPANY>
<PRICE>10.90</PRICE>
<YEAR>1985</YEAR>
</CD>
. . .
. . .
. . .
<CD>
<TITLE>Nazvanie 26</TITLE>
<ARTIST>Artist 26</ARTIST>
<COUNTRY>USA</COUNTRY>
<COMPANY>EMI</COMPANY>
<PRICE>8.20</PRICE>
<YEAR>1987</YEAR>
</CD>
</CATALOG>
```

Сам по себе этот файл не представляет ничего особенно интересного. Создадим файл, содержащий описание стилей XSL, дадим ему имя cd1.xsl (листинг 4.25).

Листинг 4.25. Файл cd1.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<body>
<h2> Моя коллекция CD </h2>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
```

```
<xsl:template match="cd">
<p>
<xsl:apply-templates select="title"/>
<xsl:apply-templates select="artist"/>
</p>
</xsl:template>

<xsl:template match="title">
Название: <span style="color:#845321">
<sl:value-of select="."/>
<br />
</span> </xsl:template>

<xsl:template match="artist">
Артист:
<xsl:value-of select="."/></span>
<br />
</xsl:template>

</xsl:stylesheet>
```

Чтобы увидеть результат, применим XSL-преобразования с использованием PHP-файла shablon1.php (листинг 4.26).

Листинг 4.26. Файл shablon1.php

```
<?php
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd1.xsl');

// Создаем XSL-процессор
$proc = new xsltprocessor;
// Указываем файл XSL-преобразований
$proc->importStyleSheet($xsl);
// Производим преобразования и отображаем результат
echo $proc->transformToXML($xml);
?>
```

Результат преобразования показан на рис. 4.45.

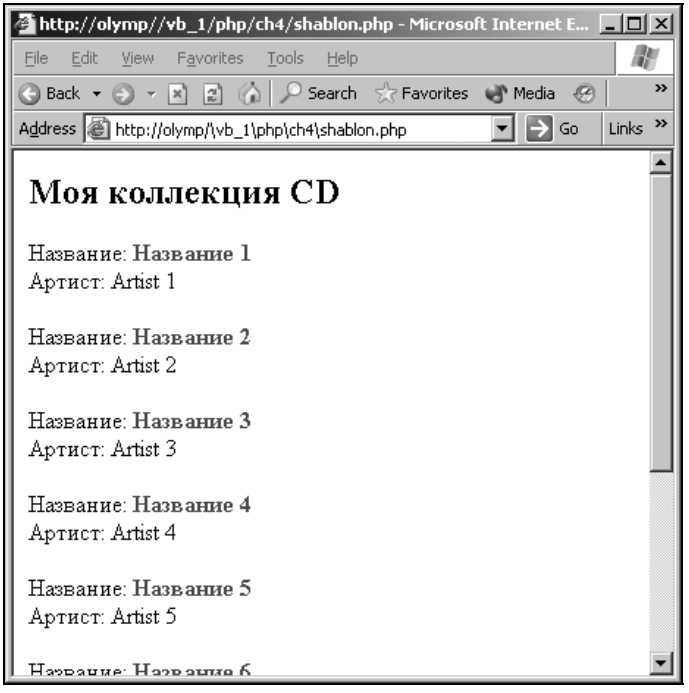

Рис. 4.45. Файл shablon.php

Мы будем использовать шаблон shablon.php и в дальнейшем, в нем достаточно будет поменять имена файлов cd1.xml и cd1.xsl на новые.

Далее мы рассмотрим более подробно содержимое файла cd1.xsl.

Элемент *<xsl:template>*

Стили XSL состоят из наборов правил, которые объединяются в шаблоны `template`. Каждый элемент `<xsl:template>` состоит из правил, которые применяются к тому или иному элементу, если в исходном XML-файле находится элемент с именем, совпадающим с именем шаблона. Атрибут `match` элемента `<xsl:template>` используется для поиска элемента исходного XML-файла, соответствующего значению этого атрибута. Этот элемент также используется, если правила относятся ко всему документу в целом, в этом случае в качестве значения `match` указывается "/". Для вывода текста заголовка в предыдущем примере применяется элемент с атрибутом `match="/"`. Например:

```
<xsl:template match="/">
<html>
<body>
```

```
<h2> Моя коллекция CD </h2>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
```

В качестве примера мы рассмотрим некоторые стили (листинг 4.27).

Листинг 4.27. Файл cd3.xsl

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>Моя коллекция CD</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Название</th>
<th>Artist</th>
</tr>
<tr>
<td>.</td>
<td>.</td>
</tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Если в сочетании с этим файлом стилей отобразить исходный файл (cd1.xml), то получим только заголовок таблицы (рис. 4.46).

В файл shablon3.php вносим следующие изменения (листинг 4.28).

Листинг 4.28. Фрагмент файла shablon3.php

```
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd3.xsl');
```

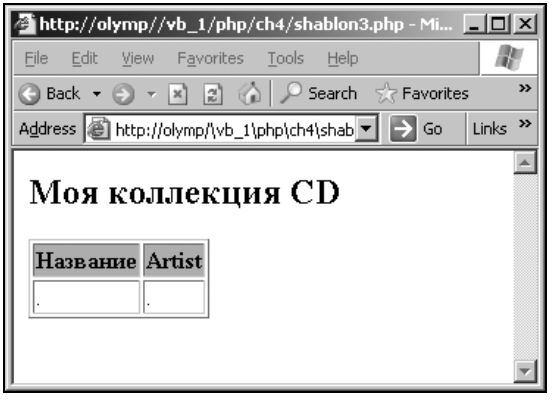

Рис. 4.46. Еще один вариант отображения XML-файла с применением стилей

Элемент *<xsl:value-of>*

Элемент `<xsl:value-of>` используется для получения данных, содержащихся в соответствующем элементе XML-файла. Пример приведен в листинге 4.29.

Листинг 4.29. Файл cd4.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<tr>
<td><xsl:value-of select="CATALOG/cd/title"/></td>
<td><xsl:value-of select="CATALOG/cd/artist"/></td>
</tr>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

В браузере мы увидим первую строку таблицы каталога (рис. 4.47).

Рис. 4.47. Первый элемент списка CD

Здесь используется файл shablon4.php (листинг 4.30).

Листинг 4.30. Фрагмент файла shablon4.php

```
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd4.xsl');
```

Элемент <xsl:for-each>

Элемент `<xsl:for-each>` используется для организации циклов. Этот элемент, например, может служить для того, чтобы показать все элементы исходного XML-файла, соответствующие заданному шаблону. Пример XSL-стиля приведен в листинге 4.31.

Листинг 4.31. Файл cd5.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
```

```
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Соответствующий файл shablon5.php показан в листинге 4.32.

Листинг 4.32. Фрагмент файла shablon5.php

```
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd5.xsl');
```

В окне браузера мы увидим весь список дисков (рис. 4.48).

Элемент `<xsl:for-each>` можно использовать для фильтрования выводимых данных. Можно написать, например, так:

```
<xsl:for-each select="catalog/cd[artist='Artist 5']">
```

Здесь допускается использование следующих операций:

- = (равенство)
- != (неравенство)
- < меньше чем
- > больше чем

Вот еще пример листа стилей, сохраненного в файле cd6.xsl (листинг 4.33).

Листинг 4.33. Файл cd6.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">
<html>
<body>
<h2>My CD Collection</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/cd[artist='Artist 7']">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

В браузере будет отображено только то имя артиста, которое было указано в условии (рис. 4.49).

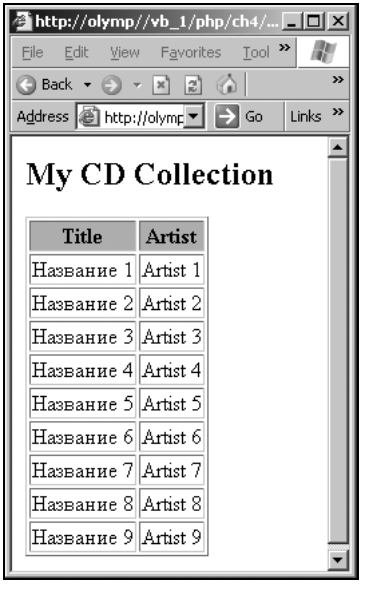

Рис. 4.48. Таблица выведена с применением элемента `<xsl:for-each>`

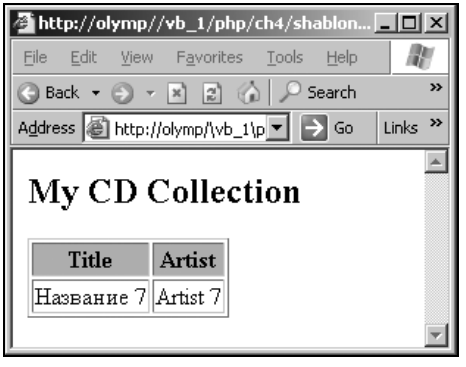

Рис. 4.49. Фильтрация данных

Вносим изменения в шаблон shablon6.php (листинг 4.34).

Листинг 4.34. Изменения в файле shablon6.php

```
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd6.xsl');
```

Элемент *<xsl:sort>*

При помощи элемента `<xsl:sort>` осуществляется сортировка результата. Для осуществления сортировки можно вставить внутрь элемента `<xsl:fro-each>` в качестве дочернего элемента элемент `<xsl:sort>`. Пример приведен в листинге 4.35.

Листинг 4.35. Файл cd7.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>Моя коллекция CD </h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Название</th>
<th>Artist</th>
<th>Год</th>
</tr>
<xsl:for-each select="CATALOG/cd">
<xsl:sort select="year"/>
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
<td><xsl:value-of select="year"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
```

```
</xsl:template>
</xsl:stylesheet>
```

В листинге 4.36 показаны изменения, которые необходимо внести в файл shablon7.php.

Листинг 4.36. Изменения в файле shablon7.php

```
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd7.xsl');
```

Выводимые элементы будут упорядочены по заданному параметру (в нашем случае это год выпуска) (рис. 4.50).

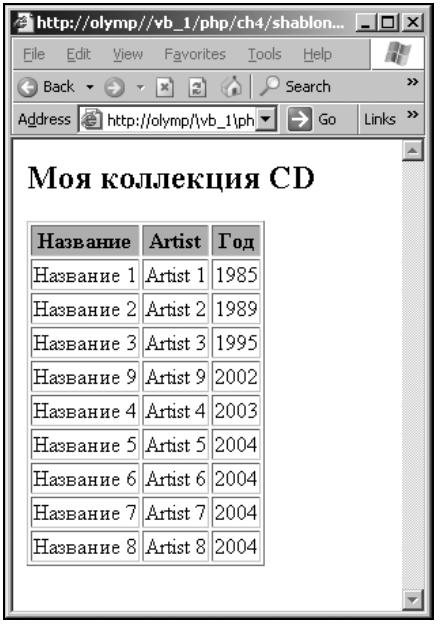

Рис. 4.50. Упорядочение данных

Элемент *<xsl:if>*

Элемент `<xsl:if>` содержит шаблон, который будет применен только в том случае, если заданное условие будет выполнено.

Конструкция с `if` используется таким образом:

