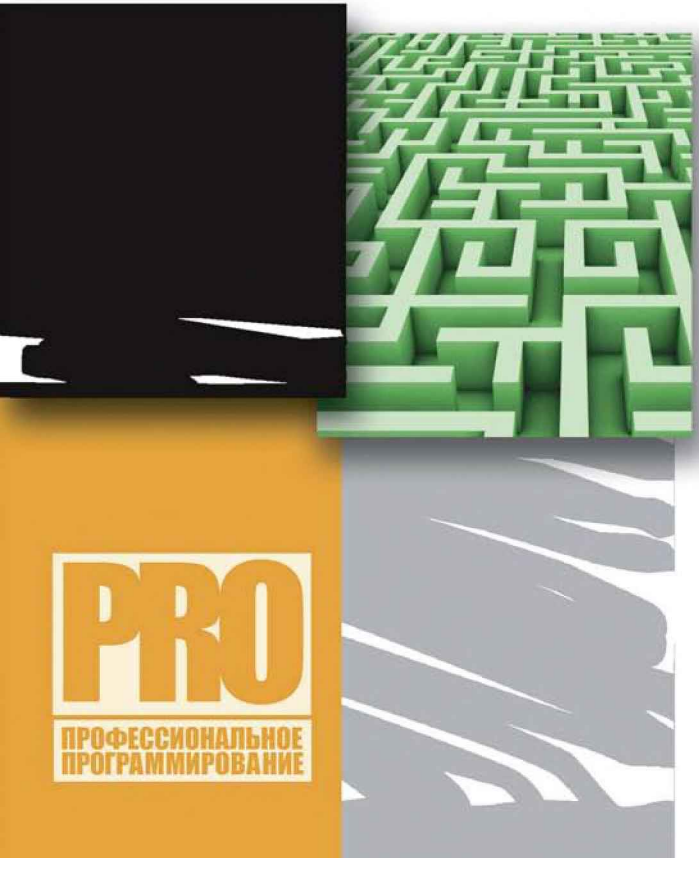
Н. А. Прохоренок, В. А. Дронов

bhv®

HTML, JavaScript, PHP и MySQL

Джентльменский набор Web-мастера

4-е издание



- HTML 5 и CSS 3
- Web-СЕРВЕР Apache
- АУДИО И ВИДЕО
- АНИМАЦИЯ
- ПРЕОБРАЗОВАНИЯ
- AJAX
- ГЕОЛОКАЦИЯ
- ЛОКАЛЬНОЕ ХРАНИЛИЩЕ ДАННЫХ
- ТРАНЗАКЦИИ MySQL
- ПРИМЕРЫ И СОВЕТЫ ИЗ ПРАКТИКИ

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Материалы на www.bhv.ru

Николай Прохоренок
Владимир Дронов

HTML, JavaScript, PHP и MySQL

Джентльменскнй набор Web-мастера

4-е издание

Санкт-Петербург
«БХВ-Петербург»
2015

УДК 004.43+004.738.5
ББК 32.973.26-018.1
П84

Прохоренок, П. А.

П84 HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. — 4-е изд., перераб. и доп. / Н. А. Прохоренок, В. А. Дронов. — СПб.: БХВ-Петербург, 2015. — 768 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-3130-6

Рассмотрены вопросы создания интерактивных Web-сайтов с помощью HTML, JavaScript, PHP и MySQL, форматирования Web-страниц при помощи CSS. Даны основы PHP и примеры написания типичных сценариев. Описаны приемы работы и администрирования баз данных MySQL при помощи PHP и программы phpMyAdmin. Особое внимание уделено созданию программной среды на компьютере разработчика и настройке Web-сервера Apache. Приведено описание текстового редактора Notepad++, шаблонизатора Smarty и прочих программ (Aptana Studio, NetBeans и HeidiSQL), необходимых Web-разработчику.

В 4-м издании содержится описание возможностей, предлагаемых HTML 5 (средства семантической разметки и размещения аудио и видео) и CSS 3 (градиенты, создание тени, анимация и преобразования), технологии AJAX, формата JSON, новых инструментов JavaScript (включая средства геолокации и локальное хранилище данных) и всех нововведений, появившихся в актуальных на данный момент версиях Apache, PHP и MySQL.

Электронный архив содержит листинги примеров, руководство по созданию динамического сайта, самоучитель языка Perl, руководство по публикации сайта, инструкции по установке дополнительных программ и видеоуроки.

Для Web-разработчиков

УДК 004.43+004.738.54
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*

Подписано в печать 30.04.15.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 61,92.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-3130-6

Оглавление

Введение

Если вы хотите научиться своими руками создавать сайты, свободно владеть HTML, CSS, JavaScript, PHP и MySQL, то эта книга для вас. Большинство подобных книг предлагают изучение или только клиентских технологий (HTML, CSS, JavaScript), или только серверных (PHP, MySQL). Но разделять эти инструменты нельзя, т. к. они могут существовать только совместно, а значит, и изучать их нужно как единое целое.

Все главы книги расположены в порядке возрастания уровня сложности материала. Если вы начинающий Web-мастер, то книгу следует изучать последовательно, главу за главой. Если же материал какой-либо из глав был изучен ранее, то можно сразу переходить к следующей главе.

Что же можно создать с использованием описываемых технологий? Давайте кратко рассмотрим возможности этих технологий, а также содержание глав книги.

Язык разметки HTML, рассматриваемый в *главе 1*, позволяет задать местоположение элементов Web-страницы в окне Web-браузера. С помощью HTML можно отформатировать отдельные символы или целые фрагменты текста, вставить изображение, таблицу или форму, создать панель навигации с помощью карт-изображений, разделить окно Web-браузера на несколько областей, вставить гиперссылку и многое другое. А новая версия языка HTML — HTML 5 — даже позволяет поместить на страницу аудио- или видеоролик, который будет воспроизводиться самим Web-браузером, без необходимости устанавливать какие бы то ни было плагины.

При помощи каскадных таблиц стилей (CSS), о которых идет речь в *главе 2*, можно задавать точные характеристики практически всех элементов Web-страницы. Это позволяет контролировать внешний вид Web-страницы в окне Web-браузера и приближает возможности Web-дизайна к настольным издательским системам. Разработчик может указать параметры шрифта, цвет текста и фона, выравнивание, создать рамку и расположить элементы на странице произвольным образом. Новая версия CSS — CSS 3 — также предоставляет инструменты для задания градиентного фона, теней у текста и самого элемента страницы и даже для создания анимации.

У Web-страниц, созданных с использованием HTML и CSS, есть существенный недостаток — они являются статическими, т. е. не могут меняться, реагируя на действия пользователя. Внедрение программ, написанных на языке JavaScript, в HTML-код позволит "оживить" Web-страницу, сделать ее интерактивной или, другими словами, заставить взаимодействовать с пользователем. С помощью JavaScript можно обрабатывать данные формы до отправки на сервер, получать информацию о Web-браузере пользователя и его мониторе и соответствующим образом менять форматирование страницы, создавать новые окна, изменять любые элементы HTML-документа в ответ на какое-либо событие, создавать часы на Web-странице, показывающие текущее время с точностью до секунды, скрывать или отображать элементы Web-страницы и выполнять многие другие действия. Как все это сделать, рассказано в *главе 3*.

В *главе 3* также описывается технология AJAX, позволяющая программно подгружать с сервера произвольные данные без перезагрузки самой страницы. Это могут быть как фрагменты HTML-кода, выводимые на страницу непосредственно, так и данные, закодированные в формате JSON и предназначенные для использования в JavaScript-программах. Применение технологии AJAX позволит значительно расширить функциональность создаваемых сайтов.

Глава 4 повествует, как установить и настроить специальное программное обеспечение для тестирования сайтов: Web-сервер Apache, среду для выполнения серверных скриптов, написанных на языке PHP, и сервер баз данных MySQL. Таким образом, можно проверить работоспособность создаваемого сайта непосредственно на своем компьютере, еще до его публикации в Интернете.

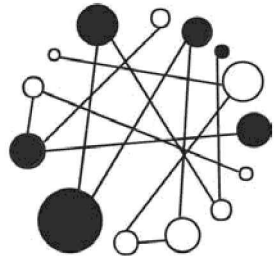
Огромные возможности открывают серверные технологии, среди которых для целей данной книги выбран язык программирования PHP. Это наиболее распространенный в настоящее время язык для написания серверных скриптов. Используя его (или другие программные платформы, применяемые для создания динамических Web-страниц), можно изменять HTML-код, получаемый Web-браузером, в зависимости от вводимых пользователем данных, типа и версии установленного Web-браузера и других факторов. Большое количество расширений и готовых программных продуктов, а также легкость освоения сделали PHP очень популярным языком программирования для Интернета. С помощью PHP можно работать с файлами и каталогами, обрабатывать данные формы на сервере, рассылать письма, загружать файлы на сервер, создавать для каждого пользователя Личный кабинет, программировать гостевые книги, форумы, блоги, интернет-магазины и многое другое. Писать программы на PHP мы научимся в *главе 5*.

На сегодняшний день ни один крупный портал не обходится без использования баз данных. В Web-разработках чаще всего применяется быстрая, бесплатная и обладающая большими возможностями система управления базами данных (СУБД) MySQL. С помощью MySQL можно эффективно добавлять, изменять и удалять данные, получать пужпую информацию по запросу. Работа с MySQL, в том числе с базами данных этого формата из программ, написанных на PHP, обсуждается в *главе 6*.

В электронном приложении, доступном на сайте издательства "БХВ-Петербург", собраны следующие материалы:

- описание процесса установки и настройки специализированных редакторов, которые позволят значительно упростить создание сайта и сделают процесс изучения материала книги более эффективным;
- описание процесса публикации сайта в Интернете (подбор подходящей площадки, работа с FTP-клиентом, настройка Web-сервера Apache, выполнение автоматического зануска программ в определенное время, подготовка сайта к индексации и т. д.);
- дополнительные руководства (описание фильтров и преобразований, которые доступны в Web-браузере Internet Explorer, и электронный самоучитель языка Perl);
- описание процесса разработки полнофункционального Web-сайта с использованием всех изученных технологий. Это каталог сайтов, включающий личный кабинет для пользователей с защитой средствами PHP, а также личный кабинет для администратора, защищенный средствами сервера Apache;
- все листинги, встречающиеся в тексте книги.

Авторы желают приятного прочтения и надеются, что эта книга станет верным снутником в вашей программистской практике.



ГЛАВА 1

Основы HTML. Создаем дизайн сайта

1.1. Основные понятия

HTML (HyperText Markup Language) — это язык разметки документа, описывающий форму отображения информации на экране компьютера.

При создании документа часто приходится выделять какую-либо часть текста полужирным шрифтом, изменять размер или цвет шрифта, выравнивать текст по центру страницы и т. д. В текстовом редакторе для этого достаточно выделить нужный фрагмент и применить к нему форматирование. Например, чтобы пометить текст курсивом, нужно выделить его и нажать кнопку **Курсив**. На языке HTML тот же эффект достигается следующей строкой кода:

```
<i>Текст</i>
```

Символ `<i>` указывает, что текст надо выделить, начиная с этого места, а `</i>` отмечает конец выделенного фрагмента.

Символы `<i>` и `</i>` принято называть *тегами*. С помощью тегов описывается вся структура документа. Теги выделяются угловыми скобками "<" и ">", между которыми указывается имя тега. Большинство тегов являются парными, т. к. есть открывающий тег (`<i>`) и соответствующий ему закрывающий (`</i>`). Закрывающий тег отличается наличием косой черты ("/") перед его именем. Есть также теги, вообще не имеющие закрывающего тега, например тег переноса строки `<br>`.

Некоторые теги могут иметь *параметры* (иногда их называют *атрибутами*). Параметры указываются после имени тега через пробел в формате `параметр="значение"`. Если параметров несколько, то они перечисляются через пробел. Например:

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
```

В этом примере параметру `http-equiv` тега `<meta>` присвоено значение `Content-Type`, а параметру `content` — значение `text/html; charset=windows-1251`.

Теги могут вкладываться друг в друга. Например:

```
<p><i>Текст</i></p>
```

При вложении тегов необходимо соблюдать последовательность их закрытия. Например, такой код использовать нельзя:

```
<p><i>Текст</p></i>
```

> ***Примечание***
>
> В HTML названия тегов и параметров можно записывать в любом регистре, а в языке XHTML только в нижнем регистре.

Просматривать HTML-документы можно с помощью специальных программ, которые называют Web-браузерами. Web-браузеры отображают документы с форматированием, выполненным на основе исходного кода, описывающего структуру документа.

Результат интерпретации HTML-документа, отображаемый в окне Web-браузера, называется Web-страницей. В отличие от HTML-документа Web-страница может содержать не только текст, но и графику, видео, звуковое сопровождение, может реагировать на действия пользователя и т. д. Кроме того, Web-страница может быть результатом интерпретации сразу нескольких HTML-документов.

Документы в формате HTML имеют расширение html или htm.

Прежде чем изучать язык HTML, советуем установить на компьютер один из редакторов — FCKeditor или tinyMCE. Эти редакторы написаны на языке программирования JavaScript и работают в Web-браузере.

Скачать FCKeditor можно со страницы **http://ckeditor.com/download**. После распаковки архива занустите файл sample07.html (расположен в папке fckeditor_samples\html\). Если вы используете Web-браузер Firefox, то для работы редактора необходимо выполнить следующие действия:

1. В адресной строке вводим `about:config` и нажимаем клавишу <Enter>.
2. Находим директиву `security.fileuri.strict_origin_policy` и двойным щелчком на строке устанавливаем значение `false`.

На рис. 1.1 можно увидеть, как выглядит редактор FCKeditor, запущенный в Web-браузере Firefox. Если вы раньше работали с текстовым редактором Microsoft Word, то большинство кнопок на панели инструментов будет вам знакомо. Принцип работы в FCKeditor точно такой же, как и в Word. После ввода текста и его форматирования редактор автоматически генерирует HTML-код. Посмотреть исходный HTML-код можно нажав кнопку **Псточннк** на панели инструментов (рис. 1.2). Следует заметить, что при изменении исходного HTML-кода автоматически изменяется и внешний вид документа.

Скачать tinyMCE можно со страницы **http://tinymce.moxiecode.com/download.php**. После загрузки распаковываем архив в текущую папку. Для русификации редактора со страницы **http://tinymce.moxiecode.com/download_i18n.php** необходимо скачать архив с файлами для русского языка. Архив следует разместить в папке tinymce\jscripts\tiny_mce\, а затем распаковать в текущую папку. Все файлы будут автоматически распределены по каталогам. Чтобы подключить поддержку русского

языка, необходимо в файле full.html (расположен в папке tinymce\examples\) добавить строку

```
language: "ru",
```

сразу после строки

```
tinyMCE.init({
```

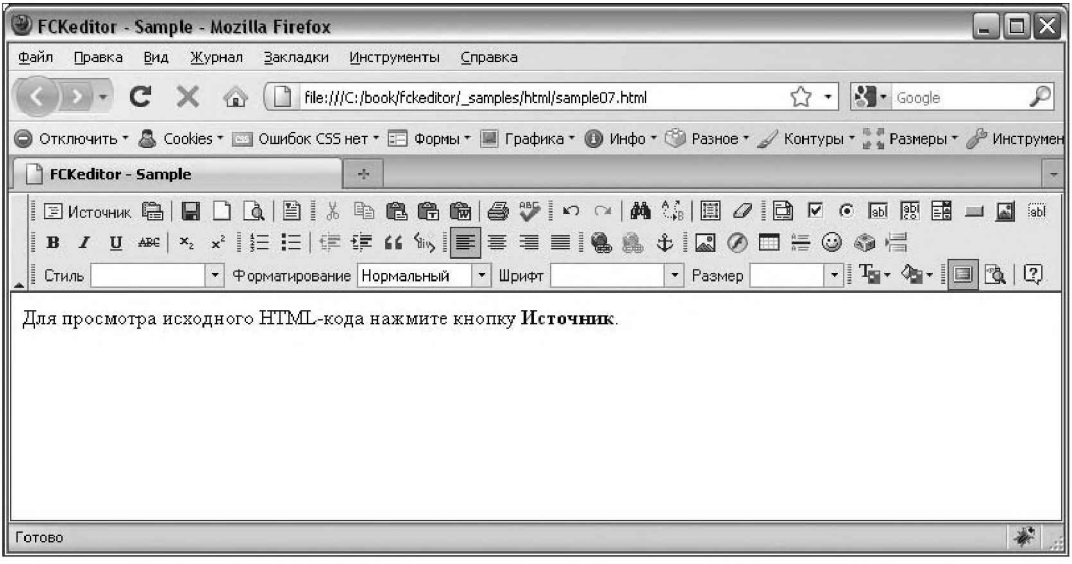


Рис. 1.1. Редактор FCKeditor, запущенный в Web-браузере Firefox

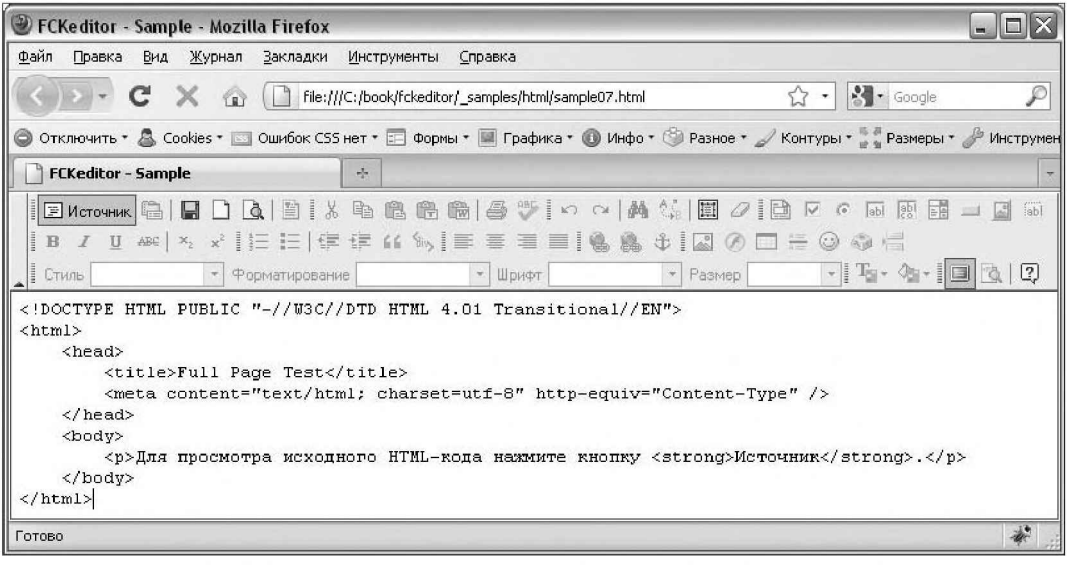


Рис. 1.2. Результат нажатия кнопки **Источник** в редакторе FCKeditor

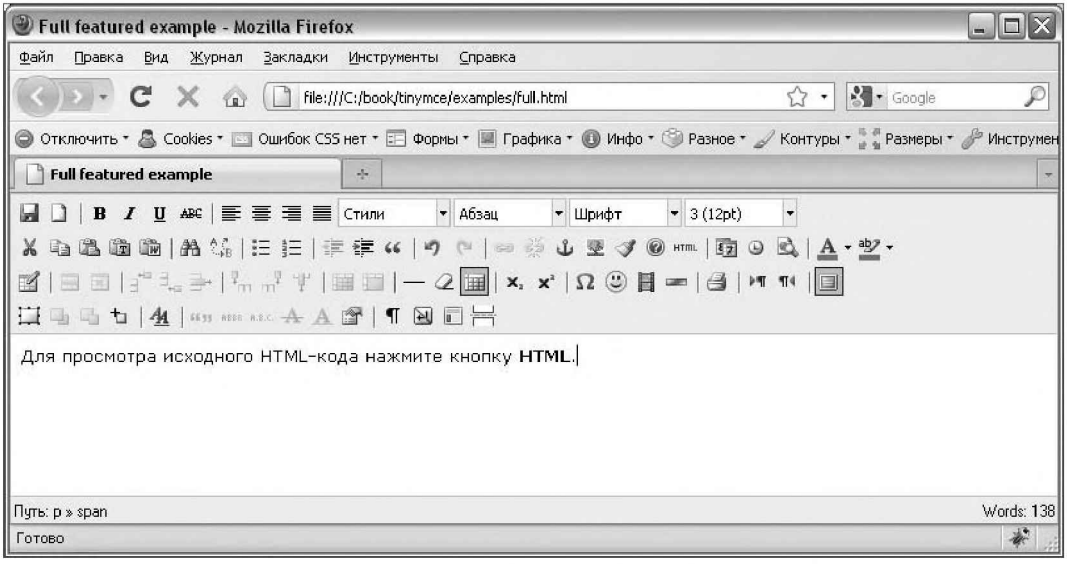


Рис. 1.3. Редактор tinyMCE, запущенный в Web-браузере Firefox

Теперь файл full.html открываем с помощью Web-браузера. На рис. 1.3 можно увидеть, как выглядит редактор tinyMCE, запущенный в Web-браузере Firefox.

1.2. Первый HTML-документ

Попробуем создать наш первый HTML-документ. Для его создания можно воспользоваться любым текстовым редактором. Самым распространенным редактором является обычный Блокнот. Открываем Блокнот и набираем содержимое листинга 1.1.

Листинг 1.1. Первый HTML-документ

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Заголовок страницы</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
 <p>
  <strong>Этот текст выделен полужирным шрифтом</strong>
 </p>
</body>
</html>
```

Сохраняем введенный текст в формате HTML, например, под именем test.html. Для этого в меню **Файл** выбираем нункт **Сохраннть как**. В открывшемся окне в строке **Пмя файла** вводим "test.html", а в списке **Тнн файла** указываем **Все файлы**. Выбираем папку, например, Рабочий стол, и нажимаем **Сохраннть**. Закрываем Блокнот.