```
<xsl:if test="price>'10'"> вывод ... </xsl:if>
```

В качестве примера рассмотрим файл cd8.xsl (листинг 4.37).

Листинг 4.37. Файл cd8.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>Моя коллеция CD<
</h2> <table border="1">
<tr bgcolor="#9acd32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/cd">
<xsl:if test="price>10">
<tr>
<td><xsl:value-of select="title"/></td>
<td><xsl:value-of select="artist"/></td>
</tr>
</xsl:if>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

В результате получим таблицу, содержащую список CD, цена которых выше 10 (рис. 4.51).

Изменения в файле shablon8.php показаны в листинге 4.38.

Листинг 4.38. Измененный файл shablon8.php

```
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd8.xsl');
```

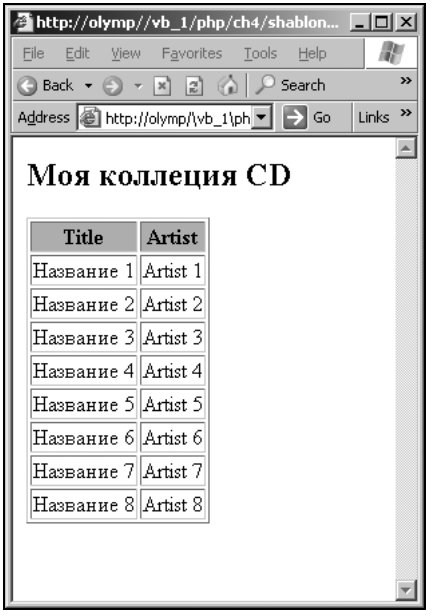

Рис. 4.51. Вывод списка CD с проверкой условия

Элемент <xsl:choose>

Элемент `<xsl:choose>` используется в сочетании с элементами `<xsl:when>` и `<xsl:otherwise>`. При помощи этих элементов организуется условный выбор. Например:

```
<xsl:choose>
<xsl:when test="price>'10'">
... код ...
</xsl:when>
<xsl:otherwise>
... код ...
</xsl:otherwise>
</xsl:choose>
```

Для нашего примера с компакт-дисками можно написать такой файл стилей XSL (листинг 4.39).

Листинг 4.39. Файл cd9.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">
<html>
<body>
<h2>Моя коллекция CD </h2>
<table border="1">
<tr bgcolor="#9aff32">
<th>Title</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<xsl:choose>
<xsl:when test="price>'15'">
<td bgcolor="#ff00ff">
<xsl:value-of select="artist"/></td>
</xsl:when>
<xsl:otherwise>
<td><xsl:value-of select="artist"/></td>
</xsl:otherwise>
</xsl:choose>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Сейчас в окне браузера после загрузки файла cd9.xsl будут отображены только такие диски, цена которых выше 15 (рис. 4.52).

Изменения, которые необходимо внести в файл shablon9.php, показаны в листинге 4.40.

Листинг 4.40. Измененный файл shablon9.php

```
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd9.xsl');
```

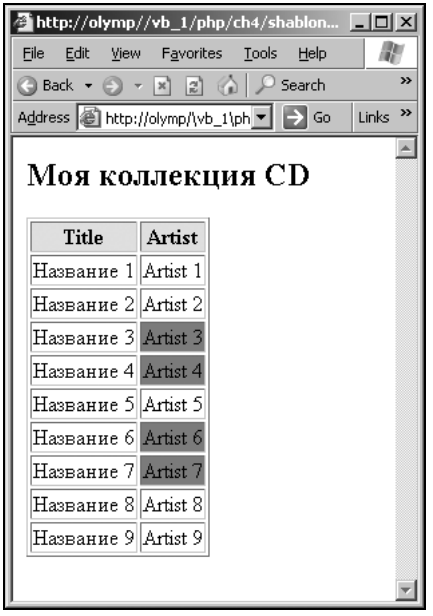

Рис. 4.52. Выделены диски, цена которых выше 15

В листинге 4.41 приведен еще один пример. Здесь мы используем несколько элементов `<xsl:when>` в одном файле.

Листинг 4.41. Файл cd10.xsl

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>Моя коллекция CD</h2>
<table border="1">
<tr bgcolor="#9a7132">
<th>Название</th>
<th>Artist</th>
</tr>
<xsl:for-each select="CATALOG/cd">
<tr>
<td><xsl:value-of select="title"/></td>
<xsl:choose>
<xsl:when test="price>'15'">
```

```
<td bgcolor="#ff00ff">
<xsl:value-of select="artist"/></td>
</xsl:when>
<xsl:when test="price>'9' and price&gt;='15'">
<td bgcolor="#cccccc">
<xsl:value-of select="artist"/></td>
</xsl:when>
<xsl:otherwise>
<td><xsl:value-of select="artist"/></td>
</xsl:otherwise>
</xsl:choose>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

В этом примере мы одним цветом выделяем диски, стоимость которых выше 15. Другим цветом — диски, стоимость которых от 9 до 15 (рис. 4.53). Вместо значка < мы воспользовались "сущностью" `<`.

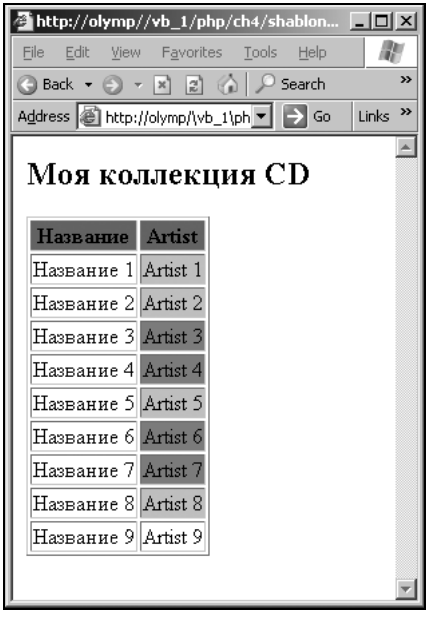

Рис. 4.53. Два условия — два цвета выделения элементов

Изменения, которые необходимо внести в файл shablon10.php, показаны в листинге 4.42.

Листинг 4.42. Изменения в файле shablon10.php

```
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd10.xsl');
```

Элемент <xsl:apply-templates>

Элемент `<xsl:apply-templates>` используется для того, чтобы применить правила шаблона к текущему элементу или к дочернему элементу текущего элемента. В качестве примера приводим файл cd11.xsl (листинг 4.43).

Листинг 4.43. Файл cd11.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>Моя коллекция CD</h2>
<xsl:apply-templates/>
</body>
</html>
</xsl:template>
<xsl:template match="cd">
<p>
<xsl:apply-templates select="title"/>
<xsl:apply-templates select="artist"/>
</p>
</xsl:template>
<xsl:template match="title">
Название: <span style="color:#ff1111">
<xsl:value-of select="."/></span>
<br />
</xsl:template>
<xsl:template match="artist">
Artist: <span style="color:#11ff11">
```

```
<xsl:value-of select="."/></span>
<br />
</xsl:template>
</xsl:stylesheet>
```

В результате применения такого стиля в окне браузера получаем список всех дисков, которые выводятся в соответствии с заданным в стиле описанием (рис. 4.54). Файл shablon11.php показан в листинге 4.44.

Листинг 4.44. Изменения в файле shablon11.php

```
// Загрузка XML-данных
$xml = new DomDocument;
$xml->load('cd1.xml');

$xsl = new DomDocument;
$xsl->load('cd11.xsl');
```

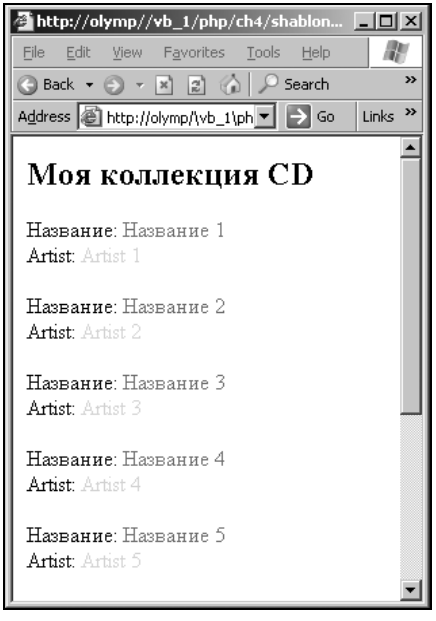

Рис. 4.54. Применение шаблонов

Краткая справка по элементам XSLT

В табл. 4.2 приведены элементы XSLT и их краткое описание. Там также указаны версии браузеров, которые поддерживают работу с тем или иным элементом.

Таблица 4.2. Элементы XSLT

Элемент	Описание
xsl:apply-imports	Применяет правило шаблона из импортируемого листа стилей
xsl:apply-templates	Применяет правило шаблона к текущему элементу или к дочернему элементу
xsl:attribute	Вставляет атрибут
xsl:attribute-set	Определяет именованный набор атрибутов
xsl:call-template	Вызывает шаблон
xsl:choose	Используется в сочетании с <xsl:when> и <xsl:otherwise> для выполнения проверок условий
xsl:comment	Создает элемент с комментариями
xsl:copy	Создает копию текущего элемента (без дочерних элементов и атрибутов)
xsl:copy-of	Создает копию текущего элемента (с дочерними элементами и атрибутами)
xsl:decimal-format	Задает символы, используемые при преобразовании чисел в строку, в функции format-number()
xsl:element	Создает элемент результирующего документа
xsl:fallback	Служит для записи альтернативного кода, если процессор не поддерживает тот или иной элемент XSL
xsl:for-each	Просматривает все элементы в указанном наборе
xsl:if	Содержит шаблон, который будет применен только тогда, когда выполняется заданное условие
xsl:import	Импортирует один лист стилей в другой (импортированный лист имеет меньший приоритет)
xsl:include	Вставляет лист стилей. Вставленный лист стилей имеет тот же приоритет, что и лист, в который он вставляется
xsl:key	Объявляет поименованный ключ, который может быть использован с применением функции key()
xsl:message	Пишет сообщение (используется для вывода сообщений об ошибках)
xsl:namespace-alias	Заменяет пространство имен листа стилей на новое пространство имен результирующего документа
xsl:number	Определяет положение текущего элемента и форматирует полученное число

Таблица 4.2 *(окончание)*

Элемент	Описание
xsl:otherwise	Действие, выполняемое по умолчанию в элементе <xsl:choose>
xsl:output	Задает формат результирующего документа
xsl:param	Объявляет локальные или глобальные параметры
xsl:preserve-space	Задает элементы, для которых важно соблюдение пробелов (пробелы будут сохранены)
xsl:processing-instruction	Выводит инструкции обработки
xsl:sort	Сортирует элементы
xsl:strip-space	Задает элементы, в которых следует удалить пробелы
xsl:stylesheet	Корневой элемент листа стилей
xsl:template	Правила стилей, применяемые к подходящему элементу
xsl:text	Выводит текст в результирующий документ
xsl:transform	Корневой элемент листа стилей
xsl:value-of	Получает значение указанного элемента
xsl:variable	Задает глобальные и локальные переменные
xsl:when	Задает действие, применяемое в элементе <xsl:choose>
xsl:with-param	Задает значение параметра, передаваемого шаблону

Язык XPath

Язык XPath представляет собой набор правил, которые определяют синтаксис описаний частей XML-документов. Этот язык необходим, чтобы организовать правильный доступ к элементам XML-документов из файла стилей XSLT. Синтаксису XPath должны удовлетворять выражения, с помощью которых осуществляется подобный доступ. Эти выражения указываются в качестве значений некоторых атрибутов в файлах XSLT. XPath создан на основе XML и является стандартом консорциума W3C и частью стандарта XSLT.