> ***Примечание***
>
> Если в списке **Тип файла** оставить **Текстовые документы (*.txt)**, то в строке **Имя файла** название файла необходимо заключить в кавычки, иначе к имени файла будет добавлено расширение txt.

Занускаем Web-браузер, например, Internet Explorer. С помощью пункта **Открыть** меню **Файл** открываем сохраненный файл test.html. Если все сделано правильно, то в окне Web-браузера будет показана выделенная надпись "Этот текст выделен полужирным шрифтом", а в строке заголовка будет надпись "Заголовок страницы — Microsoft Internet Explorer". Теги в окне Web-браузера не отображаются!

Теперь попробуем изменить заголовок в окне Web-браузера. Для этого необходимо открыть исходный текст в формате HTML. Это можно сделать тремя способами:

- в меню **Внд** выбрать нункт **Просмотр ПТML-кода**;
- правой кнопкой мыши щелкнуть в любом месте окна Web-браузера. В появившемся контекстном меню выбрать пункт **Просмотр ПТML-кода**;

> ***Примечание***
>
> В некоторых случаях результат этих двух действий может быть разным. Если Web-страница состоит из нескольких HTML-документов, то первый способ отобразит только код структуры Web-страницы, а не исходный код каждого из HTML-документов. Второй способ позволяет отобразить исходный код лишь одного HTML-документа, а от места щелчка зависит, код какого HTML-документа будет отображен. В нашем случае результат будет одним и тем же.

- открыть файл, содержащий исходный код, с помощью Блокнота или другого текстового редактора. Этот способ является самым универсальным. Настоятельно рекомендую использовать именно его.

В итоге исходный текст будет доступен для редактирования. Изменим строчку

```
<title>Заголовок страницы</title>
```

на

```
<title>Моя первая Web-страница</title>
```

и сохраним файл (меню **Файл**, нункт **Сохраннть**). Теперь вернемся в Web-браузер и обновим Web-страницу. Обновить можно следующими способами:

- в меню **Внд** выбрать нункт **Обновнть**;
- выбрать этот же пункт в контекстном меню;
- нажать кнопку **Обновнть** на Панели инструментов;
- на клавиатуре нажать клавишу <F5>.

В результате строка заголовка изменится на "Моя первая Web-страница — Microsoft Internet Explorer".

Таким образом, изменяя что-либо в исходном коде, можно визуально оценивать результаты произведенных действий. Алгоритм такой: открываем исходный код, вносим корректировку, сохраняем, а затем обновляем Web-страницу.

> ***Примечание***
>
> Необходимо заметить, что все описанные действия возможны только для локально сохраненных HTML-документов. Если HTML-документ опубликован в Интернете, то можно лишь созерцать исходный код, а вот изменить его таким способом нельзя.

Очень хорошей альтернативой Блокноту является программа Notepad++. Она позволяет корректно работать как с кодировкой windows-1251, так и с кодировкой UTF-8, а также имеет подсветку синтаксиса HTML, JavaScript, PHP и др. Именно этой программой мы будем пользоваться на протяжении всей книги.

Скачать программу Notepad++ можно абсолютно бесплатно со страницы **http://notepad-plus.sourceforge.net/ru/site.htm**. Из двух вариантов (ZIP-архив и инсталлятор) советую выбрать именно инсталлятор, т. к. при установке можно будет указать язык интерфейса программы. Установка Notepad++ предельно проста и в комментариях не нуждается.

Занускаем программу Notepad++. В меню **Коднровкн** устанавливаем флажок **Коднровать в ANSI**. Набираем код, представленный в листинге 1.1, а затем в меню **Файл** выбираем нункт **Сохраннть как**. В открывшемся окне в строке **Пмя файла** вводим "test.html". Выбираем папку, например Рабочий стол, и нажимаем **Сохраннть**. Для просмотра открываем файл с помощью Web-браузера.

Чтобы открыть какой-либо файл на редактирование, в меню **Файл** выбираем нункт **Открыть** или щелкаем правой кнопкой мыши на ярлыке файла в Проводнике Windows и из контекстного меню выбираем пункт **Edit with Notepad++**.

> ***Примечание***
>
> Вместо Notepad++ можно воспользоваться редакторами PHP Expert Editor, Aptana Studio или NetBeans. Эти редакторы помимо подсветки синтаксиса предоставляют множество дополнительных функций. Тем не менее для быстрого редактирования файла удобнее пользоваться Notepad++. Описание редакторов вы найдете в *главе 4*.

1.3. Структура документа

Итак, мы изучили технологию создания HTML-документов, научились сохранять, отображать и изменять исходный код. Пришла пора вернуться к языку HTML. В листинге 1.2 представлена структура, характерная для любого HTML-документа.

Листинг 1.2. Структура HTML-документа

```
<!DOCTYPE> <!-- Объявление формата документа -->
<html>
```

```
    <head>
        <!-- Техническая информация о документе -->
    </head>
    <body>
        <!-- Основная часть документа -->
    </body>
</html>
```

Тег `<!DOCTYPE>` позволяет определить Web-браузеру формат файла и правильно отобразить все его инструкции. Допустимые форматы для HTML 4.01:

- `Strict` — строгий формат. Не содержит тегов и параметров, помеченных как устаревшие или не одобряемые. Объявление формата:

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

- `Transitional` — переходный формат. Содержит устаревшие теги в целях совместимости и упрощения перехода со старых версий HTML. Объявление формата:

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

- `Frameset` — аналогичен переходному формату, но содержит также теги для создания фреймов. Объявление формата:

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

Если тег `<!DOCTYPE>` не указан, то Web-браузер Internet Explorer переходит в режим совместимости (Quirks Mode). В этом режиме отличается тип блочной модели. Поэтому при отсутствии тега `<!DOCTYPE>` разные Web-браузеры могут по-разному отображать Web-страницу.

> ***Примечание***
>
> Более подробную информацию о типах блочной модели можно получить в Интернете на странице консорциума W3C **http://www.w3.org/TR/CSS2/box.html** и на странице **http://www.quirksmode.org/css/quirksmode.html**.

Весь текст HTML-документа расположен между тегами `<html>` и `</html>`. HTML-документ состоит из двух разделов — заголовка (между тегами `<head>` и `</head>`) и содержательной части (между тегами `<body>` и `</body>`).

1.3.1. Раздел *HEAD*. Техническая информация о документе

Раздел `HEAD` содержит техническую информацию о странице — заголовок, ее описание и ключевые слова для поисковых машин, данные об авторе и времени создания страницы, базовом адресе страницы, кодировке и т. д.

Единственным обязательным тегом в разделе HEAD является тег <title>. Текст, расположенный между тегами <title> и </title>, отображается в строке заголовка Web-браузера. Длина заголовка должна быть не более 60 символов, иначе он полностью не поместится в заголовке Web-браузера:

```
<title>Заголовок страницы</title>
```

> ***Совет***
>
> Очень часто текст между тегами <title> и </title> используется в результатах, выдаваемых поисковым порталом, в качестве текста ссылки на эту страницу. По этой причине заголовок должен максимально полно описывать содержание страницы. Не следует писать что-то вроде "Главная страница", "Первая страница" и т. п.

С помощью одинарного тега <meta> можно задать описание содержимого страницы и ключевые слова для поисковых машин. Если текст между тегами <title> и </title> используется в качестве текста ссылки на эту страницу, то описание из тега <meta> будет отображено под ссылкой:

```
<meta name="description" content="Описание содержимого страницы">
<meta name="keywords" content="Ключевые слова через запятую">
```

Можно также указать несколько описаний на разных языках. Для этого в параметре lang следует указать используемый язык:

```
<meta name="description" lang="ru" content="Описание содержимого страницы">
<meta name="description" lang="en" content="Description">
<meta name="keywords" lang="ru" content="Ключевые слова через запятую">
<meta name="keywords" lang="en" content="Keywords">
```

Кроме того, тег <meta> позволяет запретить или разрешить индексацию Web-страницы поисковыми машинами:

```
<meta name="robots" content="<Индексация>, <Переход по ссылкам>">
```

В параметре content указывается комбинация следующих значений:

- index — индексация разрешена;
- noindex — индексация запрещена;
- follow — разрешено переходить по ссылкам, которые находятся на этой Web-странице;
- nofollow — запрещено переходить по ссылкам;
- all — комбинация index плюс follow;
- none — комбинация noindex плюс nofollow.

Приведем ряд примеров. Индексация и переход по ссылкам разрешены:

```
<meta name="robots" content="index, follow">
```

Индексация разрешена, а переход по ссылкам запрещен:

```
<meta name="robots" content="index, nofollow">
```

Индексация и переход по ссылкам запрещены:

```
<meta name="robots" content="noindex, nofollow">
```

Также с помощью тега <meta> можно указать кодировку текста:

```
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
```

Для автоматической перезагрузки страницы через заданный промежуток времени следует воспользоваться свойством refresh тега <meta>:

```
<meta http-equiv="refresh" content="30">
```

В этом примере страница будет перезагружена через 30 секунд. Если необходимо сразу перебросить посетителя на другую страницу, то можно указать URL-адрес в параметре url:

```
<meta http-equiv="refresh" content="0; url=http://mail.ru/">
```

> ***Примечание***
>
> В разделе HEAD могут быть расположены также теги <base>, <link>, <script>, <style> и некоторые другие. Эти теги мы рассмотрим по мере изучения материала.

1.3.2. Раздел *BODY*. Основная часть документа

В этом разделе располагается все содержимое документа. Большинство тегов, рассмотренных в этой главе книги, должны находиться именно между тегами <body> и </body>.

Следует отметить, что в формате Strict содержимое тега <body> должно быть расположено внутри блочных элементов, например, <p>, или др.:

```
<body>
 <p>Текст документа</p>
 <div>Текст документа</div>
</body>
```

Тег <body> имеет следующие параметры:

- bgcolor задает цвет фона Web-страницы. Даже если цветом фона является белый, все равно следует указать цвет.

  Цвет определяется цифрами в шестнадцатеричном коде. Для каждой составляющей цвета (красного, зеленого и синего) задается значение в пределах от 00 до FF. Эти значения объединяются в одно число, перед которым добавляется символ "#", например, значение #FF0000 соответствует красному цвету, #00FF00 — ярко-зеленому, а #FF00FF — фиолетовому (смеси красного и синего);

- background позволяет задать фоновый рисунок для документа путем указания URL-адреса изображения;

- alink определяет цвет активной ссылки;

- `link` устанавливает цвет еще не просмотренных ссылок;
- `vlink` определяет цвет уже просмотренных ссылок;
- `text` устанавливает цвет текста.

Например, тег `<body>` может выглядеть так:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Заголовок страницы</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body alink="#FF0000" link="#000000" vlink="#000080" text="#000000">
Текст документа
</body>
</html>
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Все рассмотренные в этом разделе параметры являются устаревшими и поддерживаются только в формате `Transitional`. Использование их в формате `Strict` недопустимо.

Существуют и другие параметры, которые мы будем рассматривать по мере изучения языка.

1.4. Форматирование отдельных символов

Как уже говорилось, HTML — это язык разметки. Следовательно, важно уметь форматировать отдельные символы, а также целые фрагменты текста. Но прежде чем изучать теги, рассмотрим возможность отображения специальных символов. Такими символами, например, являются знаки меньше (<) и больше (>), т. к. с помощью этих символов описываются HTML-теги. Для отображения специальных символов используются так называемые *HTML-эквиваленты*. Например, для вывода такого текста

```
Текст между тегами <title> и </title> используется в результатах,
выдаваемых поисковым порталом.
```

необходимо написать так

```
Текст между тегами &lt;title&gt; и &lt;/title&gt; используется
в результатах, выдаваемых поисковым порталом.
```

В этом примере мы заменили знак меньше (<) на `<`, а знак больше (>) — на `>`. Перечислим наиболее часто используемые HTML-эквиваленты:

- `<` — знак меньше (<);
- `>` — знак больше (>);

- `&` — амперсанд (&);
- ` ` — неразрывный пробел;
- `&quot;` — кавычка (");
- `&copy;` — знак авторских прав (©);
- `&reg;` — знак зарегистрированной торговой марки (®);
- `&trade;` — торговая марка (™).

1.4.1. Выделение фрагментов текста

Тег `<b>` отображает текст полужирным шрифтом:

```
<b>Полужирный шрифт</b>
```

Вместо тега `<b>` лучше использовать тег логического форматирования `<strong>`:

```
<strong>Полужирный шрифт</strong>
```

Тег `<i>` отображает текст курсивом:

```
<i>Текст, выделенный курсивом</i>
```

Вместо тега `<i>` лучше использовать тег логического форматирования `<em>`:

```
<em>Текст, выделенный курсивом</em>
```

Тег `<u>` отображает текст подчеркнутым:

```
<u>Подчеркнутый текст</u>
```

Теги `<strike>` и `<s>` отображают текст перечеркнутым:

```
<strike>Перечеркнутый текст</strike>
<s>Перечеркнутый текст</s>
```

> ***Обратите внимание***
>
> Теги `<u>`, `<strike>` и `<s>` являются устаревшими и поддерживаются только в формате `Transitional`. Использование их в формате `Strict` недопустимо.

1.4.2. Создание нижних и верхних индексов

Тег `<sub>` сдвигает текст ниже уровня строки и уменьшает размер шрифта. Он используется для создания нижних индексов, например, H_2O:

```
Формула воды H<sub>2</sub>O
```

Тег `<sup>` сдвигает текст выше уровня строки и уменьшает размер шрифта. Этот тег используется чаще всего для создания степеней, например, $м^2$:

```
Единица измерения площади – м<sup>2</sup>
```

1.4.3. Вывод текста заданным шрифтом

Тег `<font>` определяет размер, тип и цвет шрифта. Он имеет следующие параметры:

- `face` служит для указания типа шрифта:

```
<font face="Verdana">Текст</font>
```

  Можно указать как один, так и несколько типов, разделяя их занятыми. При этом список шрифтов просматривается слева направо. Указанное название должно точно соответствовать названию типа шрифта. Если шрифт не найден на компьютере пользователя, то используется шрифт по умолчанию;

- `size` задает размер шрифта в условных единицах от 1 до 7. Размер, используемый Web-браузером по умолчанию, принято приравнивать к 3. Размер шрифта можно указывать как цифрой от 1 до 7, так и в относительных единицах, указывая, на сколько единиц пужно увеличить (знак "+") или уменьшить (знак "–") размер шрифта относительно базового:

```
<font size="4">Текст</font>
<font size="+1">Текст</font>
<font size="-1">Текст</font>
```

- `color` позволяет указывать цвет шрифта. Цвета задаются так же, как для параметра `bgcolor` *(см. разд. 1.3.2)*:

```
<font color="#FF0000">Текст</font>
```

  Вместо цифр можно использовать названия цветов:

```
<font color="red">Текст</font>
```

  Перечислим названия наиболее часто используемых цветов:

  - `black` — `#000000` — черный;
  - `white` — `#FFFFFF` — белый;
  - `yellow` — `#FFFF00` — желтый;
  - `silver` — `#C0C0C0` — серый;
  - `red` — `#FF0000` — красный;
  - `green` — `#008000` — зеленый;
  - `gray` — `#808080` — темно-серый;
  - `blue` — `#0000FF` — синий;
  - `navy` — `#000080` — темно-синий;
  - `purple` — `#800080` — фиолетовый.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Тег `<font>` является устаревшим и поддерживается только в формате `Transitional`. Использование его в формате `Strict` недопустимо.

Также для форматирования текста применяются и другие теги. Для вывода текста шрифтом большего размера используется парный тег <big>:

```
Текст <big>большего</big> размера
```

А для вывода текста шрифтом меньшего размера применяется парный тег <small>:

```
Текст <small>меньшего</small> размера
```

Для вывода текста моноширинным шрифтом используется тег <tt>:

```
<tt>Моноширинный шрифт</tt>
```

1.5. Форматирование документа

Практически все теги, рассмотренные в предыдущем разделе, являются тегами физического форматирования. Исключение составляют теги и . Эти теги являются тегами логического форматирования текста и используются для выделения очень важных и просто важных фрагментов соответственно. Теги логического форматирования используются для структурной разметки документа и могут отображаться разными Web-браузерами по-разному. Перечислим основные теги логического форматирования:

- <cite>...</cite> — применяется для отметки цитат, а также названий произведений;
- <code>...</code> — служит для отметки фрагментов программного кода;
- <acronym>...</acronym> — используется для отметки аббревиатур;
- <kbd>...</kbd> — отмечает фрагмент как вводимый пользователем с клавиатуры;
- <q>...</q> — используется для отметки коротких цитат;
- <samp>...</samp> — применяется для отметки результата, выдаваемого программой;
- <var>...</var> — отмечает имена переменных.

1.5.1. Тег комментария

Текст, заключенный между тегами <!-- и -->, не отображается Web-браузером. Заметим, что это нестандартная пара тегов, т. к. открывающий тег не имеет закрывающей угловой скобки, а в закрывающем теге отсутствует открывающая угловая скобка:

```
<!-- Текст -->
```

> ***Совет***
>
> Использование комментариев в исходном коде позволит быстро найти нужный фрагмент. Это особенно важно для начинающих Web-дизайнеров.

1.5.2. Перевод строки

Для разделения строк используется одинарный тег <br>.

Если в HTML-документе набрать текст

```
Строка1
Строка2
Строка3
```

то Web-браузер отобразит его в одпу строку: "Строка1 Строка2 Строка3". Для того чтобы строки располагались друг под другом, необходимо добавить тег
 в конец каждой строки:

```
Строка1<br>
Строка2<br>
Строка3<br>
```

Для вывода текста в том же виде, что и в исходном коде, можно воспользоваться парным тегом <pre>:

```
<pre>
Строка1
Строка2
Строка3
</pre>
```

В этом примере строки также будут располагаться друг под другом.

1.5.3. Горизонтальная линия

Одинарный тег <hr> позволяет провести горизонтальпую линию.

Тег <hr> имеет следующие параметры:

- `size` — толщина линии:

  ```
  <hr size="5">
  ```

- `width` — длина линии. Можно указывать значение как в пикселах, так и в процентах относительно ширины окна Web-браузера:

  ```
  <hr size="5" width="100">
  <hr size="5" width="100%">
  ```

- `align` — выравнивание линии. Параметр может принимать следующие значения:
  - `center` — выравнивание по центру (значение по умолчанию):

    ```
    <hr size="2" width="200" color="red" align="center">
    ```

  - `left` — выравнивание по левому краю:

    ```
    <hr size="2" width="200" color="red" align="left">
    ```

- `right` — выравнивание по правому краю:

```
<hr size="2" width="200" color="red" align="right">
```

- `noshade` — присутствие этого параметра отменяет рельефность линии:

```
<hr size="2" width="200" align="center" noshade>
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Все рассмотренные параметры тега `<hr>` являются устаревшими и поддерживаются только в формате `Transitional`. Использование их в формате `Strict` недопустимо.

1.5.4. Заголовки

Заголовки могут иметь шесть различных размеров:

```
<hx>Заголовок</hx>
```

где `x` — число от 1 до 6.

Заголовок с номером 1 является самым крупным:

```
<h1>Самый крупный заголовок</h1>
```

Заголовок с номером 6 является самым мелким:

```
<h6>Самый мелкий заголовок</h6>
```

Основным параметром является `align`, он задает выравнивание заголовка относительно окна Web-браузера. Он может принимать следующие значения:

- `center` — выравнивание по центру:

```
<h1 align="center">Заголовок первого уровня с выравниванием по
центру</h1>
```

- `left` — выравнивание по левому краю (по умолчанию):

```
<h2 align="left">Заголовок второго уровня с выравниванием по
левому краю</h2>
```

- `right` — выравнивание по правому краю:

```
<h6 align="right">Самый мелкий заголовок с выравниванием по
правому краю</h6>
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Параметр `align` является устаревшим и поддерживается только в формате `Transitional`. Использование его в формате `Strict` недопустимо.

1.5.5. Разделение на абзацы

Тег `<p>` позволяет разбить текст на отдельные абзацы. Web-браузеры отделяют абзацы друг от друга пустой строкой. Закрывающий тег `</p>` не обязателен.

Основным параметром является `align`, он задает горизонтальное выравнивание. Параметр может принимать следующие значения:

- `center` — выравнивание по центру:

```
<p align="center">Абзац с выравниванием по центру</p>
```

- `left` — выравнивание по левому краю (по умолчанию):

```
<p align="left">Абзац с выравниванием по левому краю</p>
```

- `right` — выравнивание по правому краю:

```
<p align="right">Абзац с выравниванием по правому краю</p>
```

- `justify` — выравнивание по ширине (по двум сторонам):

```
<p align="justify">Абзац с выравниванием по ширине</p>
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Параметр `align` является устаревшим и поддерживается только в формате `Transitional`. Использование его в формате `Strict` недопустимо.

1.6. Списки

Список — это набор упорядоченных абзацев текста, помеченных специальными значками (маркированные списки) или цифрами (нумерованные списки). Рассмотрим каждый из вариантов в отдельности.

1.6.1. Маркированные списки

Маркированный список помещают внутри пары тегов `<ul>` и `</ul>`. Перед каждым нунктом списка необходимо поместить тег `<li>`. Закрывающий тег `</li>` не обязателен. В листинге 1.3 представлена структура маркированного списка.

Листинг 1.3. Маркированный список

```
<ul>
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ul>
```

Тег `<ul>` имеет параметр `type`, позволяющий задать значок, которым помечаются строки списка. Параметр может принимать следующие значения:

- `disc` — значки в форме кружков с заливкой:

```
<ul type="disc">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ul>
```

- `circle` — значки в форме кружков без заливки:

```
<ul type="circle">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ul>
```

- `square` — значки в форме квадрата с заливкой:

```
<ul type="square">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ul>
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Параметр `type` является устаревшим и поддерживается только в формате `Transitional`. Использование его в формате `Strict` недопустимо.

1.6.2. Нумерованные списки

Нумерованный список помещают внутри пары тегов `<ol>` и `</ol>`. Перед каждым нунктом списка необходимо поместить тег `<li>`. Закрывающий тег `</li>` не обязателен.

В листинге 1.4 показана структура нумерованного списка.

Листинг 1.4. Нумерованный список

```
<ol>
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ol>
```

Тег `<ol>` имеет два параметра. Первый из них — `type` — позволяет задать формат, которым нумеруются строки списка.

Параметр может принимать следующие значения:

- `A` — нункты нумеруются прописными латинскими буквами:

```
<ol type="A">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ol>
```

- `a` — нункты нумеруются строчными латинскими буквами:

```
<ol type="a">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ol>
```

- `I` — нункты нумеруются прописными римскими цифрами:

```
<ol type="I">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ol>
```

- `i` — нункты нумеруются строчными римскими цифрами:

```
<ol type="i">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ol>
```

- `1` — нункты нумеруются арабскими цифрами (по умолчанию):

```
<ol type="1">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ol>
```

Второй параметр тега `<ol>` — `start` — задает номер, с которого будет начинаться нумерация строк:

```
<ol type="1" start="5">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ol>
```

Тег `<li>` также имеет параметр `value`, который позволяет изменить номер данного элемента списка:

```
<ol type="1">
  <li>Первый пункт</li>
  <li value="5">Второй пункт</li>
  <li>Третий пункт</li>
</ol>
```

В этом примере "Первый нункт" будет иметь номер 1, "Второй пункт" — номер 5, а "Третий пункт" — номер 6.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Параметры `type`, `start` и `value` являются устаревшими и поддерживаются только в формате `Transitional`. Использование их в формате `Strict` недопустимо.

1.6.3. Списки определений

Списки определений состоят из пар "термин/определение". Описываются с помощью тега `<dl>`. Для вставки термина применяется тег `<dt>`, а для вставки определения — тег `<dd>`. Закрывающие теги `</dt>` и `</dd>` не обязательны. Пример использования списков определений приведен в листинге 1.5.

Листинг 1.5. Списки определений

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Списки определений</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
  <dl>
    <dt>HTML (HyperText Markup Language)</dt>
      <dd>
         Язык разметки документа, описывающий форму отображения
         информации на экране компьютера
      </dd>
    <dt>CSS (Cascading Style Sheets)</dt>
      <dd>Каскадные таблицы стилей</dd>
  </dl>
</body>
</html>
```

1.7. Графика

Применение графики делает Web-страницу визуально привлекательнее. Изображения помогают лучше передать суть и содержание документа. В Интернете применяются графические форматы:

- `GIF` — использует только 256 цветов и поддерживает прозрачность. Кроме того, GIF-файл может содержать анимацию;
- `JPEG` — метод сжатия фотографий с потерей качества. Прозрачность и анимация не поддерживаются;
- `PNG` — формат хранения графики, использующий сжатие без потерь. Поддерживает прозрачность. Разрабатывался в качестве замены формата `GIF`.

> ***Примечание***
>
> Загромождение документа графикой приводит к увеличению времени загрузки Web-страницы. По этой причине применяйте графику только там, где это действительно оправданно.

1.7.1. Изображение на Web-странице

Изображения вставляются в Web-страницы с помощью одинарного тега `<img>`. Сам тег `<img>` должен быть расположен внутри блочного тега, например, `<p>`, `<div>` или др.

Тег имеет следующие параметры:

- `src` — URL-адрес файла графического изображения:

```
<img src="foto.gif" alt="Текст подсказки">
<img src="http://www.mysite.ru/foto.gif" alt="Текст подсказки">
```

- `alt` — строка текста, которая будет выводиться на месте появления изображения до его загрузки или при отключенной графике, а также если изображение загрузить не удалось. Кроме того, при наведении курсора мыши на изображение текст, указанный в параметре `alt`, можно увидеть в качестве текста всплывающей подсказки:

```
<img src="foto.gif" alt="Текст подсказки">
```

- `width` — ширина изображения в пикселах:

```
<img src="foto.gif" width="480" alt="Текст подсказки">
```

- `height` — высота изображения в пикселах:

```
<img src="foto.gif" width="480" height="60" alt="Подсказка">
```

> ***Примечание***
>
> Значения параметров `width` и `height` могут не соответствовать реальным размерам изображения. В этом случае Web-браузер выполнит перемасштабирование. Если значение одного из параметров указать неправильно, то изображение будет искажено. Если указать только один параметр, то значение второго будет рассчитано пропорционально значению первого исходя из реальных размеров изображения.

> ***Совет***
>
> Всегда указывайте значения параметров `width` и `height`, т. к. это позволит Web-браузеру отформатировать Web-страницу до загрузки изображений. В противном случае загрузка каждого изображения приведет к необходимости произвести форматирование еще раз, что в свою очередь приведет к перемещению других элементов Web-страницы. В результате картинка в окне Web-браузера будет дергаться.

Следующие параметры доступны только при использовании формата `Transitional`:

- `border` — толщина границы изображения:

```
<img src="foto.gif" border="0" alt="Текст подсказки">
```

- `align` — расположение изображения относительно текста или других элементов Web-страницы. Параметр может принимать следующие значения:
  - `left` — изображение выравнивается по левому краю, а текст обтекает его с правой стороны:

```
<p><img src="foto.gif" align="left" alt="Подсказка">Текст</p>
```

  - `right` — изображение выравнивается по правому краю, а текст обтекает его с левой стороны:

```
<p><img src="foto.gif" align="right" alt="Подсказка">Текст</p>
```

- `top` — изображение выравнивается по верху текущей строки:

```
<p><img src="foto.gif" align="top" alt="Подсказка">Текст</p>
```

- `bottom` — изображение выравнивается по низу текущей строки:

```
<p><img src="foto.gif" align="bottom" alt="Подсказка">Текст</p>
```

- `middle` — центр изображения выравнивается по базовой линии текущей строки:

```
<p><img src="foto.gif" align="middle" alt="Подсказка">Текст</p>
```

❒ `hspace` — отступ от изображения до текста по горизонтали:

```
<p>
   <img src="foto.gif" align="left" hspace="20"
   alt="Подсказка">Текст
</p>
```

❒ `vspace` — отступ от изображения до текста по вертикали:

```
<p>
   <img src="foto.gif" align="left" vspace="20"
   alt="Подсказка">Текст
</p>
```

1.7.2. Изображение в качестве фона

Параметр `background` тега `<body>` позволяет задать фоновый рисунок для документа:

```
<body background="foto.gif" bgcolor="gray">Тело документа</body>
```

В параметре `bgcolor` следует указывать цвет, близкий к цвету фонового изображения, т. к. резкий переход, например, от светлого тона к темному вызовет неприятное мелькание, ведь фоновое изображение может загрузиться с некоторой задержкой.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Эти параметры являются устаревшими и поддерживаются только в формате `Transitional`. Использование их в формате `Strict` недопустимо.

1.8. Гиперссылки

Гиперссылки позволяют нажатием кнопки мыши быстро перемещаться от одного документа к другому. Именно гиперссылки связывают все Web-страницы в единую сеть.

1.8.1. Внешние гиперссылки

Внешние гиперссылки вставляются в HTML-документ с помощью тега `<a>`. Сам тег `<a>` должен быть расположен внутри блочного тега, например, `<p>`, `<div>` или др.

Основным параметром тега <a> является href. Именно этот параметр задает URL-адрес Web-страницы, которая будет загружена при щелчке мыши на указателе. В качестве указателя может быть текст

```
<a href="http://www.mysite.ru/file.html">Текст ссылки</a>
```

или изображение

```
<a href="http://www.mysite.ru/file.html">
<img src="http://www.mysite.ru/foto.gif" alt="Подсказка"></a>
```

Если URL-адрес содержит символ "&", то его необходимо заменить на HTML-эквивалент &:

```
<a href="index.php?id=5&amp;name=Nik">Текст ссылки</a>
```

> ***Примечание***
>
> Кроме HTML-документов можно ссылаться и на файлы других типов, например, изображения, архивы и т. д. При переходе по такой ссылке Web-браузер в зависимости от типа файла либо отобразит его, либо предложит сохранить.

URL-адреса бывают абсолютными и относительными.

Абсолютный URL-адрес

Абсолютный URL-адрес содержит обозначение протокола, доменный или IP-адрес компьютера, нуть к файлу, а также имя файла. Например:

```
http://www.mysite.ru/folder/file.html
```

Если файл находится в корневой папке, то путь может отсутствовать:

```
http://www.mysite.ru/file.html
```

Имя файла также может отсутствовать. В этом случае загружается Web-страница, заданная по умолчанию в настройках Web-сервера:

```
http://www.mysite.ru/
http://www.mysite.ru/folder/
```

Относительный URL-адрес

При относительном задании URL-адреса путь определяется с учетом местоположения Web-страницы, на которой находится ссылка. Возможны следующие варианты:

- если нужная Web-страница находится в той же папке, что и Web-страница, содержащая ссылку, то URL-адрес может содержать только имя файла. Если с Web-страницы, находящейся по адресу **http://www.mysite.ru/folder1/folder2/file1.html**, нужно перейти на **http://www.mysite.ru/folder1/folder2/file2.html**, то ссылка будет такой:

```
<a href="file2.html">Текст ссылки</a>
```

- если с Web-страницы, находящейся по адресу **http://www.mysite.ru/folder1/folder2/file1.html**, нужно перейти на **http://www.mysite.ru/folder1/folder2/folder3/file2.html**, то ссылку можно указать так:

```
<a href="folder3/file2.html">Текст ссылки</a>
```

- если с Web-страницы, находящейся по адресу **http://www.mysite.ru/folder1/folder2/file1.html**, нужно перейти на **http://www.mysite.ru/folder1/file2.html**, то ссылка будет такой:

```
<a href="../file2.html">Текст ссылки</a>
```

  А при переходе с **http://www.mysite.ru/folder1/folder2/folder3/file1.html** на **http://www.mysite.ru/folder1/file2.html** — такой:

```
<a href="../../file2.html">Текст ссылки</a>
```

Очень часто необходимо загрузить документ в новое окно Web-браузера. Для этого в параметре `target` тега `<a>` следует указать значение `_blank`:

```
<a href="http://www.mysite.ru/file.html" target="_blank">Ссылка</a>
```

Другие значения параметра `target` мы рассмотрим при изучении фреймов *(см. разд. 1.10.6)*.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Использование параметра `target` в формате `Strict` недопустимо.

1.8.2. Внутренние гиперссылки

С помощью внутренних гиперссылок можно создать ссылки на разные разделы текущей Web-страницы. Если документ очень большой, то наличие внутренних гиперссылок позволяет быстро перемещаться между разделами.

Внутренняя гиперссылка также вставляется при помощи тега `<a>` с одним отличием — параметр `href` содержит имя указателя, а не URL-адрес. Перед именем указателя ставится знак #:

```
<a href="#chapter1">Глава 1</a>
```

Указатель создается с помощью тега `<a>`, но вместо параметра `href` используется параметр `name`, который задает имя указателя:

```
<a name="chapter1"></a>
```

Иногда указатель называют "якорем". Также можно сослаться на "якорь" другого документа. Это делается так:

```
<a href="http://www.mysite.ru/file.html#chapter6">Текст</a>
```

Структура документа с внутренними ссылками приведена в листинге 1.6.

Листинг 1.6. Структура документа с внутренними ссылками

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

```
<html>
<head>
 <title>Создание внутренних ссылок</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
 <h1>Название документа</h1>
 <h2>Оглавление</h2>
 <ul>
  <li><a href="#chapter1">Глава 1</a></li>
  <li><a href="#chapter2">Глава 2</a></li>
  <li><a href="#chapter3">Глава 3</a></li>
  <li><a href="#chapter4">Глава 4</a></li>
 </ul>
 <h2><a name="chapter1"></a>Глава 1</h2>
 <p>Содержание главы 1</p>
 <h2><a name="chapter2"></a>Глава 2</h2>
 <p>Содержание главы 2</p>
 <h2><a name="chapter3"></a>Глава 3</h2>
 <p>Содержание главы 3</p>
 <h2><a name="chapter4"></a>Глава 4</h2>
 <p>Содержание главы 4</p>
</body>
</html>
```

1.8.3. Гиперссылки на адрес электронной почты

Ссылка на адрес электронной почты выглядит так:

```
<a href="mailto:mail@mysite.ru">Текст</a>
```

Вместо URL-адреса указывается адрес электронной почты, перед которым добавляется слово "mailto:".

> ***Совет***
>
> Не следует публиковать ссылку с адресом электронной почты на сайте. Такие ссылки автоматически собираются роботами, и в дальнейшем этот E-mail будет завален спамом.

1.9. Таблицы

В HTML-документе таблицы используются в следующих случаях:

- как средство представления данных;
- как элемент оформления страницы, с помощью которого можно точно разместить на странице текст и графику.

Начнем со структуры, описывающей таблицу (листинг 1.7).

Листинг 1.7. Структура HTML-таблиц

```
<table border="1" width="200">
  <caption>Заголовок таблицы</caption>
  <tbody>
    <tr>
      <td align="center">1</td>
      <td align="center">2</td>
    </tr>
    <tr>
      <td align="center">3</td>
      <td align="center">4</td>
    </tr>
  </tbody>
</table>
```

Эта структура описывает таблицу 2×2 с заголовком. Значения в ячейках выровнены по центру. Все ячейки таблицы пронумерованы от 1 до 4.

Таблица вставляется в HTML-документ с помощью парного тега `<table>`. Отдельная ячейка таблицы описывается тегом `<td>`, а ряд ячеек — с помощью тега `<tr>`. Тег `<caption>` позволяет задать заголовок таблицы.

Для логического форматирования таблицы предназначены теги `<thead>` и `<tbody>`. Тег `<thead>` описывает заголовок таблицы, а тег `<tbody>` — основное содержимое таблицы. Закрывающие теги `</thead>` и `</tbody>` не обязательны.

1.9.1. Вставка таблицы в документ

Тег `<table>` имеет следующие параметры:

- `border` управляет отображением линий сетки таблицы, а также задает толщину рамки вокруг таблицы. По умолчанию сетка не отображается:

```
<table><!-- Здесь сетка не отображается -->
<table border="0"><!-- Здесь сетка не отображается -->
<table border="5"><!-- В этом случае сетка отображается, а
толщина рамки вокруг таблицы равна 5 пикселам -->
```

- `cellspacing` задает толщину линий сетки внутри таблицы, точнее сказать, расстояние между рамками соседних ячеек. По умолчанию параметр имеет значение 2. Если параметру присвоить значение 0, то рамки смежных ячеек сольются в одну линию:

```
<table cellspacing="0">
```

- `cellpadding` указывает размер отступа между рамкой ячейки и данными внутри ячейки:

```
<table cellpadding="2">
```

  По умолчанию параметр имеет значение 1;

- `width` определяет ширину таблицы в пикселах или в процентах от размера окна:

```
<table width="200">
<table width="100%">
```

Следующие параметры доступны только при использовании формата `Transitional`:

- `align` задает выравнивание таблицы, а также обтекание таблицы текстом. Он может принимать следующие значения:
  - `left` — таблица выравнивается по левому краю, а текст обтекает ее справа:

```
<table align="left">
```

  - `right` — таблица выравнивается по правому краю, а текст обтекает ее слева:

```
<table align="right">
```

  - `center` — таблица выравнивается по центру:

```
<table align="center">
```

- `bgcolor` указывает цвет фона таблицы:

```
<table bgcolor="silver">
<table bgcolor="#C0C0C0">
```

1.9.2. Заголовок таблицы

Тег `<caption>` позволяет задать заголовок таблицы. Он имеет единственный параметр `align`. Этот параметр может принимать одно из двух значений:

- `top` — заголовок помещается над таблицей:

```
<caption align="top">Заголовок таблицы</caption>
```

- `bottom` — заголовок располагается под таблицей:

```
<caption align="bottom">Заголовок таблицы</caption>
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Параметр `align` является устаревшим и поддерживается только в формате `Transitional`. Использование его в формате `Strict` недопустимо.

1.9.3. Строки таблицы

С помощью парного тега `<tr>` описываются строки таблицы. Он имеет следующие параметры:

- `align` указывает горизонтальное выравнивание текста в ячейках таблицы. Параметр может принимать следующие значения:
  - `left` — по левому краю (по умолчанию):

```
<tr align="left">
```

- `right` — по правому краю:

```
<tr align="right">
```

- `center` — по центру:

```
<tr align="center">
```

- `justify` — по ширине:

```
<tr align="justify">
```

- `valign` определяет вертикальное выравнивание текста в ячейках таблицы. Он может принимать следующие значения:
  - `top` — по верхнему краю:

```
<tr valign="top">
```

  - `middle` — по центру:

```
<tr valign="middle">
```

  - `bottom` — по нижнему краю:

```
<tr valign="bottom">
```

  - `baseline` — по базовой линии:

```
<tr valign="baseline">
```

- `bgcolor` указывает цвет фона ячеек таблицы. Параметр является устаревшим и поддерживается только в формате `Transitional`. Использование его в формате `Strict` недопустимо.

1.9.4. Ячейки таблицы

С помощью тега `<td>` описываются ячейки таблицы. Тег `<td>` имеет следующие параметры:

- `align` и `valign` выполняют те же функции, что и в теге `<tr>`;
- `width` и `height` определяют ширину и высоту ячейки в пикселах или в процентах;
- `bgcolor` указывает цвет фона ячейки;

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Параметры `width`, `height` и `bgcolor` являются устаревшими и поддерживаются только в формате `Transitional`. Использование их в формате `Strict` недопустимо.

- `colspan` задает количество объединяемых ячеек по горизонтали;
- `rowspan` указывает количество объединяемых ячеек по вертикали.

В качестве примера объединения ячеек возьмем наш первоначальный фрагмент кода (листинг 1.7) и объединим горизонтально расположенные ячейки 1 и 2 в одну (листинг 1.8).

Листинг 1.8. Объединение ячеек по горизонтали

```
<table border="1" width="200">
  <caption>Заголовок таблицы</caption>
  <tbody>
    <tr>
      <td align="center" colspan="2">1 и 2 объединены</td>
    </tr>
    <tr>
      <td align="center">3</td>
      <td align="center">4</td>
    </tr>
  </tbody>
</table>
```

Итак, мы заменили строку

```
<td align="center">1</td>
```

на

```
<td align="center" colspan="2">1 и 2 объединены</td>
```

и при этом строка

```
<td align="center">2</td>
```

была удалена.

Теперь объединим вертикально расположенные ячейки 1 и 3 в одну (листинг 1.9).

Листинг 1.9. Объединение ячеек по вертикали

```
<table border="1" width="200">
  <caption>Заголовок таблицы</caption>
  <tbody>
    <tr>
      <td align="center" rowspan="2">1 и 3 объединены</td>
      <td align="center">2</td>
    </tr>
    <tr>
      <td align="center">4</td>
    </tr>
  </tbody>
</table>
```

В этом примере мы заменили строку

```
<td align="center">1</td>
```

на

```
<td align="center" rowspan="2">1 и 3 объединены</td>
```

при этом строка

```
<td align="center">3</td>
```

была удалена.