В XPath используются выражения, похожие на выражения, записываемые при указании пути к файлу, например: `www/xpath/default.asp`. Начнем с примеров: в листинге 4.45 приведен файл с использованием языка XPath.

Листинг 4.45. Файл xpath.xml

```
<?xml version="1.0"?>
<catalog>
<cd country="RU">
<title>За пивом</title>
<artist> Академия</artist>
<price>10.90</price>
</cd>
<cd country="RU">
<title>Моя зайка</title>
<artist>Киркоров</artist>
<price>9.90</price>
</cd>
<cd country="UK">
<title>Greatest Hits</title>
<artist>Misc</artist>
<price>9.90</price>
</cd>
</catalog>
```

Если использовать выражение `XPath /catalog`, то в приведенном выше файле xpath.xml будет выбран корневой элемент. Для выбора всех элементов `<cd>` используется выражение XPath в виде `/catalog/cd`. Всем элементам цены соответствует путь `catalog/cd/price`. Если путь начинается со знака `/`, то такой путь является абсолютным. В выражениях XPath также могут быть использованы условия: `/catalog/cd[price>10.80]`.

Синтаксис языка XPath

Мы рассмотрим примеры, основанные на работе с файлом xpath.xml, приведенном в листинге 4.44.

Мы видели, как используются выражения типа `/catalog/cd/price`. Если использовать двойную черту `//`, то будут выбраны все элементы документа, соответствующие заданным критериям, например `//cd`. Если название элемента неизвестно, то можно воспользоваться символом `*`. Следующий пример выбирает все дочерние элементы, содержащиеся в элементах `cd`, вложенных в элемент `catalog`:

```
/catalog/cd/*
```

Следующий пример выбирает все элементы `price`, которые являются внуками по отношению к элементу `catalog`:

```
/catalog/*/price
```

Следующий пример выбирает все элементы price, которые имеют по два элемента, старше их (родительский и прародительский):

```
/*/*/price
```

В примере ниже выбираются все элементы документа:

```
//*
```

Выбор ветвей

Чтобы выбрать первый элемент cd, можно указать индекс элемента в пути, например:

```
/catalog/cd[1]
```

Для выбора последнего элемента используется такая конструкция:

```
/catalog/cd[last()]
```

Выбор элементов cd, которые в своем составе имеют дочерний элемент price, осуществляется так:

```
/catalog/cd[price]
```

Выбор всех элементов (дисков), цена которых равна 10.90:

```
/catalog/cd[price=10.90]
```

Выбор всех элементов price, значение которых равно 10.90:

```
/catalog/cd[price=10.90]/price
```

Выбор нескольких путей

Для указания нескольких альтернативных путей используется разделитель |. При помощи следующей записи выбираются все элементы названий и все элементы артистов:

```
/catalog/cd/title | /catalog/cd/artist
```

Следующий фрагмент используется для выбора всех элементов названий, артистов и цен:

```
//title | //artist | //price
```

Следующий фрагмент выбирает названия в элементе cd и все элементы артистов:

```
/catalog/cd/title | //artist
```

Выбор атрибутов

Все атрибуты в XPath начинаются с символа @. Для выбора всех атрибутов country запишем такую строку:

```
//@country
```

Для выбора всех элементов cd, которые обладают атрибутом country, пишем так:

```
//cd[@country]
```

Выбор всех элементов cd, которые имеют любой атрибут:

```
//cd[@*]
```

Выбор элементов cd, атрибут country которых имеет значение "UK":

```
//cd[@country='UK']
```

Путь в XPath

Путь в XPath может быть абсолютным или относительным. Абсолютный путь начинается с символа /. Относительный путь не начинается с этого символа. Например:

```
Absolyutny put:
/step/step/...
Otnositelny put:
step/step/...
```

Путь "вычисляется" по порядку слева направо. На каждом шаге очередной переход вычисляется по отношению к текущему положению в наборе узлов в дереве пути. Шаги при вычислении пути состоят из следующих элементов:

- ось (определяет взаимоотношение между выбранным узлом и текущим узлом);
- проверка узла (задает тип узла и расширенное название узла);
- несколько (или 0) предикатов, уточняющих набор узлов.

Синтаксис при задании шага таков:

```
axisname::nodetest[predicate]
```

Например:

```
child::price[price=9.90]
```

Ось задает отношение набора узлов к текущему узлу. Проверка узла используется для задания узла на указанной оси. Проверка узла может быть осуществлена либо по имени, либо по типу (табл. 4.3, 4.4).

Таблица 4.3. *Типы осей в XPath*

Название оси (axisname)	Описание
Ancestor	Содержит все элементы потомков (дочерние, внуки и т. д.)
ancestor-or-self	Содержит текущий узел и все потомки (дочерние, внуки и т. д.)

Таблица 4.3 *(окончание)*

Название оси (axisname)	Описание
`Attribute`	Содержит все атрибуты текущего узла
`Child`	Содержит все дочерние элементы текущего узла
`descendant`	Содержит младшие узлы (дочерние, внуки и т. д.). Никогда не содержит атрибутов и пространств имен
`descendant-or-self`	Содержит текущий узел и все младшие узлы (дочерние, внуки и т. д.)
`Following`	Содержит все, что расположено после закрывающего ярлыка текущего узла
`following-sibling`	Содержит все смежные узлы, расположенные после текущего узла (но не атрибуты или пространства имен)
`namespace`	Содержит все узлы пространств имен в данном узле
`Parent`	Содержит родительский узел
`Preceding`	Содержит все, что есть в документе и расположено перед текущим узлом
`Preceding-sibling`	Содержит все смежные элементы, расположенные перед текущим узлом (но не атрибуты и не пространства имен)
`Self`	Содержит текущий узел

Таблица 4.4. *Примеры с заданием осей*

Пример	Описание
`child::cd`	Выбор элементов `cd`, являющихся дочерними по отношению к текущему узлу (если дочерних элементов `cd` нет, то ничего не будет выбрано)
`attribute::src`	Выбирает в текущем узле атрибут `src`
`child::*`	Выбирает все дочерние элементы для данного узла
`attribute::*`	Выбирает все атрибуты в данном узле
`child::text()`	Выбирает текстовые дочерние элементы в узле
`child::node()`	Выбирает все дочерние элементы
`descendant::cd`	Выбирает все элементы `cd`, являющиеся потомками по отношению к текущему узлу
`ancestor::cd`	Выбирает все родительские элементы `cd`

Таблица 4.4 *(окончание)*

Пример	Описание
`ancestor-or-self::cd`	Выбирает все родительские элементы `cd` и сам элемент (если элемент является `cd`)
`child::*/child::price`	Выбирает все элементы — внуки `price`
`/`	Выбирает корневой элемент документа

Предикаты используются для фильтрования наборов узлов, они преобразуют исходные наборы узлов в новые наборы узлов. Предикаты помещаются в квадратные скобки `[ ]` (табл. 4.5).

Таблица 4.5. *Примеры с предикатами*

Пример	Описание
`child::price[price=9.90]`	Выбирает все дочерние элементы `price`, равные 9.90
`child::cd[position()=1]`	Выбирает первый элемент `cd` в текущем узле
`child::cd[position()=last()]`	Выбирает последний элемент `cd` в текущем узле
`child::cd[position()=last()-1]`	Выбирает предпоследний элемент `cd` в текущем узле
`child::cd[position()<6]`	Выбирает первые 5 элементов `cd` в текущем узле
`/descendant::cd[position()=7]`	Выбирает седьмой элемент `cd` в документе
`child::cd[attribute::type="classic"]`	Выбирает все дочерние элементы `cd` текущего узла, в которых есть атрибут `type` со значением `classic`

При задании путей можно использовать укороченные синтаксические конструкции (табл. 4.6). Примеры приведены в табл. 4.7.

Таблица 4.6. *Укороченные синтаксические конструкции в XPath*

Сокращение	Значение	Пример
None	`child::`	`Cd` — краткое написание для `child::cd`
`@`	`attribute::`	`cd[@type="classic"]` — краткое написание для `child::cd[attribute::type="classic"]`

Таблица 4.6 *(окончание)*

Сокращение	Значение	Пример
.	self::node()	.//cd — краткое написание для self::node()/descendant-or-self::node()/child::cd
..	parent::node()	../cd — краткое написание для parent::node()/child::cd
//	/descendant-or-self::node()/	//cd — краткое написание для /descendant-or-self::node()/child::cd

Таблица 4.7. *Примеры с укороченными конструкциями*

Пример	Описание
Cd	Выбирает все элементы cd в данном узле
*	Выбирает все (дочерние) элементы в данном узле
text()	Выбирает все текстовые элементы в узле
@src	Выбирает атрибут src
@*	Выбирает все атрибуты в данном узле
cd[1]	Выбирает первый элемент cd в узле
cd[last()]	Выбирает последний элемент cd в текущем узле
*/cd	Выбирает все cd элементы-внуки
/book/chapter[3]/para[1]	Выбирает первый элемент para третьего элемента chapter элемента book
//cd	Выбирает все элементы cd в документе
.	Выбирает текущий узел
.//c	Выбирает все элементы cd, расположенные ниже по отношению к текущему узлу
..	Выбирает родительский узел
../@src	Выбирает атрибут src родительского узла
cd[@type="classic"]	Все элементы cd со значением type, равным classic
cd[@type="classic"][5]	Пятый элемент cd со значением type, равным classic
cd[5][@type="classic"]	Выбирает пятый элемент cd, если его атрибут type равен classic

Таблица 4.7 *(окончание)*

Пример	Описание
`cd[@type and @country]`	Выбирает все элементы `cd`, обладающие атрибутами `type` и `country`

Выражения XPath

В XPath поддерживаются выражения числовые, равенства/неравенства, сравнений, логические.

Числовые выражения

Числовые выражения используются для произведения арифметических действий с числами (табл. 4.8).

Таблица 4.8. *Числовые выражения*

Оператор	Описание	Пример	Результат
`+`	Сложение	`6 + 4`	`10`
`-`	Вычитание	`6 - 4`	`2`
`*`	Умножение	`6 * 4`	`24`
`Div`	Деление	`8 div 4`	`2`
`Mod`	Модуль (остаток от деления)	`5 mod 2`	`1`

Перед выполнением числовых выражений каждый операнд преобразуется к числовому типу.

Выражения равенства/неравенства

Выражения используются для определения того, являются ли операнды равными или неравными (табл. 4.9).

Таблица 4.9. *Выражения равенства/неравенства*

Оператор	Описание	Пример	Результат (при значении `price 9.80`)
`=`	Равенство	`price=9.80`	`true`
`!=`	Неравенство	`price!=9.80`	`false`

Если происходит проверка равенства наборов узлов, то `TRUE` возвращается всегда, если набор содержит хотя бы один указанный узел. Проверка на не-

равенство возвращает TRUE всегда, если набор узлов содержит хотя бы один узел, отличный от указанного.