Тег <th> позволяет указать ячейки, которые являются заголовочными. Содержимое таких ячеек выделяется полужирным шрифтом и размещается по центру. Во всем остальном тег <th> аналогичен тегу <td>. Листинг 1.10 демонстрирует возможность выделения ячеек с помощью тега <th>.

Листинг 1.10. Выделение ячеек таблицы

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Выделение ячеек таблицы</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
  <table border="1" width="500">
    <caption>Заголовок таблицы</caption>
    <thead>
      <tr>
        <th>Марка</th>
        <th>Цвет</th>
        <th>Год выпуска</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>ВАЗ-2109</td>
        <td>Красный</td>
        <td>2008</td>
      </tr>
      <tr>
        <td>Москвич-412</td>
        <td>Белый</td>
        <td>1978</td>
      </tr>
    </tbody>
  </table>
</body>
</html>
```

1.10. Фреймы

Фреймы позволяют разбить окно Web-браузера на несколько прямоугольных областей, в каждую из которых можно загрузить отдельный HTML-документ.

1.10.1. Разделение окна Web-браузера на несколько областей

Обычно заголовок и панель навигации для всех страниц сайта содержат одну и ту же информацию, а изменяется только основное содержание страниц. С помощью фреймовой структуры можно заголовок поместить в одно окно, панель навигации — во второе, а основное содержание страницы — в третье. Это позволит, оставляя в неизменном состоянии два первых окна, изменять содержание третьего.

Попробуем создать Web-страницу с такой структурой. Для этого создадим 5 файлов:

- doc1.html (листинг 1.11) — заголовок Web-страницы;
- doc2.html (листинг 1.12) — панель навигации;
- chapter1.html (листинг 1.13) — содержание главы 1;
- chapter2.html (листинг 1.14) — содержание главы 2;
- test.html (листинг 1.15) — HTML-документ, описывающий фреймовую структуру.

Листинг 1.11. HTML-документ, содержащий заголовок (doc1.html)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Заголовок</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
  <h1>Заголовок</h1>
</body>
</html>
```

Листинг 1.12. HTML-документ, содержащий панель навигации (doc2.html)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Панель навигации</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
```

```
<body>
  <h3>Оглавление</h3>
  <ul>
    <li><a href="chapter1.html" target="chapter">Глава 1</a></li>
    <li><a href="chapter2.html" target="chapter">Глава 2</a></li>
  </ul>
</body>
</html>
```

Листинг 1.13. HTML-документ, в котором находится основное содержание главы 1 (chapter1.html)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Глава 1</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
  <h1>Глава 1</h1>
  <p>Содержание главы 1</p>
</body>
</html>
```

Листинг 1.14. HTML-документ, в котором находится основное содержание главы 2 (chapter2.html)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Глава 2</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
  <h1>Глава 2</h1>
  <p>Содержание главы 2</p>
</body>
</html>
```

Листинг 1.15. HTML-документ, описывающий фреймовую структуру (test.html)

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

```
<head>
 <title>Пример использования фреймов</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<frameset rows="100, *">
  <frame src="doc1.html" scrolling="no">
  <frameset cols="20%, 80%">
    <frame src="doc2.html">
    <frame src="chapter1.html" name="chapter">
  </frameset>
  <noframes>
    <p>Ваш Web-браузер не отображает фреймы</p>
  </noframes>
</frameset>
</html>
```

Все созданные файлы сохраним в одной папке. Затем в Web-браузере откроем файл test.html. Итак, окно разделено на три прямоугольные области. В верхней части окна находится заголовок. В нижней части окна расположены панель навигации (слева) и основная часть документа (справа). При переходе по ссылкам содержимое основной части меняется, а остальные остаются неизменными.

Теперь попробуем изменить цвет фона заголовка. Для этого необходимо отобразить исходный код Web-страницы.

Как нам уже известно, отобразить исходный код обычной Web-страницы можно тремя способами:

- в меню **Вид** выбираем пункт **Просмотр HTML-кода**;
- правой кнопкой мыши щелкаем в любом месте окна Web-браузера. В появившемся контекстном меню выбираем пункт **Просмотр HTML-кода**;
- открываем файл, содержащий исходный код, с помощью Блокнота или другого текстового редактора.

Если документ содержит фреймы, результаты первых двух действий будут разными. Первый способ отобразит только код структуры Web-страницы, а не исходный код каждого из HTML-документов. Иными словами, будет отображен исходный код файла test.html. Второй способ позволяет отобразить код лишь одного HTML-документа, а от места щелчка зависит, код какого HTML-документа будет отображен. В нашем случае для отображения исходного кода файла заголовка (doc1.html) необходимо правой кнопкой мыши щелкнуть внутри области, содержащей заголовок. В появившемся контекстном меню нужно выбрать пункт **Просмотр HTML-кода**.

Заменяем строчку

```
<body>
```

на

```
<body style="background-color:#C0C0C0">
```

сохраняем файл и обновляем Web-страницу. В результате цвет фона заголовка изменится с белого на серый.

Примечание

При использовании фреймов следует учитывать, что поисковые машины при индексации сайтов заносят в свои базы именно отдельные страницы структуры фреймов, а не саму структуру. Это обстоятельство полностью разрушает всю структуру сайта. Ведь если панель навигации расположена на одной странице, а основная часть страницы на другой, то при переходе посетителя с поискового портала он попадает сразу на основную часть, а панель навигации ему не доступна.

1.10.2. Структура HTML-документа, содержащего фреймы

Структура HTML-документа с фреймами (листинг 1.16) отличается от обычной структуры.

Листинг 1.16. Структура HTML-документа с фреймами

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
                "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
 <title>Заголовок страницы</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<frameset rows="100, *">
  <frame>
  <frameset cols="20%, 80%">
    <frame>
    <frame>
  </frameset>
  <noframes>
    <p>Ваш Web-браузер не отображает фреймы</p>
  </noframes>
</frameset>
</html>
```

Как и в обычном HTML-документе, весь код расположен между тегами `<html>` и `</html>`, а в разделе `HEAD` располагаются заголовки. Основное отличие документа с фреймами от обычного HTML-документа — у документа с фреймами отсутствует раздел `BODY`, отсутствует содержимое страницы, а присутствуют только теги, служащие для определения фреймовой структуры. Иными словами, документ с фреймами не может содержать раздела `BODY` и наоборот, обычный HTML-документ не может содержать фреймовую структуру. Кроме того, содержать фреймовую структуру может только документ в формате `Frameset`. Для объявления формата используется заголовок:

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

Вместо тега <body> применяется парный тег <frameset>, описывающий фреймовую структуру. Каждое отдельное окно описывается тегом <frame>. Если Web-браузер не поддерживает фреймы, то в окне будет отображен текст, расположенный между тегами <noframes> и </noframes>. Рассмотрим эти теги подробно.

1.10.3. Описание фреймовой структуры

Парный тег <frameset> описывает фреймовую структуру. Внутри тегов <frameset> и </frameset> могут содержаться только теги <frame> или другой набор фреймов, описанный тегами <frameset> и </frameset>.

Тег <frameset> имеет следующие параметры:

- rows описывает разбиение на строки:

  ```
  <frameset rows="100, *">
  ```

- cols описывает разбиение на столбцы:

  ```
  <frameset cols="20%, 80%">
  ```

  В качестве значений параметров rows и cols указываются размеры фреймов. Должно быть указано как минимум два значения. Все значения в списке разделяются запятыми. Размеры могут быть указаны в абсолютных единицах (в пикселах) или в процентах:

  ```
  cols="20%, 80%"
  ```

  Кроме того, в качестве ширины или высоты может быть указана звездочка (*), которая означает, что под фрейм нужно отвести все оставшееся пространство:

  ```
  rows="100, *"
  ```

1.10.4. Описание отдельных областей

Тег <frame> описывает одиночный фрейм и не имеет закрывающего тега. Он располагается между тегами <frameset> и </frameset> и имеет следующие параметры:

- src определяет URL-адрес документа, который должен быть загружен во фрейм. Может быть указан абсолютный или относительный URL-адрес:

  ```
  <frame src="doc2.html">
  ```

- name задает уникальное имя фрейма:

  ```
  <frame src="chapter1.html" name="chapter">
  ```

- scrolling запрещает или разрешает отображение полос прокрутки во фрейме. Этот параметр может принимать следующие значения:

  - auto — полосы отображаются, только если содержимое не помещается во фрейме (значение по умолчанию):

    ```
    <frame src="chapter1.html" name="chapter" scrolling="auto">
    ```

- `yes` — полосы отображаются в любом случае:

```
<frame src="chapter1.html" name="chapter" scrolling="yes">
```

- `no` — полосы не отображаются в любом случае:

```
<frame src="chapter1.html" name="chapter" scrolling="no">
```

- `marginwidth` задает расстояние в пикселах между границей фрейма и его содержимым по горизонтали:

```
<frame src="chapter1.html" name="chapter" marginwidth="5">
```

- `marginheight` указывает расстояние в пикселах между границей фрейма и его содержимым по вертикали:

```
<frame src="chapter1.html" name="chapter" marginwidth="5"
marginheight="5">
```

- `frameborder` включает или отключает показ границы между фреймами. Может принимать одно из двух значений:
  - `1` — граница отображается (по умолчанию):

```
<frame src="chapter1.html" name="chapter" frameborder="1">
```

  - `0` — граница не отображается:

```
<frame src="chapter1.html" name="chapter" frameborder="0">
```

- `noresize` отключает возможность изменения размеров фрейма пользователем. По умолчанию любой пользователь может изменить размер фрейма путем перемещения границы. Добавляется этот параметр так:

```
<frame src="doc1.html" scrolling="no" noresize>
```

1.10.5. Тег <*noframes*>

Если Web-браузер не поддерживает фреймы, то в окне будет отображен текст, расположенный между тегами `<noframes>` и `</noframes>`. В противном случае содержимое этих тегов будет проигнорировано. Вот пример использования этого тега:

```
<noframes>
  <p>Ваш Web-браузер не отображает фреймы</p>
</noframes>
```

1.10.6. Загрузка документа в определенный фрейм

Для загрузки документа в определенный фрейм существует параметр `target` тега `<a>`. В параметре `target` указывается имя фрейма (которое задается с помощью параметра `name` тега `<frame>`) или одно из зарезервированных значений:

- `_blank` — документ будет загружен в новом окне Web-браузера:

```
<a href="file1.html" target="_blank">Текст ссылки</a>
```

- `_self` — документ будет загружен в тот фрейм, где находится гиперссылка:

```
<a href="file1.html" target="_self">Текст ссылки</a>
```

- `_top` — документ будет загружен поверх всех фреймов:

```
<a href="file1.html" target="_top">Текст ссылки</a>
```

- `_parent` — документ будет загружен в окне, являющемся родительским по отношению к текущему фрейму:

```
<a href="file1.html" target="_parent">Текст ссылки</a>
```

Если нужно загрузить документ во фрейм с именем `chapter`, то ссылка будет такой:

```
<a href="file1.html" target="chapter">Текст ссылки</a>
```

Имя фрейма задается с помощью параметра `name` тега `<frame>`:

```
<frame src="chapter1.html" name="chapter">
```

ОБРАТИТЕ ВНИМАНИЕ

Параметр `target` поддерживается только в формате `Transitional`.

1.10.7. Тег <iframe>. Добавление фрейма в обычный документ

С помощью парного тега `<iframe>` можно вставлять фреймы в обычный HTML-документ. Если тег `<iframe>` не поддерживается, то будет выведен текст между тегами `<iframe>` и `</iframe>`. Иногда такие фреймы называют "плавающими". Тег `<iframe>` имеет следующие параметры:

- `src` определяет URL-адрес документа, который должен быть загружен во фрейм. Может быть указан абсолютный или относительный URL-адрес:

```
<iframe src="http://www.mysite.ru/doc2.html">
<iframe src="doc2.html">
```

- `name` задает уникальное имя фрейма:

```
<iframe src="chapter1.html" name="chapter">
```

- `scrolling` запрещает или разрешает отображение полос прокрутки во фрейме. Может принимать следующие значения:
  - `auto` — полосы отображаются, только если содержимое не помещается во фрейме (значение по умолчанию):

```
<iframe src="chapter1.html" name="chapter" scrolling="auto">
```

  - `yes` — полосы отображаются в любом случае:

```
<iframe src="chapter1.html" name="chapter" scrolling="yes">
```

  - `no` — полосы не отображаются в любом случае:

```
<iframe src="chapter1.html" name="chapter" scrolling="no">
```

- `marginwidth` и `marginheight` определяют расстояние по горизонтали и по вертикали между границей фрейма и его содержимым (в пикселах):

```
<iframe src="chapter1.html" name="chapter" marginwidth="5"
marginheight="5">
```

- `frameborder` включает или отключает показ границ фрейма. Параметр может принимать одно из значений:
  - `1` — граница отображается:

```
<iframe src="chapter1.html" name="chapter" frameborder="1">
```

  - `0` — граница не отображается:

```
<iframe src="chapter1.html" name="chapter" frameborder="0">
```

- `width` и `height` задают ширину и высоту фрейма:

```
<iframe src="chapter1.html" name="chapter" width="200"
height="200">
```

- `align` определяет выравнивание фрейма. Может принимать следующие значения:
  - `left` — фрейм выравнивается по левому краю, текст обтекает фрейм справа:

```
<iframe src="chapter1.html" name="chapter" align="left">
```

  - `right` — фрейм выравнивается по правому краю, текст обтекает фрейм слева:

```
<iframe src="chapter1.html" name="chapter" align="right">
```

  - `top` — вертикальное выравнивание по верхнему краю:

```
<iframe src="chapter1.html" name="chapter" align="top">
```

  - `middle` — вертикальное выравнивание по центру:

```
<iframe src="chapter1.html" name="chapter" align="middle">
```

  - `bottom` — вертикальное выравнивание по нижнему краю:

```
<iframe src="chapter1.html" name="chapter" align="bottom">
```

Попробуем заменить содержимое файла test.html (листинг 1.15) на код, представленный в листинге 1.17.

Листинг 1.17. Применение плавающих фреймов

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Применение плавающих фреймов</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
```

```
<body>
  <h1 align="center">Название документа</h1>
  <iframe src="chapter1.html" name="chapter" align="right" width="700"
     height="400">
     <p>Ваш Web-браузер не отображает фреймы</p>
  </iframe>
  <h2>Оглавление</h2>
  <ul>
    <li><a href="chapter1.html" target="chapter">Глава 1</a></li>
    <li><a href="chapter2.html" target="chapter">Глава 2</a></li>
  </ul>
</body>
</html>
```

Как и в предыдущем примере, заголовок и панель навигации остаются в неизменном состоянии, а при переходе по ссылкам соответствующая страница загружается в окно фрейма.

> ***Обратите внимание***
>
> Тег `<iframe>` поддерживается только в формате `Transitional`. Использование его в формате `Strict` недопустимо.

1.11. Карты-изображения

С помощью карт-изображений можно создать очень красивую панель навигации. В качестве ссылок на разделы сайта будут служить области на обычном изображении формата GIF или JPG.

К минусам такого подхода можно отнести:

- Web-браузеры не могут выделять другим цветом уже пройденные ссылки;
- т. к. вместо текстовых ссылок используются изображения, это приведет к увеличению времени загрузки страницы;
- пользователи могут отключить использование графики и по этой причине не увидят панель навигации.

1.11.1. Карта-изображение как панель навигации

Давайте перепишем файл test.html (мы использовали его при изучении плавающих фреймов) и заменим текстовую панель навигации на карту-изображение (листинг 1.18).

Листинг 1.18. Применение карт-изображений

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

```
<head>
 <title>Применение карт-изображений</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
  <h1 align="center">Название документа</h1>
  <iframe src="chapter1.html" name="chapter" align="right" width="700"
    height="400">
    <p>Ваш Web-браузер не отображает фреймы</p>
  </iframe>
  <img src="foto.gif" alt="Подсказка" width="120" height="240" border="1"
    usemap="#karta">
  <map name="karta">
    <area shape="rect" coords="0,0,120,120" href="chapter1.html"
      target="chapter" alt="Глава 1">
    <area shape="rect" coords="0,120,240,240" href="chapter2.html"
      target="chapter" alt="Глава 2">
    <area shape="default" alt="" nohref>
  </map>
</body>
</html>
```

В данный момент нас не интересует само изображение, поэтому его может и не быть в папке. Чтобы видеть границы изображения на Web-странице, параметру `border` тега `<img>` присвоено значение 1. Сохраним файл и обновим Web-страницу.

Итак, как и в предыдущем примере, есть заголовок и окно фрейма, но вместо текстовой панели навигации имеется изображение 120×240 (в данном примере показана только его рамка). Изображение виртуально разделено пополам на верхнюю и нижнюю области. Если навести курсор мыши на нижнюю часть изображения, то форма курсора даст понять, что это ссылка, а рядом с курсором появится всплывающая подсказка "Глава 2". При переходе по ссылке файл chapter2.html загружается в окно фрейма. Если щелкнуть на верхней части изображения, то во фрейм опять вернется текст "Глава 1".

1.11.2. Структура карт-изображений

Рассмотрим структуру, которая позволяет вставить карту-изображение в HTML-документ (листинг 1.19).

Листинг 1.19. Структура карт-изображений

```
<!-- Часть 1 -->
<img src="foto.gif" alt="Подсказка" width="120" height="240" border="1"
    usemap="#karta">
<!-- Часть 1 -->
<!-- Часть 2 -->
```

```
<map name="karta">
  <area shape="rect" coords="0,0,120,120" href="chapter1.html"
    target="chapter" alt="Глава 1">
  <area shape="rect" coords="0,120,240,240" href="chapter2.html"
    target="chapter" alt="Глава 2">
  <area shape="default" alt="" nohref>
</map>
<!-- Часть 2 -->
```

Как видим, код для вставки карты-изображения состоит из двух частей.

Первая часть с помощью тега `<img>` вставляет изображение в Web-страницу. Параметр `usemap` указывает, что изображение является картой. В качестве значения параметра указывается URL-адрес описания конфигурации. Если описание карты расположено в том же HTML-документе, то указывается название раздела конфигурации карты, перед которым добавляется символ "#".

Вторая часть, являющаяся описанием конфигурации карты, расположена между тегами `<map>` и `</map>`. Активная область карты описывается с помощью тега `<area>`. Сколько на карте активных областей, столько и тегов `<area>` должно быть.

1.11.3. Тег <map>

Парный тег `<map>` служит для описания конфигурации областей карты-изображения. У тега `<map>` есть единственный параметр — `name`. Значение параметра `name` должно соответствовать имени в параметре `usemap` тега `<img>`.

1.11.4. Описание активной области на карте-изображении

Тег `<area>` описывает одну активную область на карте. Закрывающий тег не требуется. Если одна активная область перекрывает другую, то будет реализована первая ссылка из списка областей.

Тег имеет следующие параметры:

- `shape` задает форму активной области. Он может принимать 4 значения:
  - `rect` — активная область в форме прямоугольника (значение по умолчанию):

    ```
    <area shape="rect" alt="Подсказка">
    ```

  - `circle` — активная область в форме круга:

    ```
    <area shape="circle" alt="Подсказка">
    ```

  - `poly` — активная область в форме многоугольника:

    ```
    <area shape="poly" alt="Подсказка">
    ```

  - `default` — активная область занимает всю площадь изображения. Данное значение не поддерживается Internet Explorer:

    ```
    <area shape="default" alt="Подсказка">
    ```

- `coords` определяет координаты точек отдельной активной области. Координаты указываются относительно верхнего левого угла изображения и задаются следующим образом:
  - для области типа `rect` задаются координаты верхнего левого и правого нижнего углов прямоугольника (координаты указываются через запятую):

    ```
    <area shape="rect" coords="x1, y1, x2, y2" alt="Подсказка">
    ```

    Здесь `x1` и `y1` — координаты левого верхнего угла, а `x2` и `y2` — координаты правого нижнего угла, например:

    ```
    <area shape="rect" coords="0, 0, 120, 120" alt="Подсказка">
    ```

  - для области типа `circle` указываются координаты центра круга и радиус:

    ```
    <area shape="circle" coords="x1, y1, r1" alt="Подсказка">
    ```

    Здесь `x1` и `y1` — координаты центра круга, а `r1` — раднус круга, например:

    ```
    <area shape="circle" coords="60, 60, 30" alt="Подсказка">
    ```

  - для области типа `poly` перечисляются координаты вершин многоугольника в нужном порядке:

    ```
    <area shape="poly" coords="x1, y1, x2, y2, x3, y3" alt="">
    ```

    Здесь `x1`, `y1`, `x2`, `y2`, `x3`, `y3` — координаты вершин многоугольника (в данном случае треугольника). Можно задавать и большее количество вершин, иными словами, можно описать активиую область практически любой формы. Координаты последней вершины не обязательно должны совпадать с первой:

    ```
    <area shape="poly" coords="10, 100, 60, 10, 100, 100" alt="">
    ```

- `href` указывает URL-адрес ссылки. Может быть указан абсолютный или относительный адрес ссылки:

```
<area shape="circle" coords="60, 60, 30" alt="Подсказка"
   href="http://www.mysite.ru/chapter1.html">
<area shape="circle" coords="60, 60, 30" alt="Подсказка"
   href="chapter1.html">
```

- `nohref` указывает, что активная область не имеет ссылки. Используется для исключения части другой активной области:

```
<area shape="circle" coords="60, 120, 60" alt="Подсказка" nohref>
<area shape="rect" coords="0, 0, 240, 240" alt="Подсказка"
   href="http://www.mysite.ru/chapter1.html">
```

В данном примере активной областью является вся площадь изображения 120×240 за исключением круга радиусом 60 в центре изображения;

- `alt` задает текст всплывающей подсказки при наведении курсора мыши на активную область:

```
<area shape="rect" coords="0, 0, 240, 240" href="chapter1.html"
alt="Всплывающая подсказка">
```

- `target` указывает, куда будет загружен документ при переходе по ссылке. Может быть указано имя фрейма или одно из зарезервированных значений — `_blank`, `_top`, `_self` или `_parent`:

```
<area shape="rect" coords="0, 0, 240, 240" href="chapter1.html"
target="_blank" alt="Подсказка">
```

Эти значения рассматривались нами при изучении фреймов *(см. разд. 1.10.6)*.

ОБРАТИТЕ ВНИМАНИЕ

Использование параметра `target` в формате `Strict` недопустимо.

1.12. Формы

Формы предназначены для пересылки данных от пользователя к Web-серверу. О том, как обрабатывать эти данные на стороне сервера, будет рассказано при изучении языка PHP. А пока рассмотрим возможности HTML для создания форм.

1.12.1. Создание формы для регистрации сайта

Создадим форму регистрации сайтов в каталоге ресурсов Интернета. Откроем Блокнот и наберем код, приведенный в листинге 1.20.

Листинг 1.20. Пример использования форм

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Пример использования форм</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
 <h1>Пример формы регистрации сайта</h1>
 <form action="file.php" method="POST" enctype="multipart/form-data">
  <div>
   Логин:<br>
   <input type="text" name="pole1"><br>
   Пароль:<br>
   <input type="password" name="pole2" value="Пароль"><br>
   URL-адрес сайта:<br>
   <input type="text" name="pole3" value="http://" size="20"><br>
   Название сайта:<br>
   <input type="text" name="pole4" size="20"><br>
```

```
    Описание сайта:<br>
    <textarea name="pole5" rows="10" cols="15"></textarea><br>
    Тема сайта:<br>
    <select name="pole6">
      <option value="0" selected>Тема не выбрана</option>
      <option value="1">Тема1</option>
      <option value="2">Тема2</option>
      <option value="3">Тема3</option>
    </select><br>
    Баннер 88*31:<br>
    <input type="file" name="pole7" size="20"><br><br>
    <input type="reset" value="Очистить">
    <input type="submit" value="Отправить">
  </div>
 </form>
</body>
</html>
```

Сохраним файл под именем forma.html и откроем его в Web-браузере.

В итоге в окне Web-браузера будет отображена форма, состоящая из пяти текстовых полей (**Логии**, **Пароль**, **URL-адрес сайта**, **Пазваиие сайта** и **Оиисаиие сайта**), списка значений (**Тема сайта**), поля выбора файла с кнопкой **Обзор** (под надписью "Баннер 88*31:"), а также двух кнопок **Очистить** и **Отиравить**.

Кнопка **Очистить** возвращает все значения формы к первоначальным. Кнопка **Отправпть** позволяет отправить данные, введенные пользователем, программе (URL-адрес которой указан в параметре `action` тега `<form>`, в данном случае file.php), расположенной на Web-сервере. Программа обработает данные и либо добавит сайт в каталог и выдаст подтверждение об успешной регистрации, либо выдаст сообщение об ошибке, если обязательное поле не заполнено или заполнено неправильно. В нашем случае программы обработки нет, и отправка данных формы ни к чему не приведет, точнее, приведет к ошибке "Файл не найден".

1.12.2. Структура документа с формами

Форма добавляется в HTML-документ при помощи парного тега `<form>`. Внутри тегов `<form>` и `</form>` могут располагаться теги `<input>`, `<textarea>` и `<select>`, вставляющие в форму элементы управления:

```
<form action="file.php">
<input type="text">
<textarea></textarea>
<select>
<option></option>
</select>
<input type="file">
```

```
<input type="reset">
<input type="submit">
</form>
```

Рассмотрим эти теги подробно.

1.12.3. Добавление формы в документ

Парный тег `<form>` позволяет добавить форму в HTML-документ. Тег имеет следующие параметры:

- `action` задает URL-адрес программы обработки формы. Может задаваться в абсолютной или относительной форме:

```
<form action="file.php">
<form action="http://www.mysite.ru/file.php">
```

- `method` определяет, как будут пересылаться данные из формы Web-серверу. Может принимать два значения — `GET` и `POST`:
  - `GET` — данные формы пересылаются путем их добавления к URL-адресу после знака "?" в формате

```
[Имя параметра]=[Значение параметра]
```

Пары `параметр=значение` отделяются друг от друга символом амперсанда (`&`). Например:

```
http://www.mysite.ru/file.php?pole1=Login&pole2=Password
```

Все специальные символы, а также буквы, отличные от латинских (например, буквы русского языка), кодируются в формате `%nn`, а пробел заменяется знаком "+". Например, фраза "каталог сайтов" будет выглядеть следующим образом:

```
%EA%E0%F2%E0%EB%EE%E3+%F1%E0%E9%F2%EE%E2
```

А если эта фраза является значением поля с именем `pole1`, то строка запроса будет такой:

```
http://www.mysite.ru/file.php?pole1=
%EA%E0%F2%E0%EB%EE%E3+%F1%E0%E9%F2%EE%E2&pole2=Password
```

В теге `<form>` значение `GET` для параметра `method` задается так:

```
<form action="http://www.mysite.ru/file.php" method="GET">
```

Метод `GET` применяется, когда объем пересылаемых данных невелик, т. к. существует предел длины URL-адреса. Длина не может превышать 256 символов;

  - `POST` предназначен для пересылки данных большого объема, файлов и конфиденциальной информации (например, паролей):

```
<form action="http://www.mysite.ru/file.php" method="POST">
```

- enctype задает MIME-тип передаваемых данных. Может принимать два значения:
  - application/x-www-form-urlencoded — применяется по умолчанию:

```
<form action="http://www.mysite.ru/file.php" method="POST"
enctype="application/x-www-form-urlencoded">
```

  - multipart/form-data — указывается при пересылке Web-серверу файлов:

```
<form action="http://www.mysite.ru/file.php" method="POST"
enctype="multipart/form-data">
```

- name задает имя формы:

```
<form action="file.php" name="form1">
```

- target указывает, куда будет помещен документ, являющийся результатом обработки данных формы Web-сервером. Параметр может содержать имя фрейма или одно из зарезервированных значений — _blank, _top, _self или _parent:

```
<form action="http://www.mysite.ru/file.php" method="POST"
enctype="multipart/form-data" target="_blank">
```

Эти значения рассматривались нами при изучении фреймов.

ОБРАТИТЕ ВНИМАНИЕ

Использование параметра target в формате Strict недопустимо.

1.12.4. Описание элементов управления

Тег <input> позволяет вставить в форму элементы управления, например, текстовое поле, кнопку или флажок. Этот тег имеет следующие параметры:

- type задает тип элемента. В зависимости от значения этого поля создаются следующие элементы формы:
  - text — текстовое поле ввода:

```
<input type="text">
```

  - password — текстовое поле для ввода пароля, в котором все вводимые символы заменяются звездочками:

```
<input type="password">
```

  - file — поле ввода имени файла с кнопкой **Обзор**. Позволяет отправить файл на Web-сервер:

```
<input type="file">
```

  - checkbox — поле для установки флажка, который можно установить или сбросить:

```
<input type="checkbox">
```

- `radio` — элемент-переключатель (иногда их называют радиокнопками):

```
<input type="radio">
```

- `reset` — кнопка, при нажатии которой вся форма очищается. Точнее сказать, все элементы формы получают значения по умолчанию:

```
<input type="reset">
```

- `submit` — кнопка, при нажатии которой происходит отправка данных, введенных в форму:

```
<input type="submit">
```

- `button` — обычная командная кнопка:

```
<input type="button">
```

  Такую кнопку имеет смысл использовать только с прикрепленным к ней скриптом. Как это сделать, будет показано в *главе 3*;

- `hidden` — скрытый элемент, значение которого отправляется вместе со всеми данными формы. Элемент не показывается пользователю, но позволяет хранить, например, данные из предыдущей формы (если пользователь последовательно заполпяет несколько форм) или уникальное имя пользователя:

```
<input type="hidden">
```

- `name` задает имя элемента управления. Оно должно обязательно указываться латинскими буквами (например, pole) или комбинацией латинских букв и цифр (например, pole1). Имя элемента не должно начинаться с цифры:

```
<input type="text" name="pole1">
```

- `disabled` запрещает доступ к элементу формы. При наличии параметра элемент отображается серым цветом:

```
<input type="text" name="pole1" disabled>
```

Остальные параметры специфичны для каждого отдельного элемента. Поэтому рассмотрим каждый тип элемента отдельно.

Текстовое поле и поле ввода пароля

Для текстового поля и поля ввода пароля примепяются следующие параметры:

- `value` задает текст поля по умолчанию:

```
<input type="text" name="pole1" value="http://">
```

- `maxlength` указывает максимальное количество символов, которое может быть введено в поле:

```
<input type="text" name="pole1" value="http://" maxlength="100">
```

- `size` определяет видимый размер поля ввода:

```
<input type="text" name="pole1" value="http://" maxlength="100"
size="20">
```

- `readonly` указывает, что поле доступно только для чтения. При наличии параметра значение поля изменить нельзя:

```
<input type="text" name="pole1" readonly>
```

Кнопки *Сброс*, *Отправить* и командная кнопка

Для кнопок используется один параметр:

- `value` задает текст, отображаемый на кнопке:

```
<input type="submit" value="Отправить">
```

Скрытое поле *hidden*

Для скрытого поля указывается один параметр:

- `value` определяет значение скрытого поля:

```
<input type="hidden" name="pole1" value="1">
```

Поле для установки флажка

Для полей-флажков используются следующие параметры:

- `value` задает значение, которое будет передано Web-серверу, если флажок отмечен. Если флажок снят, значение не передается. Если параметр не задан, используется значение по умолчанию — `on`:

```
<input type="checkbox" name="check1" value="yes">Текст
```

- `checked` указывает, что флажок по умолчанию установлен:

```
<input type="checkbox" name="check1" value="yes" checked>Текст
```

Элементы `checkbox` можно объединить в группу. Для этого необходимо установить одинаковое значение параметра `name`. Чтобы получить все значения на сервере, после названия поля следует указать квадратные скобки (это признак массива в языке PHP):

```
<input type="checkbox" name="check[]" value="1">Текст 1
<input type="checkbox" name="check[]" value="2">Текст 2
<input type="checkbox" name="check[]" value="3">Текст 3
```

Элемент-переключатель

При описании элемента-переключателя используются такие параметры:

- `value` указывает значение, которое будет передано Web-серверу, если переключатель выбран:

```
<input type="radio" name="radio1" value="yes">Текст
```

Если ни одно из значений не выбрано, никаких данных передано не будет;

- `checked` обозначает переключатель, выбранный по умолчанию:

```
<input type="radio" name="radio1" value="yes" checked>Текст
```

Элемент-переключатель может существовать только в составе группы подобных элементов, из которых может быть выбран только один. Для объединения переключателей в группу необходимо установить одинаковое значение параметра `name` и разное значение параметра `value`:

```
Укажите ваш пол:<br>
<input type="radio" name="radio1" value="male" checked>Мужской
<input type="radio" name="radio1" value="female">Женский
```

Текстовая область

Парный тег `<textarea>` создает внутри формы поле для ввода многострочного текста:

```
<textarea>
Текст по умолчанию
</textarea>
```

В окне Web-браузера поле отображается в виде прямоугольной области с полосами прокрутки.

Тег имеет следующие параметры:

- `name` — уникальное имя поля:

```
<textarea name="pole2">
Текст по умолчанию
</textarea>
```

- `cols` — число столбцов видимого текста:

```
<textarea name="pole2" cols="15">
Текст по умолчанию
</textarea>
```

- `rows` — число строк видимого текста:

```
<textarea name="pole2" cols="15" rows="10">
Текст по умолчанию
</textarea>
```

Список с предопределенными значениями

Тег `<select>` создает внутри формы список с возможными значениями:

```
<select>
<option>Элемент1</option>
<option>Элемент2</option>
</select>
```

Тег <select> имеет следующие параметры:

- name задает уникальное имя списка:

```
<select name="select1">
```

- size определяет число одновременно видимых элементов списка:

```
<select name="select1" size="3">
```

  По умолчанию size имеет значение 1;

- multiple указывает, что из списка можно выбрать сразу несколько элементов одновременно. Чтобы получить все значения на сервере, после названия списка следует указать квадратные скобки (это признак массива в языке PHP):

```
<select name="select[]" size="3" multiple>
```

Внутри тегов <select> и </select> располагаются теги <option>, с помощью которых описывается каждый элемент списка.

Тег <option> имеет следующие параметры:

- value задает значение, которое будет передано Web-серверу, если пункт списка выбран. Если параметр не задан, посылается текст этого нункта:

```
<select name="select1">
<option value="val1">Элемент1</option>
<option>Элемент2</option>
</select>
```

  Если выбран пункт "Элемент1", то посылается

```
select1="val1"
```

  Если выбран пункт "Элемент2", то посылается

```
select1="Элемент2"
```

- selected указывает, какой нункт списка выбран изначально:

```
<select name="select1">
<option value="val1">Элемент1</option>
<option selected>Элемент2</option>
</select>
```

С помощью тега <optgroup> можно объединить несколько пунктов в группу. Название группы указывается в параметре label:

```
<select name="select1">
  <optgroup label="Отечественные">
    <option value="1">ВАЗ</option>
    <option value="2">ГАЗ</option>
    <option value="3">Москвич</option>
  </optgroup>
  <optgroup label="Зарубежные">
    <option value="4">BMW</option>
```

```
    <option value="5">Opel</option>
    <option value="6">Audi</option>
  </optgroup>
</select>
```

1.12.5. Тег <label>

С помощью тега <label> можно указать пояснительную надпись для элемента формы. Тег имеет два параметра:

- for позволяет указать идентификатор элемента, к которому привязана надпись. Точно такой же идентификатор должен быть указан в параметре id элемента формы:

```
<label for="pass1">Пароль*:</label>
<input type="password" name="pass1" id="pass1">
```

> **ОБРАТИТЕ ВНИМАНИЕ**
>
> Параметр id могут иметь практически все теги. В большинстве случаев он предназначен для доступа к элементу из скрипта.

Если элемент формы разместить виутри тега <label>, то надпись будет связана с элементом и без указания параметра for:

```
<label>Пароль*:
  <input type="password" name="pass1" id="pass1">
</label>
```

- accesskey задает клавишу быстрого доступа. При нажатии этой клавиши одновременно с клавишей <Alt> элемент окажется в фокусе ввода:

```
<label accesskey="N">Пароль*:
  <input type="password" name="pass1" id="pass1">
</label>
```

> **ОБРАТИТЕ ВНИМАНИЕ**
>
> Браузеры Opera и Firefox не поддерживают параметр accesskey.

В качестве примера рассмотрим форму регистрации пользователя (листинг 1.21), а заодно продемонстрируем использование CSS для форматирования страницы.

Листинг 1.21. Пример формы регистрации пользователя

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Пример формы регистрации пользователя</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
```

```
 <style type="text/css">
   body { /* Стиль для всего документа */
     font-size: 10pt; /* Размер шрифта */
     font-family: "Verdana", sans-serif; /* Название шрифта */
   }
   label { /* Стиль для всех элементов label */
     display: inline-block; /* Тип блока */
     width: 150px; /* Ширина */
     vertical-align: top; /* Вертикальное выравнивание */
   }
   select { /* Стиль для всех списков */
     width: 250px; /* Ширина */
     border: 1px solid black; /* Определение стиля для границы */
   }
   input.pole { /* Стиль для элемента input, имеющего класс pole */
     width: 250px; /* Ширина */
     border: 1px solid black; /* Определение стиля для границы */
   }
   textarea { /* Стиль для многострочного текстового поля */
     width: 250px; /* Ширина */
     height: 100px; /* Высота */
     border: 1px solid black; /* Определение стиля для границы */
  }
  form div { /* Стиль для всех div, расположенных внутри form */
     margin-bottom: 20px; /* Отступ блока снизу */
  }
 </style>
</head>
<body>
 <h1>Пример формы регистрации пользователя</h1>
 <form action="reg.php" method="POST" enctype="multipart/form-data">
  <div><label for="login">Логин*:</label>
   <input type="text" name="login" id="login" class="pole"></div>
  <div><label for="pass1">Пароль*:</label>
   <input type="password" name="pass1" id="pass1" class="pole"></div>
  <div><label for="pass2">Повторите пароль*:</label>
   <input type="password" name="pass2" id="pass2" class="pole"></div>
  <div><label for="sex1">Пол*:</label>
   Муж. <input type="radio" name="sex" id="sex1" value="1">
   Жен. <input type="radio" name="sex" id="sex2" value="2"></div>
  <div><label for="education">Образование*:</label>
   <select id="education" name="education">
    <option value=""></option>
    <option value="1">Среднее</option>
    <option value="2">Высшее</option>
   </select></div>
```

```
  <div><label for="comment">Комментарий*:</label>
   <textarea id="comment" name="comment" cols="15" rows="10"></textarea>
  </div>
  <div><label for="userfile">Ваше фото:</label>
   <input type="file" name="userfile" id="userfile"></div>
  <div><label for="rule">С правилами согласен*:</label>
   <input type="checkbox" name="rule" id="rule" value="yes"></div>
  <div>
    <input type="submit" value="Отправить" style="margin-left: 150px;">
  </div>
 </form>
</body>
</html>
```

1.12.6. Группировка элементов формы

Парный тег `<fieldset>` позволяет сгруппировать элементы формы. Web-браузеры вокруг группы отображают рамку. На линии рамки с помощью тега `<legend>` можно разместить надпись. Пример:

```
<fieldset>
   <legend>Пол</legend>
   Муж. <input type="radio" name="sex" value="1">
   Жен. <input type="radio" name="sex" value="2">
</fieldset>
```

1.13. Теги * * и **. Группировка элементов страницы

Теги `<div>` и `<span>` визуально ничего не делают. Зато они позволяют сгруппировать несколько элементов страницы в один (листинг 1.22). Кроме того, тег `<div>` используется для блочной верстки Web-страницы. Если необходимо выделить фрагмент текста внутри абзаца, то следует использовать тег `<span>`, т. к. тег `<div>` отобразит фрагмент на новой строке, а не виутри абзаца.

**Листинг 1.22. Теги и **

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Теги &lt;div&gt; и &lt;span&gt;</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
```

```
   div { /* Стиль для всех тегов DIV */
      font-size: 12pt; /* Размер шрифта */
      color: green; /* Цвет шрифта */
      font-family: "Arial"; /* Название шрифта */
      border: 1px solid black; /* Определение стиля для границы */
      padding: 5px; /* Размер внутренних отступов */
   }
   span { /* Стиль для всех тегов SPAN */
      font-size: 12pt; /* Размер шрифта */
      color: red; /* Цвет шрифта */
      font-family: "Arial"; /* Название шрифта */
      font-weight: bold; /* Жирность шрифта */
   }
 </style>
</head>
<body>
  <div>
    Элемент DIV занимает всю ширину родительского элемента
  </div>
  <p>
    С помощью элемента <span>SPAN</span> можно отформатировать
    <span>фрагмент</span> внутри абзаца
  </p>
</body>
</html>
```

Надо уточнить, что тег `<div>` позволяет не только группировать элементы, но и управлять горизонтальным выравниванием с помощью параметра `align`. Параметр может принимать следующие значения:

- `center` — выравнивание по центру:

```
<div align="center">Элемент с выравниванием по центру</div>
```

- `left` — выравнивание по левому краю:

```
<div align="left">Элемент с выравниванием по левому краю</div>
```

- `right` — выравнивание по правому краю:

```
<div align="right">Элемент с выравниванием по правому краю</div>
```

- `justify` — выравнивание по ширине (по двум сторонам):

```
<div align="justify">Элемент с выравниванием по ширине</div>
```

> ***Обратите внимание***
>
> Параметр `align` является устаревшим и поддерживается только в формате `Transitional`. Использование его в формате `Strict` недопустимо.