Выражения сравнения

Выражения сравнения используются для сравнения двух величин (табл. 4.10).

Таблица 4.10. *Выражения сравнения*

Оператор	Описание	Пример	Результат (при значении price 9.80)
<	Меньше чем	price<9.80	false
<=	Меньше или равно	price<=9.80	true
>	Больше чем	price>9.80	false
>=	Больше или равно	price>=9.80	true

Логические выражения

Логические выражения осуществляют операции логической конъюнкции и дизъюнкции (табл. 4.11).

Таблица 4.11. *Логические выражения*

Оператор	Описание	Пример	Результат (при значении price 9.80)
Or	ИЛИ	price=9.80 or price=9.70	TRUE
And	И	price<=9.80 and price=9.70	FALSE

Функции в XPath

В XPath присутствует библиотека функций для преобразования данных. Краткое описание этих функций содержится в табл. 4.12—4.15.

Таблица 4.12. *Функции для работы с узлами*

Функция	Описание
count()	Количество выбранных элементов
id()	Выбор элементов по идентификатору
last()	Номер последнего выбранного элемента
Local-name()	Локальная часть имени

Таблица 4.12 *(окончание)*

Функция	Описание
`name()`	Имя элемента
`namespace-uri()`	Пространство имен в полном имени
`position()`	Положение среди выбранных элементов

Таблица 4.13. *Функции для работы со строками*

Функция	Описание	Пример
`concat()`	Конкатенация	`concat('The',' ','XML')` Результат: `'The XML'`
`contains()`	`TRUE` — если первая строка содержит вторую, `FALSE` — если не содержит	`contains('XML','X')` Результат: `TRUE`
`normalize-space()`	Удаляет пробелы в начале и в конце строки	`normalize-space(' The XML')` Результат: `'The XML'`
`starts-with()`	`TRUE`, если первая строка начинается со второй строки	`starts-with('XML','X')` Результат: `TRUE`
`string()`	Преобразует объект в строку	`string(3.14)` Результат: `'3,14'`
`string-length()`	Возвращает длину строки	`string-length('Beatles')` Результат: `7`
`substring()`	Возвращает подстроку	`substring('Beatles',1,4)` Результат: `'Beat'`
`substring-after()`	Возвращает часть строки, следующую после второй строки	`substring-after('12/10','/')` Результат: `'10'`
`substring-before()`	Возвращает часть строки, предшествующую второй строке	`substring-before('12/10','/')` Результат: `'12'`
`translate()`	Заменяет символы в строке	`translate('12:30',':','!')` Результат: `'12!30'`

Таблица 4.14. *Функции для работы с числами*

Функция	Описание	Пример
`ceiling()`	Возвращает минимальное целое, не меньшее аргумента	`ceiling(3.14)` Результат: `4`
`Floor()`	Возвращает наибольшее целое, не большее аргумента	`floor(3.14)` Результат: `3`
`number()`	Преобразует в число	`number(price)`
`round()`	Возвращает ближайшее целое	`round(3.14)` Результат: `3`
`Sum()`	Возвращает сумму всех узлов, выбранных значением, указанным в аргументе	`sum(/cd/price)`

Таблица 4.15. *Функции для работы с логическими значениями*

Функция	Описание	Пример
`Boolean()`	Преобразует аргумент к логическому значению	—
`False()`	Возвращает `FALSE`	`number(false())` Результат: `0`
`Lang()`	—	—
`not()`	Возвращает `TRUE`, если аргумент `FALSE`, и наоборот	`not(false())`
`True()`	Возвращает `TRUE`	`number(true())` Результат: `1`

XML и MS SQL

Сервер MS SQL предоставляет возможность создавать SQL-запросы таким образом, чтобы получать в качестве ответа XML-текст. Работать с сервером MS SQL на языке PHP можно так же, как и с сервером MySQL.

В примере, приведенном в листинге 4.46, предполагается, что в базе данных есть таблица tovar, структура которой важна только в части отображения получаемого результата с использованием XSLT-преобразований.

Листинг 4.46. Файл mssql.php

```
</html>
<?
$db = new COM("ADODB.Connection");
```

```
$dsn = "DRIVER={SQL Server}; SERVER=ICL;UID=sa;PWD=matrox; DATABASE=itv";
$db->Open($dsn);
$rs = $db->Execute("SELECT * FROM tovar FOR XML AUTO, elements");
echo "<body>";
$var='<?xml version="1.0" encoding="Windows-1251"?> <top>';
while (!$rs->EOF)
{
   $var=$var.$rs->Fields[0]->Value;
   $rs->MoveNext();
}
$var=$var."</top>";
// echo $var;

// Открываем ресурс файла для записи
$file = fopen("1.xml", "w");

// Записываем измененный счетчик в файл
fwrite($file, $var);
// Закрываем ресурс файла
fclose($file);

   $xml = DOMDocument::loadXML($var);
//print $doc->saveXML();

$xsl = new DomDocument;
$xsl->load('1.xsl');

// Создаем XSL-процессор
$proc = new xsltprocessor;
// Указываем файл XSL-преобразований
$proc->importStyleSheet($xsl);
// Производим преобразования и отображаем результат
echo $proc->transformToXML($xml);
echo "<p>";
echo "Запрос: ";
echo "SELECT * FROM zak FOR XML AUTO, elements";
echo "</P>";
echo "<a href='1.xml'>XML-ответ MSSQL-сервера</a><br>";
echo "<a href='1.xsl'>Файл xsl-преобразовний</a><br>";
?>
</body>
</html>
```

Отображаемый в клиентском приложении HTML-код выглядит примерно так, как показано на рис. 4.55. В листинге 4.47 приведен файл XSL-преобразований.

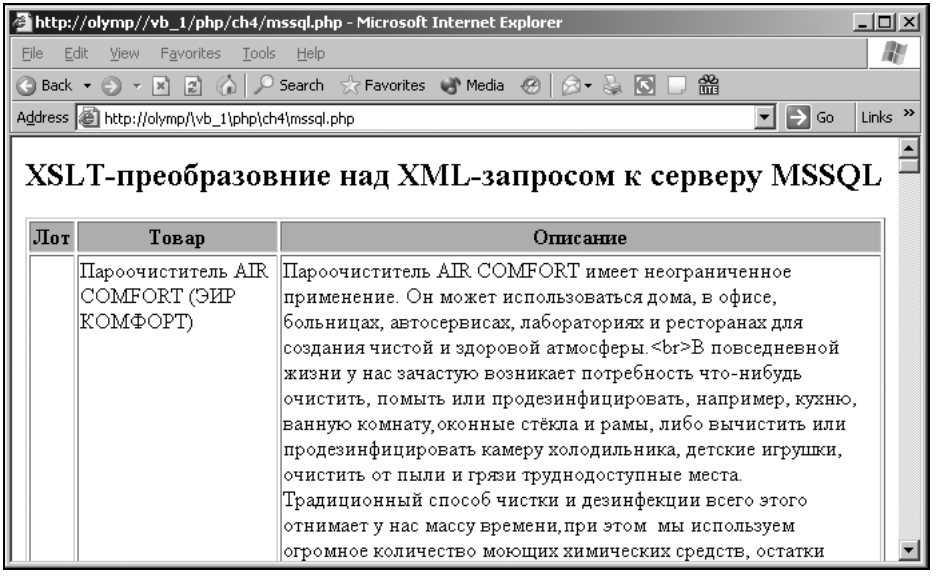

Рис. 4.55. Результат преобразования

Листинг 4.47. Файл xsl-преобразований (1.xsl)

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>XSLT-преобразование XML-запроса к серверу MSSQL</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th>Лот</th>
<th>Товар</th>
<th>Описание</th>

</tr>
<xsl:for-each select="top/tovar">
<tr>
<td valign="top"><xsl:value-of select="T_IDPOSTAVSHIKA"/> <br/>
<font color="blue"><xsl:value-of select="Z_TEL"/></font></td>
<td valign="top"><xsl:value-of select="T_TOVARTITLE"/></td>
<td valign="top"><xsl:value-of select="T_DESCRIPTION"/></td>
```

```
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

ГЛАВА 5

Сетевое программирование на PHP

Создание клиент-серверных приложений средствами PHP

Изначально язык PHP создавался с целью облегчить процесс создания серверных сценариев. В нем реализована функциональность, позволяющая решать наиболее часто встречающиеся при создании Web-приложений задачи. Причем разработчику предлагается богатый выбор функций, с помощью которых ранее громоздкие и необозримые построения могут быть заменены буквально несколькими строками кода на языке PHP.

PHP может работать в двух основных режимах: как независимый от Web-сервера CGI-модуль и как ISAPI-модуль, который расширяет функциональность Web-сервера и представлен в системе Windows в виде библиотеки DLL, при этом PHP работает внутри основного серверного процесса в Windows. CGI-модуль представлен в виде программы php.exe, он вызывается всякий раз при новом обращении к php-скриптам, инициализируя новый независимый от сервера процесс. В любом из этих случаев результат работы сценария возвращается серверу, который перенаправляет его запрашивающему Web-сервер клиенту.

Мы создадим два модуля:

- *серверный модуль*, который будет возвращать клиенту все, что будет от него получено, при этом он будет отображать в своем окне получаемые от клиента данные;
- *клиентский модуль*, который будет запрашивать данные, отображать их в окне, передавать их серверу, получать от сервера ответ и отображать его.

Современный язык PHP предоставляет возможность создавать приложения, работающие в режиме командной строки. При этом пользователь имеет возможность вводить данные по запросу программы непосредственно из

командной строки в то время, когда приложение работает. Этот факт мы используем для того, чтобы облегчить создание клиентского приложения.

Создание автономного сервера

Создаваемый серверный код может практически с равным успехом работать как обычное PHP-приложение, или как PHP-приложение в режиме командной строки. Современные версии языка PHP предлагают отдельный PHP-модуль, специально предназначенный для работы в режиме командной строки. Режим командной строки отличается тем, что автоматически не будут возвращаться HTML-заголовки, а также имеется возможность ввода данных из командной строки. Предназначенное для работы в командной строке PHP-приложение не требовательно: достаточно вместе с файлом php.exe иметь файл конфигурации php.ini и стандартную библиотеку php4ts.dll. Также в случае, если используются библиотеки-расширения, необходимо иметь эти библиотеки.

Для создания сервера и клиента необходимо использовать библиотеку, позволяющую работать на уровне сокетов (библиотеку php_sockets.dll). Чтобы включить эту библиотеку, следует вставить в файл конфигурации php.ini следующую строчку:

```
extension=php_sockets.dll
```

Наш сервер будет работать в локальной сети на компьютере с адресом 192.168.10.244. Сервер будет "слушать" порт 8082.

Заранее оговоримся, что мы не ставим перед собой целью создание полнофункционального приложения, нашей задачей является демонстрация возможности создания полноценного клиент-серверного приложения. В созданном нами примере будет оставлено достаточное количество мест, требующих дальнейшего улучшения. Наш сервер будет приспособлен принимать запросы только от одного клиента, и при создании полноценных приложений этот недостаток следует устранить, предусмотрев возможность работать с несколькими клиентами одновременно.

В листинге 5.1 приведен пример серверного кода.