1.14. Отличия XHTML 1.0 от HTML 4.01

XHTML (*eXtensible HyperText Markup Language*, расширяемый язык разметки гипертекста) — это язык разметки Web-страниц, созданный на базе XML. XHTML 1.0 базируется на HTML 4.01, но приведен в соответствие с правилами XML 1.0. Обрабатывая страницу на языке HTML 4.01, Web-браузер, обнаружив ошибку, может попробовать обработать ее. XHTML 1.0 имеет более строгие правила. Web-браузер, обнаружив ошибку, должен прекратить дальнейшую обработку страницы.

Допустимые форматы для XHTML 1.0:

- `Strict` — строгий формат. Не содержит тегов и параметров, помеченных как устаревшие или не одобряемые. Объявление формата:

```
<!DOCTYPE html PUBLIC  "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

- `Transitional` — переходный формат. Содержит устаревшие теги в целях совместимости и упрощения перехода со старых версий HTML. Объявление формата:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

- `Frameset` — аналогичен переходному формату, но содержит также теги для создания фреймов. Объявление формата:

```
<!DOCTYPE html PUBLIC  "-//W3C//DTD XHTML 1.0 Frameset//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

Кроме объявления формата, XHTML требует указания пространства имен в корневом теге в параметре `xmlns`:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ru" lang="ru">
```

В этом примере с помощью параметра `lang` мы также указали язык документа (`lang="ru"`).

Стандарт рекомендует также указывать XML-пролог

```
<?xml version="1.0" encoding="windows-1251"?>
```

перед объявлением формата, а в заголовках ответа сервера отправлять MIME-тип `application/xhtml+xml`. Однако Web-браузер Internet Explorer в этом случае некорректно обрабатывает документ. По этой причине XML-пролог не указывают, а в заголовках ответа сервера задают MIME-тип `text/html`.

При использовании языка XHTML необходимо придерживаться следующих правил.

- Все названия тегов и параметров указываются в нижнем регистре.

  Неправильно:

```
<P>Текст абзаца</P>
```

Правильно:

```
<p>Текст абзаца</p>
```

- Значения параметров следует обязательно указывать внутри кавычек.

  Неправильно:

```
<textarea name="pole" cols=15 rows=10></textarea>
```

  Правильно:

```
<textarea name="pole" cols="15" rows="10"></textarea>
```

- Если параметр не имеет значения (например, `selected`), то в качестве значения указывается название параметра.

  Неправильно:

```
<select name="select1">
<option value="1">Элемент1</option>
<option value="2" selected>Элемент2</option>
</select>
```

  Правильно:

```
<select name="select1">
<option value="1">Элемент1</option>
<option value="2" selected="selected">Элемент2</option>
</select>
```

- Необходимо строго соблюдать правильность вложенности тегов.

  Неправильно:

```
<p><b>Текст</p></b>
```

  Правильно:

```
<p><b>Текст</b></p>
```

- Нельзя вкладывать блочный тег (например, `<p>`, `<div>`) во впутристрочный (например, `<b>`, `<span>`).

  Неправильно:

```
<b><p>Текст</p></b>
```

  Правильно:

```
<p><b>Текст</b></p>
```

- Все теги должны быть закрыты.

  Неправильно:

```
<ol>
  <li>Первый пункт
  <li>Второй пункт
</ol>
```

Правильно:

```
<ol>
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ol>
```

- Для одинарных тегов (например, `<br>`, `<img>` и др.) перед закрывающей угловой скобкой необходимо указать пробел и слэш (" /").

  Неправильно:

```
<input type="text" name="pole1">
<img src="foto.gif" alt="Текст подсказки">
<br>
```

  Правильно:

```
<input type="text" name="pole1" />
<img src="foto.gif" alt="Текст подсказки" />
<br />
```

- Все специальные символы внутри тегов должны быть заменены на HTML-эквиваленты (например, символ "<" необходимо заменить на `<`).

  Неправильно:

```
<p> i < 0 </p>
```

  Правильно:

```
<p> i &lt; 0 </p>
```

  Если специальные символы невозможно заменить на HTML-эквиваленты, то их следует разместить внутри комментария

```
<script type="text/javascript">
<!--
   var i = -10; if (i < 0) alert("Переменная i меньше нуля");
//-->
</script>
```

  или впутри блока `CDATA`

```
<script type="text/javascript">
//<![CDATA[
   var i = -10; if (i < 0) alert("Переменная i меньше нуля");
//]]>
</script>
```

Пример использования языка XHTML приведен в листинге 1.23.

Листинг 1.23. Пример использования языка XHTML

```
<!DOCTYPE html PUBLIC  "-//W3C//DTD XHTML 1.0 Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ru" lang="ru">
<head>
 <title>Пример использования языка XHTML</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251" />
</head>
<body>
<p>
   <img src="foto.gif" alt="Альтернативный текст" />
   Символ амперсанд внутри ссылки должен заменяться на HTML-
   эквивалент.<br />
   <a href="index.php?id=5&amp;name=Nik">Текст ссылки</a>
</p>
<form action="index.php">
   <div>
      <input type="text" name="txt1" />
      <input type="submit" value="Отправить" />
   </div>
</form>
</body>
</html>
```

1.15. Проверка HTML-документов на соответствие стандартам

После создания HTML-документа его необходимо проверить на отсутствие ошибок и соответствие стандартам. Ведь можно случайно забыть закрыть тег, допустить опечатку в названии тега или параметра, нарушить правильность вложенности тегов и др. Web-браузеры обычно не сообщают об ошибках, а пытаются их обработать. Поэтому о существовании ошибок можно узнать только в случае, если Web-браузер неправильно их обработал и это визуально видно.

Для проверки (X)HTML-документов предназначен сайт **http://validator.w3.org/**. Чтобы проверить документ, размещенный в Интернете, достаточно ввести URL-адрес и нажать кнопку **Check**. Можно также загрузить файл или вставить HTML-код в поле ввода многострочного текста. Если после проверки были обнаружены ошибки, то будет выведено их подробное описание. После исправления ошибок следует повторно проверить HTML-документ.

1.16. Специальный тег в Web-браузере Internet Explorer

Начиная с пятой версии, Internet Explorer поддерживает специальный тег, позволяющий учитывать особенности разных версий этого Web-браузера. Формат тега:

```
<!--[if Условие IE Версия]>Какой-то текст<![endif]-->
```

Как не трудно заметить, все содержимое тега расположено внутри HTML-комментария. По этой причине тег не нарушает стандарты консорциума W3C и может использоваться в строгом режиме.

В необязательном параметре `Условие` могут быть указаны следующие операторы:

- `lt` — меньше чем;
- `lte` — меньше или равно;
- `gt` — больше чем;
- `gte` — больше или равно;
- `!` — не равно.

В необязательном параметре `Версия` указывается версия Web-браузера Internet Explorer. Пример использования тега приведен в листинге 1.24.

Листинг 1.24. Специальный тег в Web-браузере Internet Explorer

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Специальный тег в Web-браузере Internet Explorer</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
 <p>
  <!--[if IE]>Это Web-браузер Internet Explorer<br><![endif]-->
  <!--[if IE 5.5]>версия 5.5<br><![endif]-->
  <!--[if IE 6]>версия 6<br><![endif]-->
  <!--[if IE 7]>версия 7<br><![endif]-->
  <!--[if IE 8]>версия 8<br><![endif]-->
  <!--[if gte IE 6]>версия больше или равна 6<br><![endif]-->
  <!--[if lt IE 6]>версия меньше 6<br><![endif]-->
  <!--[if lte IE 6]>версия меньше или равна 6<br><![endif]-->
  <!--[if gt IE 6]>версия больше 6<br><![endif]-->
  <!--[if ! IE 7]>Инструкция для всех версий, кроме 7-й<br><![endif]-->
  <!--[if ! IE]> <-->
  Если вы видите эту надпись, то не используете Internet Explorer
  <!--> <![endif]-->
 </p>
</body>
</html>
```

Результат выполнения листинга 1.24 в Internet Explorer 7.0 будет выглядеть так:

```
Это Web-браузер Internet Explorer
версия 7
версия больше или равна 6
версия больше 6
```

Как видно из примера, содержимое тега выводится только в том случае, если соблюдается условие.

1.17. HTML 5

Язык HTML продолжает развиваться. В настоящее время в разработке находится его пятая версия — *HTML 5*. Она уже является рабочим стандартом W3C, и все современные Web-обозреватели поддерживают большинство представленных в ней нововведений.

Давайте перечислим версии Web-обозревателей, в которых появилась поддержка HTML 5.

- Microsoft Internet Explorer — 9;
- Mozilla Firefox — 4.0;
- Google Chrome — 6.0;
- Opera — 11.1;
- Apple Safari — 5.0.

Если далее мы не укажем список Web-обозревателей и их версий, в которых появилась поддержка какой-либо из описываемых возможностей HTML 5, значит, она поддерживается версиями, перечисленными выше, и более поздними. В противном случае мы явно упомянем, в каких версиях каких программ данная возможность поддерживается или, наоборот, не поддерживается.

1.17.1. Требования к страницам, написанным на HTML 5

Сначала поговорим о требованиях, предъявляемых к Web-страницам, которые написаны на языке HTML 5. Их немного.

- Тег `<!doctype>`, указывающий формат страницы, должен выглядеть следующим образом:

```
<!doctype html>
```

- Для написания страниц настоятельно рекомендуется кодировка UTF-8. Применение прочих кодировок нежелательно.

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

- Наименования тегов и их параметров рекомендуется набирать символами нижнего регистра: `<p>`, `<table>`, `<html>`, `<div class="special">` и пр. (хотя не возбраняется применять для этого символы верхнего регистра, как в случае HTML 4.01). Именно в нижнем регистре мы и будем записывать их в дальнейшем.
- Шаблон страницы, написанной на HTML 5, иллюстрирует листинг 1.25.

Листинг 1.25. Шаблон страницы, написанной на HTML 5

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    . . .
  </head>
  <body>
    . . .
  </body>
</html>
```

1.17.2. Семантическая разметка

Одно из ключевых нововведений HTML 5 — средства семантической разметки страниц. Под *семантической разметкой* подразумевается разделение содержимого страницы на части, каждая из которых имеет особое значение: заголовок страницы, панель навигации, основное содержимое, часть основного содержимого, "поддон" и др.

Для семантической разметки предусмотрены особые теги. Все они ведут себя как обычные блоки (теги `<div>`).

Кроссплатформенные теги семантической разметки:

- `<article>` — независимый и самодостаточный фрагмент основного содержимого страницы: отдельная статья, запись форума, блога или комментарий.
- `<aside>` — примечание к статье, обычно располагающееся сбоку от основного текста.
- `<figcaption>` — подпись к иллюстрации. Может присутствовать только в теге `<figure>`.
- `<figure>` — иллюстрация к статье: обычное графическое изображение, аудио-, видеоролик или даже фрагмент текста.
- `<footer>` — "поддон" статьи или самой страницы.
- `<header>` — "шапка" статьи или самой страницы.
- `<mark>` — важный фрагмент статьи или самой страницы.
- `<nav>` — набор гиперссылок для навигации по содержимому статьи или самому сайту (панель навигации).
- `<section>` — значимая часть материала, например, параграф большой статьи. Также применяется в разметке страницы *(см. разд. 2.12)* для создания блока с основным содержимым.
- `<time>` — число и время. Может включать параметр `datetime`, указывающий дату и время в "машинном" представлении, для чего используется формат *<год>-<месяц>-<число> <часы>:<минуты>*.

Пример использования семантической разметки иллюстрирует листинг 1.26.

На данный момент Web-обозреватели никак не выделяют визуально теги семантической разметки. Однако мы всегда сможем привязать к ним стили, чтобы задать нужное нам оформление (о стилях будет рассказано в *главе 2*).

Листинг 1.26. Пример страницы, отформатированной с применением семантической разметки

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Семантическая разметка</title>
  </head>
  <body>
    <header>
      <h1>Семантическая разметка</h1>
    </header>
    <nav>
      <p><a href="page2.html">Страница 2</a></p>
      <p><a href="page3.html">Страница 3</a></p>
    </nav>
    <section>
      <p>Основное содержимое</p>
      <article>
        <h2>Заголовок статьи</h2>
        <p>Текст статьи</p>
        <mark>
          <p>Важный фрагмент статьи</p>
        </mark>
        <figure>
          <p><img src="picture.jpg"></p>
          <figcaption>Иллюстрация</figcaption>
        </figure>
        <aside>
          <p>Примечание к статье</p>
        </aside>
      <article>
    </section>
    <footer>
      <p>Страница была сделана в <time datetime="2014-01-01 00:00">Новый
      год</time></p>
      <p>Все права защищены</p>
    </footer>
  </body>
</html>
```

Что касается перечисленных далее некроссплатформенных тегов, то их не очень много, и полезность их довольно сомнительна.

- `<main>` — блок основного содержимого страницы. Не поддерживается Internet Explorer.
- `<detail>` — дополнительные сведения о статье или странице: имя автора, сведения об авторских правах и т. п.
- `<summary>` — заголовок для блока дополнительных сведений (тега `<detail>`). Может присутствовать только в этом теге.

  Оба этих тега поддерживаются Chrome версии 12.0 и выше, Opera версии 15.0 и выше и Safari версии 6.0 и выше.

1.17.3. Мультимедиа

Другое ключевое нововведение HTML 5 — поддержка вставки на страницы аудио- и видеороликов без применения сторонних программ.

К сожалению, разные Web-обозреватели поддерживают различные форматы аудио и видео. Поэтому, если мы хотим поместить на страницу аудиоролик, придется подготовить его в двух вариантах:

- для Internet Explorer и Safari — формат файла MPEG 3 level 3 (MP3) и такой же формат кодирования;
- для Firefox и Opera — формат файла OGG и формат кодирования Vorbis.

Chrome поддерживает оба этих варианта.

Если же мы хотим поместить на страницу видеоролик, то также потребуются два варианта:

- для Internet Explorer и Safari — формат файла MPEG 4 File (MP4), формат кодирования звука AAC LC и формат кодирования видео MPEG 4 AVC (H.264);
- для Firefox, Opera и Chrome — формат файла WebM, формат кодирования звука Vorbis и формат кодирования видео VP8.

Для перекодирования аудио- или видеофайла в нужный формат подойдет любая программа-перекодировщик, например SUPER (**http://www.erightsoft.com/SUPER.html**).

Вставка аудиоролика

Аудиоролик на страницу вставляется с помощью тега `<audio>`:

```
<audio src="sound.mp3"></audio>
```

Этот тег поддерживает следующие параметры:

- `src` — интернет-адрес мультимедийного файла с роликом;
- `autoplay` — если указан, воспроизведение ролика начнется сразу после загрузки страницы;

- `controls` — если указан, на экране будут присутствовать элементы для управления воспроизведением ролика (шкала прогресса с регулятором позиции воспроизведения, регулятор громкости, кнопки пуска и паузы и кнопка приглушения звука);
- `loop` — если указан, ролик будет воспроизводиться бесконечно ("зациклится");
- `muted` — если указан, после загрузки страницы звук будет приглушен;
- `preload` — позволяет указать, как будет выполнена предварительная загрузка файла с роликом. Доступны три значения: `none` (не выполнять предзагрузку), `metadata` (загрузить только самое начало файла, где хранятся сведения о ролике, в частности, его продолжительность) и `auto` (загрузить весь файл; поведение по умолчанию). Этот параметр имеет смысл указывать лишь в том случае, если автоматическое воспроизведение не активизировано (параметр `autoplay` отсутствует в теге).

Пример:

```
<audio src="sound.mp3" controls preload="metadata"></audio>
```

Вставка видеоролика

Для вставки на страницу видеоролика применяется тег `<video>`:

```
<video src="video.webm"></video>
```

Этот тег поддерживает все параметры, уже знакомые нам по тегу `<audio>`, и три дополнительных:

- `width` — ширина панели проигрывателя (в пикселах), в которой будет выводиться видеоролик. Если не указан, ширина панели будет равна ширине видео, записанного в файле;
- `height` — высота панели проигрывателя (в пикселах), в которой будет выводиться видеоролик. Если не указан, высота панели будет равна высоте видео, записанного в файле;
- `poster` — интернет-адрес графического файла, содержимое которого будет выведено в панели проигрывателя после загрузки ролика. Если не указан, перед началом воспроизведения в панели проигрывателя будет выведен первый кадр ролика.

Пример:

```
<video src="video.webm" controls wideo="640" height="480"
poster="poster_image.jpg"></video>
```

Указание нескольких источников аудио или видео

Ранее говорилось, что разные Web-обозреватели поддерживают различные форматы аудио- и видеофайлов, и каждый из помещаемых на страницу файлов нам приходится готовить в двух вариантах. В связи с этим возникает вопрос: можно ли ука-

зать в теге <audio> или <video> оба этих варианта, чтобы Web-обозреватель загрузил тот, который он поддерживает?

Можно. Требуется лишь удалить из тега <audio> или <video> параметр src и поместить в сам этот тег набор тегов <source>, каждый из которых укажет один из вариантов помещаемого на страницу файла.

Пример:

```
<video controls>
  <source src="video.mp4">
  <source src="video.webm">
</video>
```

Здесь мы указываем в теге <video> два файла (в форматах MP4 и WebM), предназначенные для разных программ.

Тег <source> поддерживает следующие параметры:

- src — собственно интернет-адрес мультимедийного файла;
- type — обозначение формата мультимедийного файла в виде типа MIME. MIME-типы для всех четырех упомянутых ранее форматов файлов перечислены в табл. 1.1. Если этот параметр не указан, Web-обозреватель сам определит формат файла по его расширению.

Таблица 1.1. *MIME-типы мультимедийных файлов*

Формат файла	MIME-тип
MP3	audio/mpeg
OGG	audio/ogg
MP4	video/mp4
WebM	video/webm

Обеспечение совместимости со старыми Web-обозревателями

Старые Web-обозреватели, не поддерживающие теги <audio> и <video>, не смогут воспроизвести аудио- или видеоролик. В этом случае будет уместным вывести на страницу какой-либо текст, сообщающий об этом.

Текст сообщения можно поместить прямо в тег, выводящий мультимедийный файл. Web-обозреватели, поддерживающие аудио и видео в стиле HTML 5, его проигнорируют, а старые программы благополучно выведут на экран.

Пример:

```
<audio src="sound.mp3">
  <div>Ваш Web-обозреватель не поддерживает аудио в стиле HTML 5.</div>
</audio>
```

1.17.4. Новые возможности форм и элементов управления

Еще HTML 5 предоставляет Web-дизайнерам дополнительные возможности при работе с формами, в частности, новые элементы управления.

Новые возможности форм

Тег `<form>`, создающий форму, получил поддержку двух новых параметров:

- `autocomplete` — включает или отключает автодополнение значений, занесенных в поля ввода формы. Значение `on` включает автодополнение, `off` — отключает. Web-обозреватель Opera поддерживает этот параметр, начиная с версии 12;
- `novalidate` — если присутствует, то значения, занесенные в поля ввода, не будут проверяться. (Подробнее об этом будет рассказано далее.) Internet Explorer поддерживает этот параметр, начиная с версии 10; Safari его не поддерживает.

Пример:

```
<form action="process_data.php" autocomplete="off" novalidate> . . .
</form>
```

Новые возможности элементов управления

Прежде всего, HTML 5 позволяет создавать новые элементы управления. Это можно сделать, указав соответствующее значение параметра `type` в теге `<input>`. Перечень новых значений этого параметра и элементов управления, создаваемых с их помощью:

- `number` — поле для ввода числового значения;
- `range` — регулятор для указания числового значения;
- `email` — поле ввода адреса электронной почты; не поддерживается Safari;
- `url` — поле ввода интернет-адреса; не поддерживается Safari;
- `color` — инструмент для выбора цвета; поддерживается лишь Firefox, Chrome и Opera;
- `data` — поле ввода даты; поддерживается лишь Chrome, Opera и Safari;
- `datetime` — поле ввода даты, времени и часового пояса; поддерживается лишь Opera и Safari;
- `datetime-local` — поле ввода даты и времени; поддерживается лишь Opera и Safari;
- `month` — поле ввода месяца и года; поддерживается лишь Chrome, Opera и Safari;
- `search` — поле ввода подстроки для поиска; поддерживается лишь Chrome и Safari;
- `time` — поле ввода времени; поддерживается лишь Chrome, Opera и Safari;

- `week` — поле ввода номера недели и года; поддерживается лишь Chrome, Opera и Safari.

Элемент управления подобного рода сначала проверяет, является ли занесенное в него значение корректным числом, адресом электронной почты, интернет-адресом, датой, временем и т. д., и только в случае успешного выполнения этой проверки позволяет форме отправить данные на сервер. Если проверка не увенчается успехом, данные отправлены не будут, а ниже элемента управления появится сообщение о некорректном вводе данных.

Мы можем отключить эту проверку, указав в теге `<form>` формы параметр `novalidate`. Однако делать это следует лишь в особых случаях.

Также тег `<input>` поддерживает новые параметры:

- `autocomplete` — аналогичен одноименному параметру тега `<form>`;
- `autofocus` — если указан, данный элемент управления получит фокус ввода сразу после загрузки страницы; Internet Explorer поддерживает этот параметр, начиная с версии 10;
- `max` — максимальное число или значение даты, которое можно ввести в поле или задать с помощью регулятора; Internet Explorer поддерживает этот параметр, начиная с версии 10;
- `min` — минимальное число или значение даты, которое можно ввести в поле или задать с помощью регулятора; Internet Explorer поддерживает этот параметр, начиная с версии 10;
- `multiple` — если указан в поле ввода файла или адреса электронной почты, туда можно занести сразу несколько значений; Internet Explorer поддерживает этот параметр, начиная с версии 10;
- `pattern` — формат вводимого значения в виде регулярного выражения JavaScript (о регулярных выражениях будет рассказано в *разд. 3.15.10*); Internet Explorer поддерживает этот параметр, начиная с версии 10, а Safari вообще не поддерживает;
- `placeholder` — текст подсказки, который будет выводиться прямо в поле ввода; Internet Explorer поддерживает этот параметр, начиная с версии 10;
- `required` — если указан, в это поле ввода обязательно должно быть занесено значение; Internet Explorer поддерживает этот параметр, начиная с версии 10;
- `step` — интервал между двумя допустимыми значениями. Этот параметр применим к полям ввода числа, даты, времени, даты и времени, месяца и недели, а также к регуляторам. Internet Explorer поддерживает его, начиная с версии 10.

Если посетитель не ввел значение в элемент управления, помеченный как обязательный к заполнению, или если это значение не соответствует заданным параметрам (не укладывается в указанный диапазон или не совпадает с регулярным выражением), ниже элемента появится соответствующее предупреждение. Понятно, что данные на сервер в этом случае отправлены не будут.

Примеры:

```
<input type="email" name="email" required placeholder="Ваш e-mail">
<input type="range" name="level" min="1" max="10" step="2">
<input type="date" name="birthday" autofocus>
<input type="text" name="post_code" pattern="[0-9]{6}"
placeholder="Код почтового отделения (шесть цифр)">
```

Тег `<textarea>` поддерживает уже знакомые нам параметры `autofocus`, `placeholder` и `required`; также введена поддержка параметра `maxlength`, известного нам по обычным полям ввода. Новый параметр `wrap` управляет вставкой символов переноса строк в текст, отправляемый на сервер: значение `soft` отключает их вставку (поведение по умолчанию), а значение `hard` включает (в этом случае требуется указать параметр `cols`).

Тег `<select>` теперь поддерживает параметры `autofocus` и `required`.

Задание значений для автодополнения

Автодополнение значений, заносимых в поле ввода, — очень полезная вещь. (Впрочем, не всегда, и именно поэтому HTML 5 позволяет нам отключить его, указав для параметра `autocomplete` тега `<form>` значение `off`.) Но мы можем сделать его еще лучше, задав для какого-либо поля ввода несколько предопределенных значений, которые будут подставлены в список автодополнения.

Этот список создается с помощью парного тега `<datalist>`. Для него мы в обязательном порядке с помощью параметра `id` укажем идентификатор.

Позиции для списка автодополнения создаются с помощью тегов `<option>`, знакомых нам по обычным спискам. Эти теги помещаются внутрь тега `<datalist>`.

И напоследок мы привяжем созданный список к полю ввода, задав в его теге идентификатор списка в качестве значения параметра `list` (листинг 1.27).

Листинг 1.27. Пример автозаполнения

```
<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
<input type="text" name="browser" list="browsers"
placeholder="Ваш Web-обозреватель">
```

Internet Explorer поддерживает тег `<datalist>`, начиная с версии 10, Chrome — с версии 20, а Safari не поддерживает совсем.

1.17.5. Прочие нововведения

Осталось рассмотреть еще несколько нововведений, появившихся в HTML 5. Их совсем немного.

- Появилась поддержка графического формата *SVG*. Это векторный формат, описывающий изображение как набор примитивов: прямых, кривых, многоугольников и др. Векторные изображения масштабируются без потери качества, а формат SVG, кроме того, поддерживает создание анимированных изображений и даже интерактивной графики, откликающейся на действия посетителя. Изображения SVG вставляют на страницу с помощью тега `<img>`.
- Появилась поддержка программного рисования произвольной графики посредством особого элемента, называемого канвой. Подробнее о канве и рисовании на ней будет рассказано в *разд. 3.21*.

1.17.6. Теги и параметры, объявленные устаревшими

Ряд тегов и их параметров, поддерживавшихся в предыдущих версиях HTML, в HTML 5 объявлены устаревшими и не рекомендованными к использованию. Нам обязательно нужно их рассмотреть.

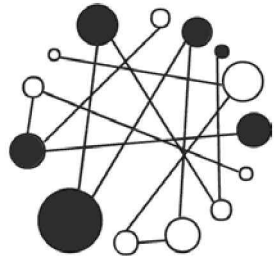
Начнем с устаревших тегов. Вот их список, вместе с рекомендациями по их замене другими тегами и стилями CSS (о них будет рассказано в *главе 2*):

- `<acronym>` — его следует заменять тегом `<abbr>`;
- `<frame>`, `<frameset>`, `<noframes>` — либо использовать `<iframe>`, либо создавать цельные страницы без фреймов;
- `<strike>` — заменять на тег `<del>`;
- `<big>`, `<font>`, `<small>`, `<tt>` — использовать форматирование средствами CSS или другие подходящие теги.

Теперь займемся устаревшими параметрами тегов. Здесь также приводятся рекомендации по замене такого параметра другим, допустимым в HTML 5, или применении другого приема, обеспечивающего аналогичный результат.

- `name` в тегах `<a>`, `<img>` и `<option>` — использовать параметр `id`;
- `nohref` в теге `<area>` — исключить параметр `href`;
- `lowsrc` в теге `<img>` — использовать изображение JPEG с прогрессивной разверткой;
- `align`, `alink`, `background`, `bgcolor`, `border`, `bordercolor`, `cellpadding`, `cellspacing`, `clear`, `color`, `compact`, `frameborder`, `hspace`, `link`, `marginbottom`, `marginheight`, `marginleft`, `marginright`, `margintop`, `marginwidth`, `noshade`, `nowrap`, `scrolling`, `text`, `valign`, `vlink` и `vspace`, `size` в теге `<hr>`, `type` в тегах `<li>`, `<ol>` и `<ul>`, `width` в тегах `<hr>`, `<pre>`, `<table>`, `<td>`, `<th>` — применять форматирование средствами CSS.

ГЛАВА 2



Основы CSS. Форматируем Web-страницу с помощью стилей

2.1. Основные понятия

Каскадные таблицы стилей *(CSS — Cascading Style Sheets)* позволяют существенно расширить возможности языка HTML за счет более гибкого управления форматированием Web-страницы.

Для примера возьмем параметр `size` тега `<font>`:

```
<font size="3">Текст</font>
```

В языке HTML размер шрифта указывается в условных единицах от 1 до 7. Размер, используемый Web-браузером по умолчанию, принято приравнивать к 3. А какой размер шрифта в пунктах используется по умолчанию? В разных Web-браузерах по-разному... Это означает, что наша Web-страница также будет выглядеть по-разному...

С помощью CSS можно точно задать размер шрифта в пунктах. Делается это с помощью параметра `style`:

```
<font style="font-size: 12pt">Текст</font>
```

Применение стилей позволяет задавать точные характеристики практически всех элементов Web-страницы, а это значит, что можно точно контролировать внешний вид Web-страницы в окне Web-браузера.

Прежде чем приступить к изучению CSS, разберемся с основными понятиями.

Значение параметра `style` (`font-size: 12pt`) называется *определением стиля* или *стилем*. Элемент определения стиля (`font-size`) называется *атрибутом*. Каждый атрибут имеет *значение* (`12pt`), указываемое после двоеточия. Совокупность определений стилей, вынесенных в заголовок HTML-документа или в отдельный файл, называют *таблицей стилей*.

2.2. Способы встраивания определения стиля

Задать стиль можно тремя способами: встроить определение стиля в тег, встроить определения стилей в заголовок HTML-документа или вынести таблицу стилей в отдельный файл.

2.2.1. Встраивание определения стиля в тег

Определение стиля встраивается в любой тег с помощью параметра `style`. Обратите внимание, параметр `style` поддерживают все теги:

```
<font style="font-size: 12pt">Текст</font>
```

Если определение стиля состоит из нескольких атрибутов, то они указываются через точку с запятой:

```
<font style="font-size: 12pt; color: red">Текст</font>
```

Если какое-либо значение атрибута требует наличия кавычек, то оно указывается в апострофах:

```
<font style="font-size: 12pt; color: red;
font-family: 'Times New Roman'">Текст</font>
```

2.2.2. Встраивание определения стилей в заголовок HTML-документа

Все определения стилей можно собрать в одном месте (листинг 2.1). В этом случае стили указываются между тегами `<style>` и `</style>`. Сам тег `<style>` должен быть расположен в разделе `HEAD` HTML-документа. А в теге, к которому нужно применить стиль, указывается имя стиля с помощью параметра `class`.

Листинг 2.1. Пример использования стилей

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Пример использования стилей</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
  .text1 { font-size: 12pt; color: red; font-family: "Arial" }
  font { font-size: 12pt; color: green; font-family: "Arial" }
  font.text2 { font-size: 12pt; color: blue; font-family: "Arial" }
 </style>
</head>
```

```
<body>
 <font class="text1">Текст1</font><br><!-- Красный текст -->
 <font>Текст2</font><br><!-- Зеленый текст -->
 <font class="text2">Текст3</font><br><!-- Синий текст -->
 <p class="text2">Текст4</p><!-- Цвет по умолчанию -->
 <p class="text1">Текст5</p><!-- Красный текст -->
</body>
</html>
```

Атрибуты определения стиля, указанные между тегами `<style>` и `</style>`, заключаются в фигурные скобки. Если атрибутов несколько, то они перечисляются через точку с запятой:

```
<Селектор> { <Атрибут 1>: <Значение 1>; ..., <Атрибут N>: <Значение N> }
```

В параметре `<Селектор>` могут быть указаны следующие селекторы:

- `*` — все теги. Уберем все внешние и внутренние отступы:

```
* { margin: 0; padding: 0 }
```

- `Тег` — все теги, имеющие указанное имя:

```
font { font-size: 12pt; color: green; font-family: "Arial" }
...
<font>Текст2</font><!-- Зеленый текст -->
```

- `.Класс` — все теги, имеющие указанный класс:

```
.text1 { font-size: 12pt; color: red; font-family: "Arial" }
...
<font class="text1">Текст1</font><!-- Красный текст -->
<p class="text1">Текст2</p><!-- Красный текст -->
```

  И "Текст1" и "Текст2" будут красного цвета, хотя они находятся в разных тегах;

- `Тег.Класс` — все теги, имеющие указанные имя и класс:

```
font.text2 { font-size: 12pt; color: blue }
...
<font class="text2">Текст1</font><!-- Синий текст -->
```

  Обратите внимание, что если имя класса указать в другом теге, то стиль не будет применен к этому тегу:

```
<p class="text2">Текст2</p>
```

  В данном случае фрагмент текста "Текст2" не будет отображен синим цветом, т. к. имя класса `text2` применяется только к тегу `<font>`;

- `#Идентификатор` — тег с указанным идентификатором:

```
#txt1 { color: red }
...
<p id="txt1">Текст</p>
```

- `Тег#Идентификатор` — идентификатор, указанный в определенном теге:

```
p#txt1 { color: red }
...
<p id="txt1">Текст</p>
```

Если идентификатор находится в другом теге, то стиль не будет применен к этому тегу.

Стилевой класс можно привязать сразу к нескольким тегам. В этом случае селекторы указываются через занятую:

```
h1, h2 { font-family: "Arial" }
```

Привязаться к другим элементам можно следующими способами:

- `Селектор1 Селектор2` — все элементы, соответствующие параметру `Селектор2`, которые располагаются внутри контейнера, соответствующего параметру `Селектор1`:

```
div a { color: red }
```

Цвет текста ссылки станет красным, если тег `<a>` находится внутри тега `<div>`:

```
<div><a href="link.html">Ссылка</a></div>
```

- `Селектор1 > Селектор2` — все элементы, соответствующие параметру `Селектор2`, которые являются дочерними для контейнера, соответствующего параметру `Селектор1`:

```
div > a { color: red }
```

Цвет текста ссылки станет красным, если тег `<a>` находится внутри тега `<div>` и не вложен в другой тег:

```
<div>
<a href="link1.html">Ссылка 1</a><br>
<span><a href="link2.html">Ссылка 2</a></span>
</div>
```

В этом примере только первая ссылка станет красного цвета, т. к. вторая ссылка расположена внутри тега `<span>`;

> ***Примечание***
>
> Web-браузер Internet Explorer поддерживает селектор, начиная с версии 7.0.

- `Селектор1 + Селектор2` — элемент, соответствующий параметру `Селектор2`, который является соседним для контейнера, соответствующего параметру `Селектор1`, и следует сразу после него:

```
div + a { color: red }
```

Цвет текста ссылки станет красным, если тег `<a>` следует сразу после элемента `div`:

```
<div>Текст</div><a href="link.html">Ссылка</a>
```

Примечание

Web-браузер Internet Explorer поддерживает селектор, начиная с версии 8.0.

При необходимости можно составлять выражения из нескольких селекторов:

```
div span a { color: red }
```

Цвет текста ссылки станет красным, если тег <a> расположен внутри тега , а тот в свою очередь вложен в тег :

```
<div>
  <a href="link1.html">Ссылка 1</a><br>
  <span>
    <a href="link2.html">Ссылка 2</a><br>
  </span>
</div>
```

В этом примере только ссылка 2 будет красного цвета.

Для привязки к параметрам тегов применяются следующие селекторы:

- [Параметр] — элементы с указанным параметром:

  ```
  a[id] { color: red }
  ```

  Цвет текста ссылки станет красным, если тег <a> имеет параметр id:

  ```
  <a id="link1" href="link1.html">Ссылка 1</a>
  ```

- [Параметр='Значение'] — элементы, у которых параметр точно равен значению:

  ```
  a[href="link1.html"] { color: red }
  ```

  Цвет текста ссылки станет красным, если параметр href тега <a> имеет значение "link1.html";

- [Параметр^='Значение'] — элементы, у которых параметр начинается с указанного значения:

  ```
  a[href^="li"] { color: red }
  ```

  Цвет текста ссылки станет красным, если значение параметра href тега <a> начинается с "li";

- [Параметр$='Значение'] — элементы, у которых параметр оканчивается указанным значением:

  ```
  a[href$=".html"] { color: red }
  ```

  Цвет текста ссылки станет красным, если значение параметра href тега <a> оканчивается на расширение ".html";

- [Параметр*='Значение'] — элементы, у которых параметр содержит указанное значение:

  ```
  a[href*="link"] { color: red }
  ```

  Цвет текста ссылки станет красным, если значение параметра href тега <a> содержит фрагмент "link".

В качестве селектора могут быть также указаны следующие псевдоэлементы:

- `:first-letter` — задает стиль для первой буквы. Выделим первую букву всех абзацев:

```
p:first-letter { font-size: 150%; font-weight: bold;
                 color: red }
```

- `:first-line` — задает стиль для первой строки. Пример:

```
p:first-line { font-weight: bold; color: red }
```

- `:before` и `:after` — позволяют добавить текст в начало и конец элемента соответственно. Добавляемый текст должен быть указан в атрибуте `content`:

```
p:before { content: "before "; }
p:after { content: " after"; }
...
<p>Текст</p>
```

Результат:

```
before Текст after
```

> ***Примечание***
>
> Web-браузер Internet Explorer поддерживает псевдоэлементы `:before` и `:after`, начиная с версии 8.0.

2.2.3. Вынесение таблицы стилей в отдельный файл

Таблицу стилей можно вынести в отдельный файл. Файл с таблицей стилей обычно имеет расширение css и может редактироваться любым текстовым редактором, например Блокнотом. Задать расширение файлу можно точно так же, как и при создании файла с расширением html.

Вынесем таблицу стилей в отдельный файл style.css (листинг 2.2) и подключим его к основному документу test.html (листинг 2.3).

Листинг 2.2. Содержимое файла style.css

```
/* Так можно вставить комментарий */
.text1 { /* Стиль для элементов с class="text1" */
   font-size: 12pt; /* Размер шрифта */
   color: red; /* Цвет текста */
   font-family: Arial /* Название шрифта */
}
font { /* Стиль для всех тегов FONT */
   font-size: 12pt; /* Размер шрифта */
   color: green; /* Цвет текста */
```

```
   font-family: Arial /* Название шрифта */
}
font.text2 { /* Стиль для тегов FONT с class="text2" */
   font-size: 12pt; /* Размер шрифта */
   color: blue; /* Цвет текста */
   font-family: Arial /* Название шрифта */
}
```

Листинг 2.3. Содержимое файла test.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Пример использования стилей</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
 <font class="text1">Текст1</font><br><!-- Красный текст -->
 <font>Текст2</font><br><!-- Зеленый текст -->
 <font class="text2">Текст3</font><br><!-- Синий текст -->
 <p class="text2">Текст4</p><!-- Цвет по умолчанию -->
 <p class="text1">Текст5</p><!-- Красный текст -->
</body>
</html>
```

Сохраним оба файла в одной папке и откроем файл test.html в Web-браузере. Результат будет таким же, как и в предыдущем примере.

Отдельный файл с таблицей стилей прикрепляется к HTML-документу с помощью одинарного тега `<link>`. В параметре `href` указывается абсолютный или относительный URL-адрес файла, а в параметре `rel` должно быть значение `stylesheet`, показывающее, что присоединяемый таким образом документ содержит таблицу стилей:

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

Подключить внешний CSS-файл можно также с помощью правила `@import`:

```
@import url(<URL-адрес>)[ <Тип устройства>];
@import <URL-адрес>[ <Тип устройства>];
```

Правило `@import`: должно быть расположено внутри тега `<style>`:

```
<style type="text/css">
  @import url("style.css");
</style>
```

В необязательном параметре <Тип устройства> можно указать устройство, для которого предназначена подключаемая таблица стилей. Например, all — для любых устройств, print — для предварительного просмотра и распечатки документа. Пример:

```
<style type="text/css">
  @import "style.css" print;
</style>
```

Таблицу стилей, вынесенную в отдельный файл, можно использовать в нескольких HTML-документах.

2.2.4. Приоритет применения стилей

Предположим, что для абзаца определен атрибут color в параметре style одного цвета, в теге <style> другого цвета, а в отдельном файле (листинг 2.4) — третьего цвета. Кроме того, в параметре color тега <font> задан четвертый цвет (листинг 2.5).