Листинг 5.1. Серверный код (файл ssock4.pp)

```
<?php
error_reporting (E_ALL);

// Задаем неограниченное время работы.
set_time_limit (0);

$address = '192.168.10.244';
$port = 10000;
$value=true;
```

```
echo "TCP-server running on port '$port' at '$address'\n\rPowered
by PHP.\n\r\n\r";
if (($sock = socket_create (AF_INET, SOCK_STREAM, 0)) < 0) {
echo "socket_create() failed: reason: " . socket_strerror ($sock) . "\n";
}

if (($ret = socket_bind ($sock, $address, $port)) < 0) {
echo "socket_bind() failed: reason: " . socket_strerror ($ret) . "\n";
}

if (($ret = socket_listen ($sock, 5)) < 0) {
echo "socket_listen() failed: reason: " . socket_strerror ($ret) . "\n";
}
echo "Ready...\n\r\n\r";
do {
if (($msgsock = socket_accept($sock)) < 0) {
echo "socket_accept() failed: reason: " . socket_strerror ($msgsock) . "\n";
break;
}
/* Send instructions. */
$msg = "\nWelcome to the PHP Test Server. \n\r" .
"To quit, type 'quit'. To shut down the server type 'shutdown'.\n\r";
socket_write($msgsock, $msg, strlen($msg));

do {
// if (FALSE === ($buf = socket_read ($msgsock, 2048, PHP_NORMAL_READ))) {
if (FALSE === ($buf = socket_read ($msgsock, 1024))) {
echo "socket_read() failed: reason: " . socket_strerror ($ret) . "\n";
// break 2;
$value=false;
}
if (!$buf = trim ($buf)) {
continue;
}
if ($buf == 'quit') {
break;
}
if ($buf == 'shutdown') {
socket_close ($msgsock);
break 2;
}
$talkback = "$buf \n\r";
socket_write ($msgsock, $talkback, strlen ($talkback));
// echo "'$buf'\n\r";
```

```
echo $talkback;
} while ($value);
$value=true;
socket_close ($msgsock);
} while (true);

socket_close ($sock);
?>
```

Чтобы убедиться, что приложение работает, запустим сервер в режиме командной строки. Для этого в окне командной строки входим в директорию, где расположены файлы php.exe и ssock4.pp (в нашем случае они помещены в C:/php/cli), и выполняем команду `php.exe ssock4.pp` В окне командной строки видим информацию о том, что сервер начал функционировать (рис. 5.1).

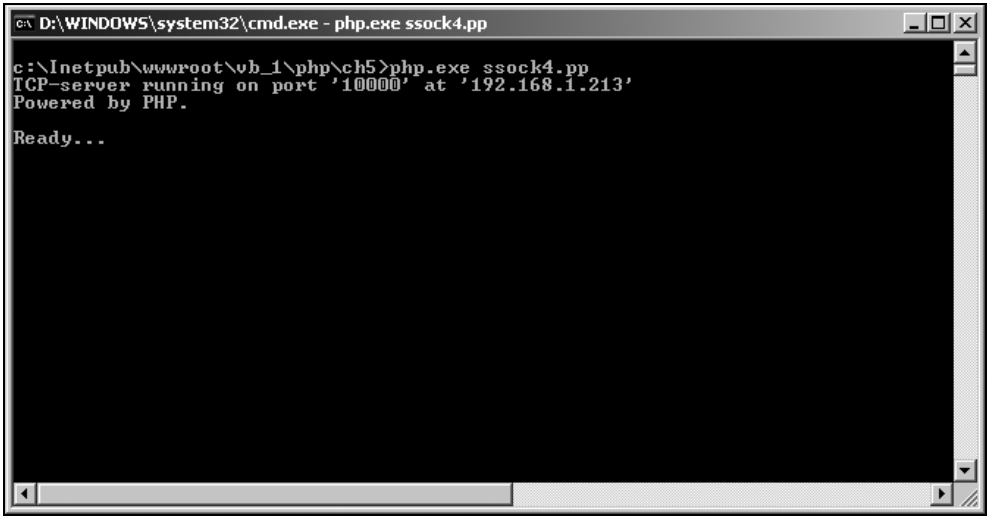

Рис. 5.1. Сервер запущен

Рассмотрим основные функции, используемые в приложении. Некоторые из них мы также будем использовать и при создании клиентского приложения.

Функция *socket_create*

Функция `socket_create ()` используется для создания сокета и возвращает идентификатор ресурса, являющегося одной из двух точек, между которыми осуществляется коммуникация. Обычное сетевое соединение состоит из двух сокетов, первый выполняет роль клиента, второй — роль сервера. Па-

раметр `domain` обозначает семейство протоколов, с которыми будет работать сокет. В табл. 5.1 приведены доступные значения параметров и описание протоколов.

Таблица 5.1. *Доступные протоколы*

Значения параметра `domain`	**Описание**
`AF_INET`	Протоколы IPv4 (включая TCP (Transmission Control Protocol) и UDP (User Datagram Protocol))
`AF_INET6`	Протокол IPv6, поддержка добавлена в версии PHP 5.0.0
`AF_UNIX`	Семейство локальных протоколов, используется для осуществления межпроцессовых коммуникаций IPC (Interprocess Communication)

Параметр `type` задает тип коммуникации, осуществляемой с помощью сокета. Типы сокетов представлены в табл. 5.2.

Таблица 5.2. *Типы сокетов*

Значения параметра `type`	**Описание**
`SOCK_STREAM`	Надежный дуплексный поток, используемый при работе с протоколом TCP
`SOCK_DGRAM`	Неподтверждаемый обмен дейтаграммами, используемый при работе с протоколом UDP
`SOCK_SEQPACKET`	Последовательный двусторонний обмен дейтаграммами фиксированной длины с подтверждением, при каждом чтении считывается полный пакет
`SOCK_RAW`	Используется для доступа к сетевым протоколам, при этом возможно создание произвольного типа протокола, полезен при создании ICMP-запросов (таких как `ping`, `traceroute`), используемых для диагностики работы сети
`SOCK_RDM`	Используется для надежного обмена дейтаграммами

Параметр `protocol` устанавливает конкретный протокол, используемый для работы с сокетом, при этом протокол должен соответствовать заданному значению семейства протоколов, указанному в параметре `domain`. Для протоколов TCP и UDP можно использовать константы `SOL_TCP` и `SOL_UDP` соответственно. Наиболее распространенные значения параметра `name` приведены в табл. 5.3.

Таблица 5.3. *Значения параметра* `name`

Параметр `name`	Описание
`ICMP`	Служебный протокол ICMP (Internet Control Message Protocol) для определения ошибок при передаче дейтаграмм. Примером использования этого протокола может служить программа Ping
`UDP`	Протокол обмена дейтаграммами UDP (User Datagram Protocol), длина дейтаграмм фиксирована, ненадежный протокол
`TCP`	Протокол с контролем передачи TCP (Transmission Control Protocol) — надежный протокол, основанный на принципе установления связи между участниками коммуникации. Протокол гарантирует передачу пакетов данных от источника к получателю, если происходит потеря пакета, то этот пакет посылается многократно до тех пор, пока получатель не подтвердит его получение

Функция `socket_create()` возвращает идентификатор ресурса в случае удачного создания сокета или `FALSE`, если возникает ошибка.

Функция *socket_bind*

После создания сокета его необходимо связать с интернет-адресом. Для этого используется функция `socket_bind (resource socket, string address [, int port])`, которой в качестве параметров передаются: идентификатор ресурса сокета, IP-адрес и порт. Функция возвращает `TRUE` при удачном установлении связи с заданным адресом и `FALSE` в случае возникновения ошибки.

Функция *socket_listen*

Функция `socket_listen` используется после функций `socket_create` и `socket_bind`. Эту функцию применяют для того, чтобы начать "прослушивание" сокета в ожидании входящих запросов. Синтаксис ее таков:

```
socket_listen ( resource socket [, int backlog])
```

В качестве параметров функция получает идентификатор ресурса сокета и максимальное количество входящих запросов, устанавливаемых в очередь для обработки.

Функция *socket_accept*

Стандартный механизм взаимодействия сокетов предполагает, что при получении запроса на соединение создается новый сокет, который используется для непосредственного осуществления коммуникации. В PHP для этого используется функция `socket_accept`, которая возвращает идентификатор

нового ресурса сокета или FALSE, если возникает ошибка. Возвращаемый этой функцией ресурс сокета не может быть использован для прослушивания новых входящих запросов на соединение. Функция socket_accept вызывается после функции socket_listen, исходный серверный сокет, созданный при помощи функции socket_create, может быть использован снова и снова. В качестве единственного параметра функции передается именно этот исходный сокет: socket_accept (resource socket).

Функция *socket_write*

Синтаксис функции:

```
socket_write ( resource socket, string buffer [, int length])
```

Функция осуществляет запись данных в сокет и возвращает количество записанных байтов. В качестве параметров указывается идентификатор ресурса сокета, созданного функцией socket_accept, строка buffer и (необязательный параметр) длина записываемой строки в байтах. Если заданная длина оказывается больше реальной длины передаваемой в строке buffer, то длина автоматически уменьшается до длины передаваемой строки.

Функция *socket_read*

Функция socket_read используется для чтения данных из сокета.

Синтаксис функции:

```
socket_read ( resource socket, int length [, int type])
```

В качестве параметров функции передается ресурс сокета, созданный функцией socket_accept, и количество считываемых байт. Чтение может быть прекращено также при получении символов \n, \r или \0, для этого нужно в качестве значения параметра type указать PHP_NORMAL_READ. По умолчанию используется значение PHP_BINARY_READ.

Создание клиентского приложения

Чтобы проверить сервер в работе, необходимо создать клиентское приложение. Его код приведен в листинге 5.2.

Листинг 5.2. Код приложения-клиента (файл csock4_1.pp)

```
<?php
error_reporting (E_ALL);

$service_port = 10000;
```

```
$address = '192.168.10.244';
echo "Making connection to $address:$service_port \n\r";
/* Create a TCP/IP socket. */
echo 'Creating socket...';
$socket = socket_create (AF_INET, SOCK_STREAM, 0);
if ($socket < 0) {
echo "socket_create() failed: reason: " . socket_strerror ($socket) .
"\n";
} else {
echo "OK.\n";
}

function communicate($line, $socket, $service_port, $is_created) {
$address = '192.168.10.244';
//echo $address;
//echo "Attempting to connect to '$address' on port '$service_port'...";
if (!$is_created) {
$result = socket_connect ($socket, $address, $service_port);
}
/*
if ($result < 0) {
echo "socket_connect() failed.\nReason: ($result) " .
socket_strerror($result) . "\n";
} else {
// echo "OK.\n";
}
*/

//$in = "HEAD / HTTP/1.0\r\n\r\n";
$in = "quit";
$out = '';

socket_write ($socket, $line, strlen ($line));

//echo "OK.\n";
//echo "Reading response:\n\n";
$out = socket_read ($socket, 1048);
echo $out;
//while ($out = socket_read ($socket, 2048)) {
// echo $out;
//}

}
//echo "Closing socket...";
//socket_close ($socket);
//echo "OK.\n\n";
//}
```

```
function read ($length='255')
{
if (!isset ($GLOBALS['StdinPointer']))
{
$GLOBALS['StdinPointer'] = fopen ("php://stdin","r");
}
$line = fgets ($GLOBALS['StdinPointer'],$length);
// return trim ($line);
return $line;
}
communicate("...", $socket, $service_port, false);
communicate(". .", $socket, $service_port, true);

communicate("|\n\r\n\r\n\r", $socket,$service_port, true);
// ' фрагмент с картинкой опущен '
do {
echo 'Enter whatever you want: ';
$test1=true;
do {
$entered = trim(read ());
if (!$entered == "") $test1=false;
if ($test1) echo "You have entered empty line. Please, try again.\n\r ";
} while ($test1);
//echo 'You have entered: '.$entered;

//echo '\n\r';
communicate($entered, $socket, $service_port, true);
} while (true)
?>
```

Чтобы удостовериться, что клиентское приложение успешно работает, запустим его в режиме командной строки:

```
php.exe csock4_1.pp
```

В окне клиентского приложения видим информацию о начале клиент-серверного взаимодействия и приглашение ввести пользовательскую информацию (рис. 5.2).

При этом в серверном окне также будут отображены принятые при старте сообщения (рис. 5.3).

После того как соединение установлено, мы можем вводить произвольную информацию в клиенте и отправлять ее серверу. Сервер будет возвращать клиенту то, что было получено, после чего клиент отображает ответ сервера в своем окне (рис. 5.4).

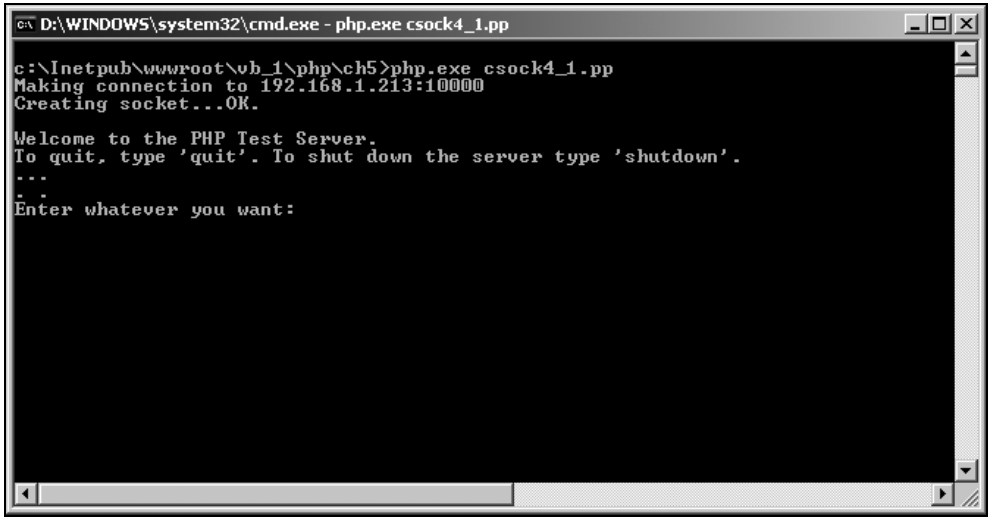

Рис. 5.2. Клиент стартовал

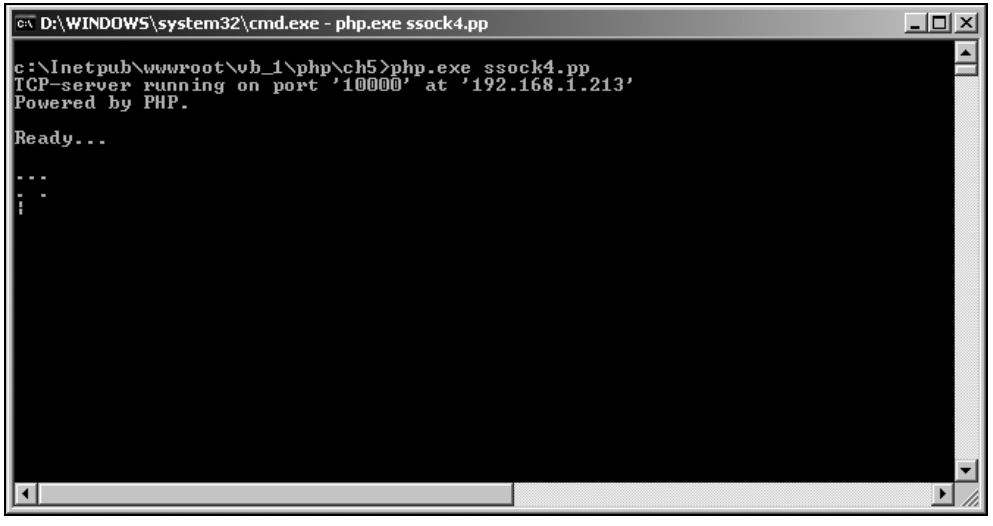

Рис. 5.3. Установлено взаимодействие сервера и клиента

Можно поэкспериментировать с сервером. Запустим сервер на локальной машине и обратимся к нему из браузера по адресу **http://192.168.1.213:10000**. Мы можем увидеть, какой HTTP-запрос сформирует наш браузер (рис. 5.5).

Этот пример показывает, как простое серверное приложение может служить полезным средством для тестирования создаваемых клиентских программ. Ответ сервера будет отображен в браузере (рис. 5.6).

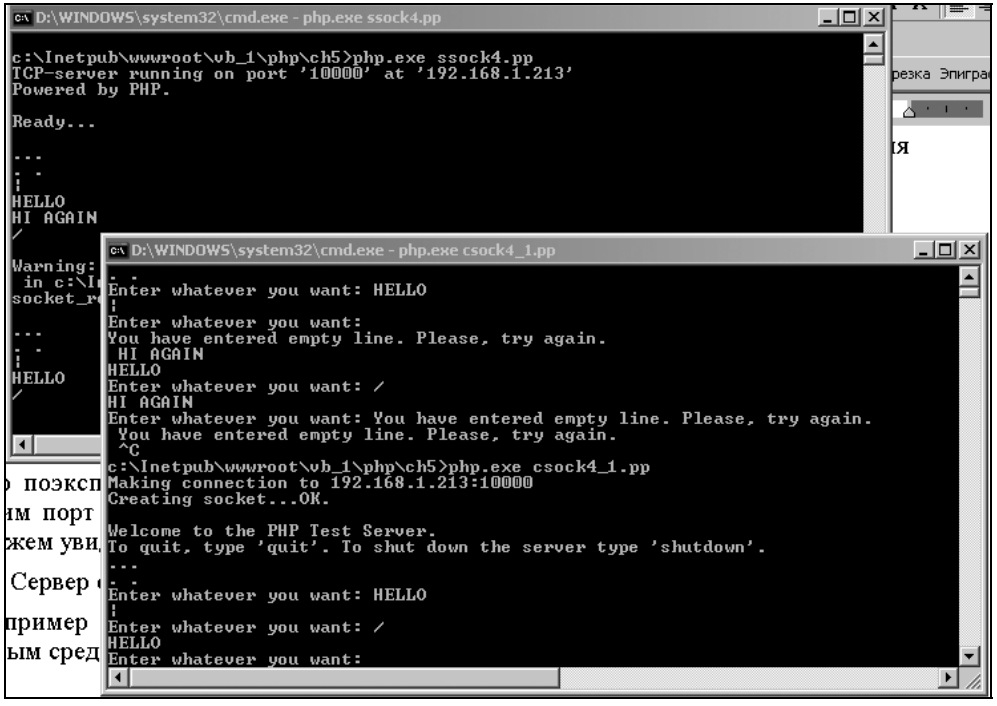

Рис. 5.4. Клиент "беседует" с сервером

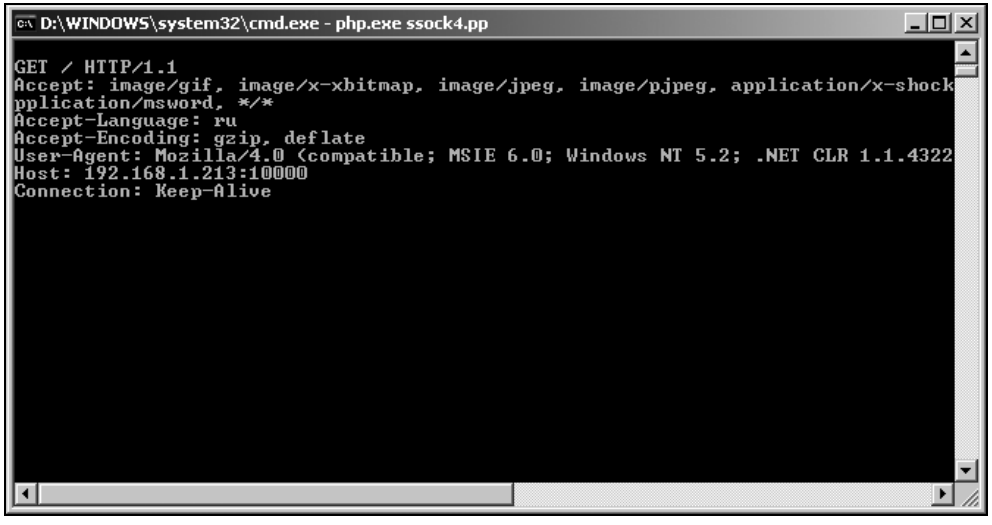

Рис. 5.5. Сервер отобразил запрос браузера

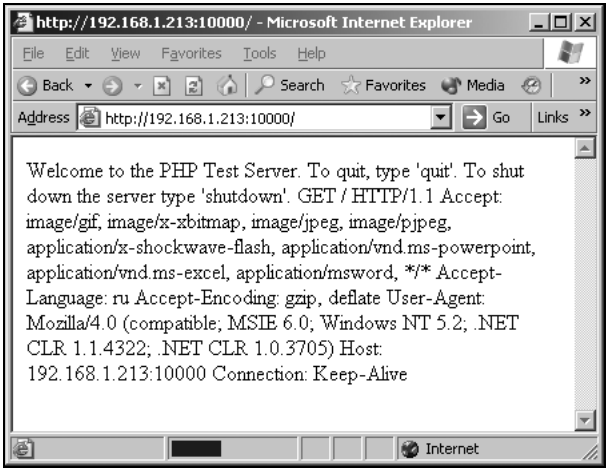

Рис. 5.6. Ответ браузера

Создание Web-служб средствами языка PHP

PHP позволяет создавать Web-службы, причем для облегчения задачи разработки служб можно воспользоваться готовыми библиотеками. Размещенная в сети служба предоставляет программный интерфейс, который может быть использован в Web-приложениях и доступен по сети Интернет. В этой главе рассмотрен пример создания простейшей Web-службы с использованием библиотеки nusoap средствами языка PHP.

Web-службы были задуманы и развиваются как механизм, призванный превратить сеть Интернет в распределенный инструмент, предоставляющий постоянно растущий набор API, который реализуется на основе простого протокола доступа к объектам (SOAP), передаваемого, как правило, поверх протокола HTTP. Web-служба предоставляет объект или набор объектов и набор методов, доступных для такого объекта (или объектов). Объекту передается запрос с использованием протокола SOAP, после получения запроса объект выполняет заданный метод с переданными параметрами и возвращает результат Web-клиенту, направившему запрос. В основе всей коммуникации лежит обычное клиент-серверное взаимодействие с использованием сокетов. Web-служба располагается на машине провайдера Web-сервиса, который также должен позаботиться о том, чтобы тем или иным способом заказчики были оповещены о структуре предоставляемого интерфейса, а также указать URL, по которому доступен предлагаемый API.

Провайдер Web-службы имеет возможность поставлять готовые решения, обеспечивающие необходимый доступ к службе в соответствии с требова-

ниями заказчика. Альтернативой готовым решениям является детальное описание объектов и методов, доступных в данной службе. При этом заказчик сам разрабатывает клиентское программное обеспечение, при помощи которого будет осуществляться обращение к службе.

Типичным клиентским решением является "тонкий" клиент, предоставляющий базовую функциональность (мануальный ввод данных, отправка данных службе), работающий с использованием механизмов Web-интерфейса, клиентских HTML-форм, получаемых клиентом службы и передаваемых службе. Однако ничто не мешает созданию полнофункциональных клиентских приложений, при этом следует учитывать, что получение ответа от Web-службы может потребовать значительного времени. Для того чтобы облегчить разработку Web-служб и создание клиентов, можно воспользоваться уже готовыми библиотеками. Для языка PHP существует библиотека nusoap (**http://dietrich.ganx4.com/nusoap/index.php**).

Файл nusoap.php (несколько видоизмененная библиотека nusoap) должен быть включен как в серверную, так и в клиентскую часть приложения. Рассмотрим пример создания Web-службы.

Web-службы

Общий вид запроса к службе представляет собой HTTP-сообщение, телом которого является XML-текст запроса в формате SOAP (листинг 5.3).

Листинг 5.3. Запрос к службе

```
POST /vb_1/php/soap/server1.php HTTP/1.0
User-Agent: NuSOAP/0.6.3
Host: olymp
Content-Type: text/xml; charset="ISO-8859-1"
Content-Length: 568
SOAPAction: ""

<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  x
mlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd">
<SOAP-ENV:Body><ns1:calc xmlns:ns1="http://testuri.org">
<rate xsi:type="xsd:string">2323</rate>
<sub xsi:type="xsd:string">12</sub>
</ns1:calc>
```

```
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Ответ Web-службы выглядит примерно так:

```
HTTP/1.1 200 OK
Content-Length: 515
Content-Type: text/xml; charset=UTF-8
Server: Microsoft-IIS/6.0
X-Powered-By: PHP/5.0.1
Server: NuSOAP Server v0.6.3
MicrosoftOfficeWebServer: 5.0_Pub
X-Powered-By: ASP.NET
Date: Mon, 04 Oct 2004 06:48:21 GMT
Connection: close

<?xml version="1.0" encoding="ISO-8859-1"?>
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:si="http://soapinterop.org/xsd">
<SOAP-ENV:Body>
<calcResponse>
<soapVal xsi:type="xsd:float">
290.76
</soapVal>
</calcResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Как видим, запрос выглядит весьма просто: мы передаем службе цену и ставку налога, и ожидаем от нее число, равное цене с учетом налога. Для этого в запросе создаются элементы, описываемые тэгами `<rate>` и `<sub>` (`sub` обозначает subtotal — промежуточная сумма). Ответ службы также прост, в нем содержится элемент, телом которого является ожидаемый нами результат. Все что нужно знать разработчику клиента — это в каком виде передать службе запрос (какие XML-элементы должны быть переданы и что в них должно содержаться), а также в каком виде будет получен ответ (какие элементы будут получены в теле HTTP-ответа и что будет содержаться в этих элементах).

SOAP-сообщение содержит также обязательные заголовки. Сейчас мы не будет останавливаться на рассмотрении деталей протокола SOAP, однако

эти детали будут необходимы, если возникнет задача создания Web-служб вручную. Знание протокола SOAP также практически необходимо и при работе с уже готовыми библиотеками.

Всю функциональность Web-службы, анализатор запросов и формирование ответов можно довольно легко реализовать собственными силами без использования каких-либо готовых механизмов. Сейчас мы воспользуемся библиотекой nusoap.php.

Создаем Web-службу

Web-служба получает цену в виде значения элемента `<sub>` и ставку налога — значение элемента `<rate>`, которые передаются методу `Calc`. Все, что нам потребуется сделать, — это создать новый серверный объект (описан в библиотеке nusoap.php), имплементировать функцию `Calc`, зарегистрировать эту функцию в созданном объекте, и наконец, запустить серверный объект, заставив его работать при помощи метода `service($HTTP_RAW_POST_DATA)` (описан в библиотеке). После этого служба готова к работе.

Код службы приведен в листинге 5.4.

Листинг 5.4. Файл server1.php

```
<?php

// Вставляем библиотеку NuSOAP
require_once('nusoap.php');
// Создание объекта сервера
$server = new soap_server;
// Регистрация метода сервиса
$server->register('calc');
// Создание метода
function calc($rate,$sub) {

$result=$sub*$rate/100.+$sub;
    return $result;
}
$HTTP_RAW_POST_DATA = isset($HTTP_RAW_POST_DATA) ? $HTTP_RAW_POST_DATA :
'';
// Обращение к сервису
$server->service($HTTP_RAW_POST_DATA);
?>
```

Служба готова. Она будет расположена по адресу **http://localhost/vb_1/php/soap/server1.php**.

Наш клиент Web-службы выполнен в виде PHP-файла. Он получает данные из HTML-формы (цену и ставку налога). Клиент формирует запрос к серверу, получает ответ, формирует HTML-вывод. В программе-клиенте (листинг 5.5) необходимо сформировать массив передаваемых службе параметров, создать клиентский объект, вызвать службу, указав метод и передаваемые параметры.

Листинг 5.5. Файл client1.php

```
<?php

// Выводить HTML-форму или нет?
//$sub=100;
//$rate=12;
$sub =$_REQUEST['sub'];
$rate =$_REQUEST['rate'];
if (!$sub) {echo "<H1> Введите цену и ставку налога</h1><form
submit=soap.php><P>Цена: <input type=text name='sub'><P>Ставка налога:
<input type=text name='rate'><input type=submit></form>";}
else {
// Инициализируем параметры, передаваемые службе, на основе
// переменных, переданных HTML-формой
$parameters = array('rate'=>$_GET['rate'], 'sub' => $_GET['sub']);
// Вставляем библиотеку NuSOAP
require_once('nusoap.php');
// Создаем объект клиента
$sclient = new soapclient('http://olymp/vb_1/php/soap/server1.php');
// Вызываем функцию сервиса, передаем ей массив параметров
$result = $sclient->call('calc', $parameters);
// Отображаем результат
echo "Цена с налогом $".$result.".";
// Отображаем запрос и ответ
echo '<h2>Запрос</h2>';
echo '<pre>' . htmlspecialchars($sclient->request, ENT_QUOTES) . '</pre>';
echo '<h2>Ответ</h2>';
echo '<pre>' . htmlspecialchars($sclient->response, ENT_QUOTES) . '</pre>';
}
?>
```

Проверим службу в работе. Пусть клиент доступен по адресу **http://olymp/vb_1/php/soap/client1.php**.

При первом обращении к клиенту нам будет предложено ввести цену и ставку налога (рис. 5.7). Укажем, например, значения 365 и 8. После нажа-

тия кнопки отправки формы клиент обратиться к службе, от которой получит ответ. В окне браузера мы увидим значение, равное цене с учетом налога (рис. 5.8). На рис. 5.9 и 5.10 также показаны запрос к Web-службе и ответ Web-службы.

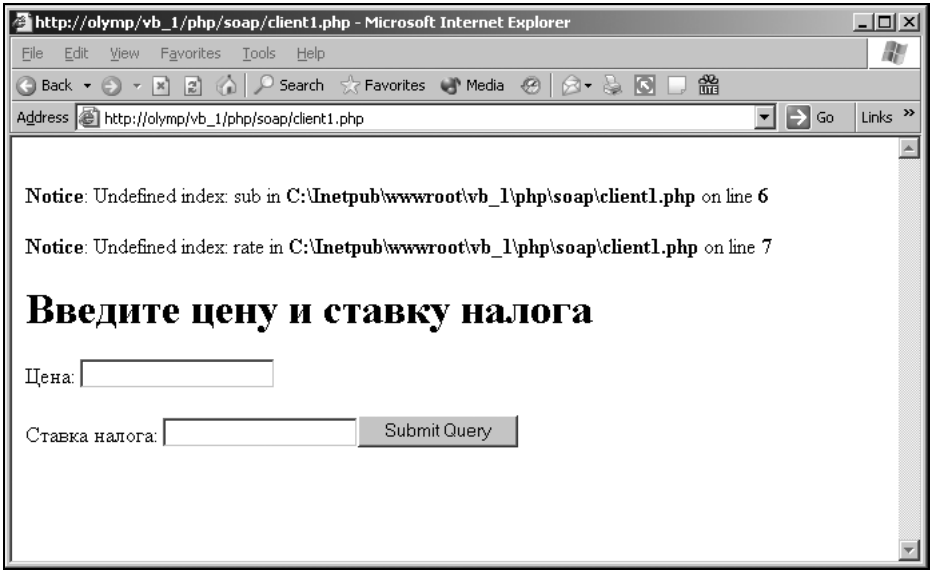

Рис. 5.7. Обращаемся к клиенту

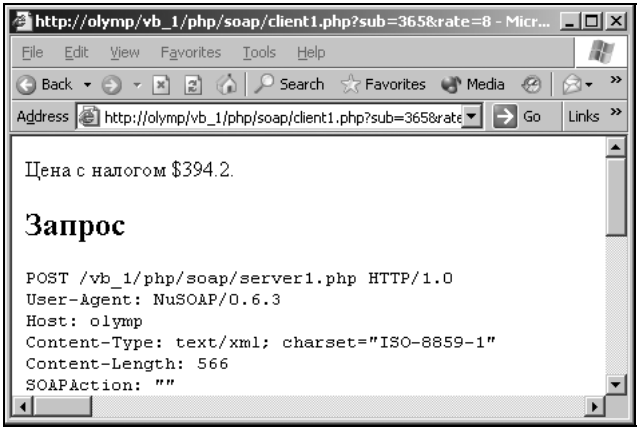

Рис. 5.8. От службы получена цена с учетом налога

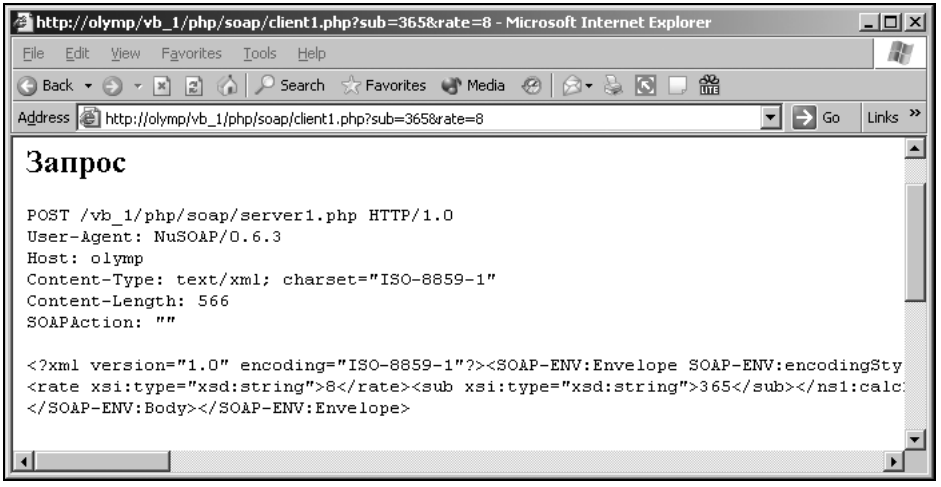

Рис. 5.9. Запрос к службе

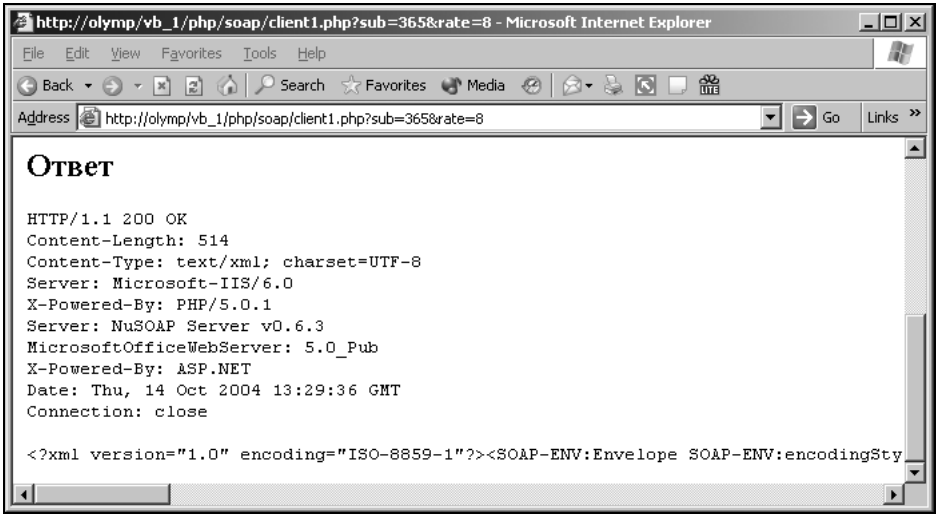

Рис. 5.10. Ответ службы

Доступ к простым объектам SOAP

Аббревиатура SOAP означает Simple Objects Access Protocol — протокол доступа к простым объектам. Протокол SOAP использует формат XML. Обмен информацией происходит по протоколу HTTP. SOAP предназначен для обмена информацией на уровне приложений через сеть Интернет, причем он независим от платформы и от языков программирования. SOAP отличается

простотой и возможностью создания расширений. Консорциум W3C принял этот протокол как стандартный.

Ранее мы уже рассматривали пример создания Web-службы средствами языка PHP, и в общих чертах познакомились с устройством протокола SOAP. Сейчас мы более подробно рассмотрим его "внутренне" устройство.

Протокол SOAP предназначен для передачи поверх протокола HTTP. Протокол HTTP используется во всех клиентских и серверных приложениях, работающих с Web. SOAP позволяет различным программам, работающим на разных платформах, написанных на разных языках с использованием различных технологий, общаться друг с другом.

Синтаксис SOAP

SOAP основан на сообщениях. Сообщение SOAP представляет собой документ XML, в состав которого входят следующие элементы:

- конверт SOAP, в него вкладывается содержимое сообщения;
- заголовок SOAP, который не является обязательным элементом;
- тело SOAP содержит информацию ответов и запросов.

Основные правила синтаксиса достаточно просты, они перечислены ниже:

- сообщение SOAP использует синтаксис XML;
- сообщение SOAP должно иметь единственный конверт;
- сообщение SOAP может иметь заголовок;
- сообщение SOAP должно иметь тело сообщения;
- сообщение SOAP использует пространства имен конверта SOAP;
- сообщение SOAP использует пространства имен кодировок;
- сообщение SOAP не должно иметь ссылок на DTD;
- сообщение SOAP не должно иметь инструкций обработки XML.

В качестве основного пространства имен конверта сообщений SOAP по умолчанию используется **http://www.w3.org/2001/12/soap-envelope**. В качестве основного пространства имен кодировок сообщений SOAP по умолчанию используется **http://www.w3.org/2001/12/soap-encoding**.

В листинге 5.6 приведен пример запроса SOAP.

Листинг 5.6. Запрос SOAP

```
<soap:Envelope>
<soap:Body>
<GetPrice>
```

```
<Item>Pivo</Item>
</GetPrice>
</soap:Body>
</soap:Envelope>
```

Элементы SOAP

В качестве корневого элемента в сообщениях SOAP используется элемент <Envelope>. Дочерними элементами для него являются элементы <Header>, <Body>, <Fault>.

Общий план сообщения SOAP приведен в листинге 5.7.

Листинг 5.7. Каркас сообщения SOAP

```
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
...
...
</soap:Header>
<soap:Body>
...
...
<soap:Fault>
...
...
</soap:Fault>
</soap:Body>
</soap:Envelope>
```

Элемент *<Envelope>*

Элемент <Envelope> — это корневой элемент сообщения SOAP. Этот элемент определяет, что данный XML-документ является сообщением SOAP. В качестве пространства имен указывается соответствующее имя конверта SOAP. Это пространство имен используется с конвертами SOAP.

Конверт SOAP

```
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
...
```

```
Message information goes here
...
</soap:Envelope>
```

Элемент <Header>

Заголовок <Header> — это необязательный элемент. В нем указывается дополнительная вспомогательная информация о сообщении SOAP:

```
<soap:Header>
<m:country xmlns:m="http://www.e-olymp.com/oblast/">
<m:language>X</m:language>
<m:currency>XYU</m:currency>
</m:country>
</soap:Header>
```

Отметим, что элементы <language> и <currency> — это пользовательские элементы. Эти элементы не являются частью стандарта SOAP.

Элемент <Body>

Основным элементом сообщения SOAP является обязательный элемент <Body>. Этот элемент содержит тело сообщения.

```
<soap:Body>
<m:GetPrice xmlns:m="http://www.e-olymp.com/prices/">
<m:Item>Apples</m:Item>
</m:GetPrice>
</soap:Body>
```

Здесь также присутствуют пользовательские элементы, а именно элементы <GetPrice> и <Item>. Эти элементы не являются частью стандарта SOAP.

Элемент <Fault>

Элемент <Body> в качестве дочернего элемента может содержать элемент <Fault>. Элемент <Fault> используется для задания информации об ошибках, произошедших при обработке сообщения. Этот элемент может быть использован только в сообщениях-ответах. Приведем пример использования этого элемента:

```
<soap:Body>
<soap:Fault>
<faultcode>soap:Server</faultcode>
<faultstring>Oshibka, odnako!</faultstring>
```

```
</soap:Fault>
</soap:Body>
```

Атрибуты SOAP

Элемент SOAP может иметь следующие атрибуты:

- `actor`;
- `encodingStyle`;
- `mustUnderstand`.

Атрибут *actor*

Атрибут `actor` задает имя URI, к которому относится элемент заголовка `<Header>`.

Синтаксис:

```
soap:actor="URI"
```

Пример с этим атрибутом приведен в листинге 5.8.

Листинг 5.8. Пример с атрибутом `actor`

```
<soap:Header>
<m:local xmlns:m="http://www.e-olymp.com/local/"
soap:actor="http://www.e-olymp.com/appml">
<m:language>en</m:language>
<m:currency>USD</m:currency>
</m:local>
</soap:Header>
```

Атрибут *encodingStyle*

Атрибут `encodingStyle` используется для задания типов данных в документе.

Синтаксис:

```
soap:encodingStyle="URI"
```

Пример с атрибутом `encodingStyle` приведен в листинге 5.8.

Листинг 5.8. Пример с атрибутом `encodingStyle`

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
...
```

```
Здесь само сообщение
...
</soap:Envelope>
```

Атрибут *mustUnderstand*

Атрибут `mustUnderstand` определяет, должен ли получатель сообщения обрабатывать элемент заголовка сообщения.

Синтаксис:

```
soap:mustUnderstand="boolean"
```

Пример с атрибутом `mustUnderstand` приведен в листинге 5.9.

Листинг 5.9. Пример с атрибутом `mustUnderstand`

```
<soap:Header>
<m:local xmlns:m="http://www.e-olymp.com/local/">
<m:language soap:mustUnderstand="0">en</m:language>
<m:currency soap:mustUnderstand="1">USD</m:currency>
</m:local>
</soap:Header>
```

Пример работы с SOAP

Создадим пример запроса `GetStockPrice`, который посылается серверу (листинг 5.10). Этот запрос имеет параметр `StockName`. Ответ (листинг 5.11) содержит параметр `Price`. Пространство имен, определяемое префиксом `m`, задано в виде адреса **http://www.stock.org**.

Листинг 5.10. Запрос SOAP

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap; charset=utf-8

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.stock.org/stock">
<m:GetStockPrice>
<m:StockName>IBM</m:StockName>
</m:GetStockPrice>
```

```
</soap:Body>
</soap:Envelope>
```

Листинг 5.11. Ответ SOAP

```
HTTP/1.1 200 OK
Connection: close
Content-Type: application/soap; charset=utf-8
Date: Sat, 12 May 2002 08:09:04 GMT
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.stock.org/stock">
<m:GetStockPriceResponse>
<m:Price>34.5</m:Price>
</m:GetStockPriceResponse>
</soap:Body>
</soap:Envelope>
```

Ошибки SOAP

Сообщения об ошибках записываются в ответе SOAP в элементе `<Fault>`. Элемент `<Fault>` вкладывается в элемент `<Body>`. Элемент `<Fault>` может встречаться в сообщении SOAP только один раз. Он может включать в себя дочерние элементы (табл. 5.4). Коды ошибок представлены в табл. 5.5.

Таблица 5.4. *Ошибки в SOAP*

Элемент	Описание
`<faultcode>`	Код ошибки
`<faultstring>`	Строка ошибки
`<faultactor>`	Что вызвало ошибку
`<detail>`	Информация об ошибке

Таблица 5.5. *Коды ошибок*

Ошибка	Описание
`VersionMismatch`	Неправильное пространство имен элемента конверта SOAP
`MustUnderstand`	Атрибут `mustUnderstand` имеет значение 1, но элемент в заголовке не распознан

Таблица 5.5 *(окончание)*

Ошибка	Описание
`Client`	Сообщение создано некорректно или содержит некорректную информацию
`Server`	Проблемы на сервере, сообщение не может быть обработано

Заключение

В рамках экспресс-курса мы не можем рассмотреть в подробностях детали программирования на языке PHP 5, и многие вопросы остались неосвещенными. Мы познакомились с некоторыми подходами, которые несомненно могут быть полезны в практической работе. Мы старались увидеть сущность предмета на примерах, которые можно использовать для дальнейшего изучения, доработки и применения.

В книге не приведено детального описания важных функций языка PHP 5 и методов их использования. Подробную документацию читатель может найти в сети Интернет на сайтах **http://www.php.net** и **http://www.zend.com**.

Предметный указатель

Z

А

Б

В

Г

Д

З

И

К

Л

М

О

П

Р

С

Т

Ф

Ц

Ч

Э