Листинг 2.4. Содержимое файла style.css

```
p { color: red }
```

Листинг 2.5. Содержимое файла test.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Приоритет применения стилей</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <link rel="stylesheet" type="text/css" href="style.css">
 <style type="text/css">
   p { color: blue }
 </style>
</head>
<body>
 <p style="color: green"><font color="yellow">Текст1</font></p><br>
 <p style="color: green">Текст2</p>
</body>
</html>
```

Какого цвета будет абзац со словом "Текст1"? И какого цвета будет абзац со словом "Текст2"? Для ответа на эти вопросы и существует приоритет стилей:

- стиль, заданный таблицей стилей, будет отменен, если в HTML-коде явно описано форматирование блока;

- стиль, заданный в теге `<style>`, будет отменен, если в параметре `style` тега указан другой стиль;
- стиль, заданный в отдельном файле, будет отменен, если в теге `<style>` указано другое определение стиля.

Именно из-за такой структуры приоритетов таблицы стилей называют *каскадными*.

Иными словами, наименьший приоритет имеет стиль, описанный в отдельном файле, а самый высокий — стиль, указанный последним. В нашем примере к абзацу со словом "Текст1" будет применено форматирование, указанное параметром `color` тега `<font>`, т. е. абзац будет желтого цвета. А абзац со словом "Текст2" будет иметь цвет, указанный в параметре `style`, т. е. зеленый.

Кроме того, следует учитывать, что стиль, заданный через идентификатор, будет иметь более высокий приоритет, чем стиль, заданный через класс. Пример:

```
<style type="text/css">
   #id1 { color: red }
   .cls1 { color: blue }
</style>
...
<p id="id1" class="cls1">Текст1</p>
```

В этом примере текст абзаца будет красного цвета, а не синего.

С помощью свойства `!important` можно изменить приоритет. Для примера изменим содержимое файла style.css (листинг 2.4) на:

```
p { color: red !important }
```

В результате мы переопределили все стили и абзац со словом "Текст2" будет иметь красный цвет, а не зеленый. Однако абзац со словом "Текст1" так и останется желтого цвета, т. к. цвет указан в параметре `color` тега `<font>`, а не задан через стиль.

2.3. Единицы измерения в CSS

Размеры в CSS можно задавать в абсолютных или относительных единицах.

Абсолютные единицы:

- пиксел (`px`);
- миллиметр (`mm`);
- сантиметр (`cm`);
- дюйм (`in`) — `1 in = 2.54 cm`;
- пункт (`pt`) — `1 pt = 1/72 in`;
- пика (`pc`) — `1 pc = 12 pt`.

Относительные единицы:

- процент (%);
- высота текущего шрифта (em);
- высота буквы "x" текущего шрифта (ex).

Для значения 0 можно не указывать единицы измерения.

Цвет можно задать одним из следующих способов:

- именем цвета — blue, green и т. д.:

```
p { color: red }
```

- значением вида #[R][G][B], где R — насыщенность красного, G — насыщенность зеленого и B — насыщенность синего в цвете. Значения задаются одинарными шестнадцатеричными числами от 0 до F:

```
p { color: #F00 }
```

- значением вида #[RR][GG][BB], где RR — насыщенность красного, GG — насыщенность зеленого и BB — насыщенность синего в цвете. В таком формате значения задаются двузначными шестнадцатеричными числами от 00 до FF:

```
p { color: #FF0000 }
```

- значением вида rgb([R], [G], [B]), где R, G и B — насыщенности красного, зеленого и синего цветов, которые задаются десятичными числами от 0 до 255:

```
p { color: rgb(255, 0, 0) }
```

- значением вида rgb([R%], [G%], [B%]), где R%, G% и B% — насыщенности красного, зеленого и синего цветов, которые задаются в процентах:

```
p { color: rgb(100%, 0%, 0%) }
```

Все перечисленные примеры задают красный цвет.

2.4. Форматирование шрифта

Каскадные таблицы стилей позволяют задать название, цвет и размер шрифта, его стиль и "жирность". Кроме того, можно указать несколько имен шрифтов и одно из названий альтернативных семейств. Ведь на компьютере пользователя может не быть нужного шрифта.

2.4.1. Имя шрифта

Атрибут font-family позволяет задать имя шрифта:

```
p { font-family: "Arial" }
```

В ряде случаев шрифт может отсутствовать на компьютере пользователя. Поэтому лучше указывать несколько альтернативных шрифтов. Имена шрифтов указываются через запятую:

```
p { font-family: "Verdana", "Tahoma" }
```

Можно также указать одно из пяти типовых семейств шрифтов — `serif`, `sans-serif`, `cursive`, `fantasy` или `monospace`:

```
p { font-family: "Verdana", "Tahoma", sans-serif }
```

2.4.2. Стиль шрифта

Атрибут `font-style` позволяет задать стиль шрифта. Он может принимать следующие значения:

- `normal` — нормальный шрифт:

```
p { font-family: "Arial"; font-style: normal }
```

- `italic` — курсивный шрифт:

```
p { font-family: "Arial"; font-style: italic }
```

- `oblique` — наклонный шрифт:

```
p { font-family: "Arial"; font-style: oblique }
```

2.4.3. Размер шрифта

Атрибут `font-size` позволяет задать размер шрифта:

```
.text1 { font-size: 12pt; font-family: "Arial" }
```

Можно указать абсолютную величину или одну из типовых констант — `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` или `xx-large`:

```
.text1 { font-size: large; font-family: "Arial" }
```

Кроме того, можно указать относительную величину (например, значение в процентах) или одну из двух констант — `larger` или `smaller`:

```
.text1 { font-size: 150%; font-family: "Arial" }
.text2 { font-size: smaller; font-family: "Arial" }
```

2.4.4. Цвет шрифта

Атрибут `color` позволяет задать цвет шрифта:

```
.text1 { font-size: 12pt; font-family: "Arial"; color: red }
.text2 { font-size: 12pt; font-family: "Arial"; color: #00FF00 }
.text3 { font-size: 12pt; font-family: "Arial"; color: rgb(255, 0, 0) }
```

2.4.5. Жирность шрифта

Атрибут `font-weight` позволяет управлять жирностью шрифта. Может принимать следующие значения:

- `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, `900` — значение `100` соответствует самому бледному шрифту, а `900` — самому жирному:

```
p { font-family: "Arial"; font-style: italic; font-weight: 700 }
```

- `normal` — нормальный шрифт. Соответствует значению 400:

```
p { font-family: "Arial"; font-style: italic; font-weight: normal }
```

- `bold` — полужирный шрифт. Соответствует значению 700:

```
p { font-family: "Arial"; font-style: italic; font-weight: bold }
```

2.5. Форматирование текста

Для текстовых фрагментов, кроме указания характеристик шрифтов, можно задать некоторые дополнительные параметры — расстояние между символами, словами и строками, вертикальное и горизонтальное выравнивание, отступ первой строки.

2.5.1. Расстояние между символами в словах

Атрибут `letter-spacing` задает расстояние между символами текста. Он может принимать следующие значения:

- `normal` — значение по умолчанию:

```
p { letter-spacing: normal; font-style: italic;
font-weight: normal }
```

- абсолютная величина в поддерживаемых CSS единицах:

```
p { font-size: 12pt; color: red; letter-spacing: 5mm }
```

2.5.2. Расстояние между словами

Атрибут `word-spacing` задает расстояние между словами. Он может принимать следующие значения:

- `normal` — значение по умолчанию:

```
p { word-spacing: normal; font-style: italic; font-weight: normal }
```

- абсолютная величина в поддерживаемых CSS единицах:

```
p { font-size: 12pt; color: red; word-spacing: 5mm }
```

2.5.3. Отступ первой строки

Атрибут `text-indent` задает отступ для "красной строки". Может задаваться абсолютная или относительная величина отступа:

```
p { text-indent: 10mm; font-style: italic; font-weight: normal }
```

2.5.4. Вертикальное расстояние между строками

Атрибут `line-height` задает вертикальное расстояние между базовыми линиями двух строк. Он может принимать следующие значения:

- `normal` — значение по умолчанию:

```
p { line-height: normal; font-style: italic; font-weight: normal }
```

- абсолютная или относительная величина в поддерживаемых CSS единицах:

```
p { font-size: 12pt; color: red; font-family: "Arial";
line-height: 5mm }
```

2.5.5. Горизонтальное выравнивание текста

Атрибут `text-align` задает горизонтальное выравнивание текста. Этот атрибут может принимать следующие значения:

- `center` — выравнивание по центру:

```
<p style="text-align: center">Абзац с выравниванием
по центру</p>
```

- `left` — выравнивание по левому краю:

```
<p style="text-align: left">Абзац с выравниванием
по левому краю</p>
```

- `right` — выравнивание по правому краю:

```
<p style="text-align: right">Абзац с выравниванием
по правому краю</p>
```

- `justify` — выравнивание по ширине (по двум сторонам):

```
<p style="text-align: justify">Абзац с выравниванием
по ширине</p>
```

2.5.6. Вертикальное выравнивание текста

Атрибут `vertical-align` задает вертикальное выравнивание текста относительно элемента-родителя, например, ячейки таблицы. Он может принимать следующие значения:

- `baseline` — по базовой линии:

```
td { font-size: 12pt; color: red; vertical-align: baseline }
```

- `middle` — по центру:

```
td { font-size: 12pt; color: red; vertical-align: middle }
```

- `top` — по верху:

```
td { font-size: 12pt; color: red; vertical-align: top }
```

- `bottom` — по низу:

```
td { font-size: 12pt; color: red; vertical-align: bottom }
```

2.5.7. Подчеркивание, надчеркивание и зачеркивание текста

Атрибут `text-decoration` позволяет подчеркнуть, надчеркнуть или зачеркнуть текст. Он может принимать следующие значения:

- `none` — обычный текст (по умолчанию):

```
<p style="text-decoration: none">Текст</p>
```

- `underline` — подчеркивает текст:

```
<p style="text-decoration: underline">Подчеркнутый текст</p>
```

- `overline` — проводит линию над текстом:

```
<p style="text-decoration: overline">Надчеркнутый текст</p>
```

- `line-through` — зачеркивает текст:

```
<p style="text-decoration: line-through">Зачеркнутый текст</p>
```

- `blink` — мигающий текст:

```
<p style="text-decoration: blink">Мигающий текст</p>
```

> ***Примечание***
> Web-браузер Internet Explorer не поддерживает значение `blink`.

2.5.8. Изменение регистра символов

Атрибут `text-transform` позволяет изменить регистр символов. Он может принимать следующие значения:

- `capitalize` — делает первую букву каждого слова прописной;
- `uppercase` — преобразует все буквы в прописные;
- `lowercase` — преобразует все буквы в строчные;
- `none` — без преобразования.

Пример:

```
<h1 style="text-transform: capitalize">заголовок из нескольких слов</h1>
<h1 style="text-transform: uppercase">заголовок2</h1>
<h1 style="text-transform: lowercase">ЗАГОЛОВОК3</h1>
```

Результат:

```
Заголовок Из Нескольких Слов
ЗАГОЛОВОК2
заголовок3
```

2.5.9. Обработка пробелов между словами

Атрибут `white-space` позволяет установить тип обработки пробелов. По умолчанию несколько пробелов подряд выводятся в окне Web-браузера как один пробел. Атрибут может принимать следующие значения:

- `normal` — текст выводится стандартным образом:

```
<p style="white-space: normal">
Строка          1
Строка 2
</p>
```

Результат в окне Web-браузера:

```
Строка 1 Строка 2
```

- `pre` — сохраняются все пробелы и переносы строк:

```
<p style="white-space: pre">
Строка          1
Строка 2
</p>
```

Результат в окне Web-браузера:

```
Строка          1
Строка 2
```

- `nowrap` — переносы строк в HTML-коде игнорируются. Если внутри строки содержится тег `<br>`, то он вставляет перенос строки:

```
<p style="white-space: nowrap">
Строка          1
Строка 2<br>
Строка 3
</p>
```

Результат в окне Web-браузера:

```
Строка 1 Строка 2
Строка 3
```

2.6. Отступы

Любой элемент Web-страницы занимает в окне Web-браузера некоторую прямоугольную область. Причем эта область имеет как внутренние, так и внешние отступы. Внутренний отступ — это расстояние между элементом страницы и реальной или воображаемой границей области. Внешний отступ — это расстояние между реальной или воображаемой границей и другим элементом Web-страницы, точнее сказать, между границей и крайней точкой внешнего отступа другого элемента Web-страницы.

2.6.1. Внешние отступы

С помощью атрибутов `margin-left`, `margin-right`, `margin-top` и `margin-bottom` можно задать отступы одного элемента Web-страницы от другого. Может быть задано абсолютное или относительное значение. Более того, атрибуты могут иметь отрицательные значения.

Эти атрибуты означают следующее:

- `margin-left` — отступ слева:

```
body { margin-left: 0 }
```

- `margin-right` — отступ справа:

```
body { margin-right: 5% }
```

- `margin-top` — отступ сверху:

```
body { margin-top: 15mm }
```

- `margin-bottom` — отступ снизу:

```
body { margin-bottom: 20px }
```

Убрать все внешние отступы можно с помощью этой строки кода:

```
<body style="margin-left: 0; margin-right: 0; margin-top: 0;
margin-bottom: 0">
```

Или так:

```
body { margin-left: 0; margin-right: 0; margin-top: 0; margin-bottom: 0 }
```

С помощью атрибута `margin` можно задать все внешние отступы за один раз:

```
margin: <top> <right> <bottom> <left>
```

Например:

```
body { margin: 15mm 5% 20px 0 }
```

Для совпадающих значений допускается и более короткая запись:

```
body { margin: 0 }
```

2.6.2. Внутренние отступы

С помощью атрибутов `padding-left`, `padding-right`, `padding-top` и `padding-bottom` можно задать отступы от элемента Web-страницы до его рамки (если она есть). Например, ими задается расстояние между текстом и рамкой ячейки таблицы. Может быть задано абсолютное или относительное значение:

- `padding-left` — отступ слева:

```
td { padding-left: 0 }
```

- `padding-right` — отступ справа:

```
td { padding-right: 50px }
```

- `padding-top` — отступ сверху:

```
td { padding-top: 15mm }
```

- `padding-bottom` — отступ снизу:

```
td { padding-bottom: 20px }
```

С помощью атрибута `padding` можно задать все внутренние отступы за один раз:

```
padding: <top> <right> <bottom> <left>
```

Например:

```
td { padding: 15mm 50px 20px 0 }
```

Совпадающие отступы можно задать и короче:

```
td { padding: 5px }
```

2.7. Рамки

Как вы уже знаете, любой элемент Web-страницы занимает в окне Web-браузера некоторую прямоугольную область. Содержимое этой области может быть окружено рамками. Иными словами, рамки могут иметь не только таблицы, но и любые элементы Web-страницы, например абзацы.

2.7.1. Стиль линии рамки

С помощью атрибутов `border-left-style` (левая линия), `border-right-style` (правая линия), `border-top-style` (верхняя линия) и `border-bottom-style` (нижняя линия) можно задать тип линии рамки. Могут принимать следующие значения:

- `none` — линия не отображается;
- `solid` — линия отображается сплошной линией;
- `dotted` — пунктирная линия;
- `dashed` — штриховая линия;
- `double` — линия отображается в виде двойной линии;
- `groove` — вдавленная рельефная линия;
- `ridge` — выпуклая рельефная линия;
- `inset` — весь блок элемента отображается, как будто он вдавлен в лист;
- `outset` — весь блок элемента отображается, как будто он выдавлен из листа.

В качестве примера укажем стиль линии рамки для разных элементов Web-страницы (листинг 2.6).

Листинг 2.6. Тип рамки

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

```
<html>
<head>
 <title>Тип рамки</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
  table { border-left-style: dashed; border-right-style: dotted;
    border-top-style: solid; border-bottom-style: groove }
  td { border-left-style: none; border-right-style: none;
  border-top-style: none; border-bottom-style: none; text-align: center }
  caption { border-top-style: solid }
  p { border-left-style: dotted; border-right-style: dotted;
    border-top-style: dotted; border-bottom-style: dotted }
 </style>
</head>
<body>
 <table width="200">
  <caption>Заголовок таблицы</caption>
  <tr>
   <td>1</td>
   <td>2</td>
  </tr>
  <tr>
   <td>3</td>
   <td>4</td>
  </tr>
 </table>
 <p>Текст в пунктирной рамке</p>
</body>
</html>
```

Этот пример показывает, что указать тип рамки можно не только для границ таблицы, но и для заголовков таблицы, и даже для абзацев.

Эти атрибуты могут быть объединены в одном атрибуте `border-style`:

```
border-style: <top> <right> <bottom> <left>
```

Если все значения совпадают, можно указать это значение один раз.

2.7.2. Толщина линии рамки

С помощью атрибутов `border-left-width` (левая линия), `border-right-width` (правая линия), `border-top-width` (верхняя линия) и `border-bottom-width` (нижняя линия) можно задать толщину линии рамки. Может быть задано абсолютное значение:

```
table { border-left-width: 5px; border-right-width: 5px;
border-top-width: 0; border-bottom-width: 10px }
```

Также можно указать одно из предопределенных значений:

- `thin` — тонкая линия;
- `medium` — средняя толщина линии;
- `thick` — толстая линия.

```
table { border-left-width: medium; border-right-width: medium;
border-top-width: thin; border-bottom-width: thick }
```

Эти атрибуты могут быть объединены в одном атрибуте `border-width`:

```
border-width: <top> <right> <bottom> <left>
```

Если значения совпадают, можно указать их один раз.

2.7.3. Цвет линии рамки

С помощью атрибутов `border-left-color` (левая линия), `border-right-color` (правая линия), `border-top-color` (верхняя линия) и `border-bottom-color` (нижняя линия) можно задать цвет линий рамки:

```
table { border-left-color: red; border-right-color: #000080;
border-top-color: green; border-bottom-color: black }
```

Как и в случае `border-style` и `border-width`, эти атрибуты можно объединить в один атрибут `border-color`.

2.7.4. Одновременное задание атрибутов рамки

Если атрибуты рамки одинаковы для всех ее сторон, можно задавать их в одном атрибуте `border`:

```
border: <стиль> <толщина> <цвет>
```

Поскольку значение атрибута однозначно определяет, к какому именно компоненту он относится, то их можно указывать в произвольном порядке:

```
td { border: red thin solid }
```

2.8. Фон элемента

Каскадные таблицы стилей позволяют задать цвет фона или использовать изображение в качестве фона. Для фонового рисунка можно задать начальное положение и указать, будет ли рисунок прокручиваться вместе с содержимым Web-страницы. Кроме того, можно контролировать, как фоновый рисунок повторяется, что позволяет получить интересные спецэффекты. Например, если в качестве фонового рисунка указать градиентную полосу с высотой, равной высоте элемента страницы, и шириной, равной 1—2 мм, а затем задать режим повторения только по горизонтали, то первоначальное изображение будет размножено по горизонтали, и мы получим градиентную полосу любой ширины.

2.8.1. Цвет фона

С помощью атрибута `background-color` можно задать цвет фона:

```
body { background-color: green }
td { background-color: silver }
```

В качестве значения атрибута можно использовать слово `transparent`. Оно означает, что фон прозрачный:

```
td { background-color: transparent }
```

2.8.2. Фоновый рисунок

С помощью атрибута `background-image` можно задать URL-адрес изображения, которое будет использовано в качестве фонового рисунка. Может быть указан абсолютный или относительный URL-адрес (относительно местоположения таблицы стилей, а не документа):

```
body { background-image: url(foto1.gif) }
```

В качестве значения атрибута можно использовать слово `none`. Оно означает, что фоновая заливка отключена:

```
body { background-image: none }
```

2.8.3. Режим повтора фонового рисунка

Атрибут `background-repeat` задает режим повтора фонового рисунка. Он может принимать следующие значения:

- `repeat` — рисунок повторяется и по вертикали, и по горизонтали (по умолчанию):

  ```
  body { background-image: url(foto1.gif); background-repeat: repeat }
  ```

- `repeat-x` — рисунок повторяется по горизонтали:

  ```
  body { background-image: url(foto1.gif); background-repeat: repeat-x }
  ```

- `repeat-y` — рисунок повторяется по вертикали:

  ```
  body { background-image: url(foto1.gif); background-repeat: repeat-y }
  ```

- `no-repeat` — рисунок не повторяется:

  ```
  body { background-image: url(foto1.gif); background-repeat: no-repeat }
  ```

2.8.4. Прокрутка фонового рисунка

Атрибут `background-attachment` определяет, будет ли фоновый рисунок прокручиваться вместе с документом. Он может принимать следующие значения:

- `scroll` — фоновый рисунок прокручивается вместе с содержимым страницы (по умолчанию):

```
body { background-image: url(foto1.gif);
background-repeat: no-repeat; background-attachment: scroll }
```

- `fixed` — фоновый рисунок не прокручивается:

```
body { background-image: url(foto1.gif);
background-repeat: no-repeat; background-attachment: fixed }
```

2.8.5. Положение фонового рисунка

Атрибут `background-position` задает начальное положение фонового рисунка. В качестве значений атрибута задаются координаты в абсолютных единицах или в процентах. Координаты указываются через пробел:

```
body { background-image: url(foto1.gif); background-repeat: no-repeat;
background-attachment: fixed; background-position: 50% 50% }
```

Кроме того, могут быть указаны следующие значения:

- `left` — выравнивание по левому краю;
- `right` — по правому краю;
- `center` — по центру;
- `top` — по верху;
- `bottom` — по низу.

Пример:

```
body { background-image: url(foto1.gif); background-repeat: no-repeat;
background-attachment: fixed; background-position: left center }
```

2.8.6. Одновременное задание атрибутов фона

Атрибут `background` является сокращенным вариантом для одновременного указания значений атрибутов `background-color`, `background-image`, `background-position`, `background-repeat` и `background-attachment`. Пример:

```
body { background: green url(foto1.gif) no-repeat fixed left center }
body { background: green }
```

Обратите внимание на то, что во втором примере мы указали только цвет фона. Если остальные атрибуты не указаны, то они получают значения по умолчанию. Кроме того, поскольку значение атрибута однозначно определяет, к какому именно компоненту он относится, то их можно указывать в произвольном порядке.

2.9. Списки

Задать параметры списка можно не только с помощью параметров тегов, но и с помощью атрибутов стиля. Более того, каскадные таблицы стилей позволяют использовать в качестве маркера списка любое изображение.

2.9.1. Вид маркера списка

Атрибут `list-style-type` задает вид маркера списка. Он может принимать следующие значения:

- `disc` — выводит значки в форме кружков с заливкой:

```
ul { list-style-type: disc }
```

- `circle` — выводит значки в форме кружков без заливки:

```
ul { list-style-type: circle }
```

- `square` — выводит значки в форме квадрата с заливкой:

```
ul { list-style-type: square }
```

- `decimal` — нумерует строки арабскими цифрами:

```
ol { list-style-type: decimal }
```

- `lower-roman` — нумерует строки малыми римскими цифрами:

```
ol { list-style-type: lower-roman }
```

- `upper-roman` — нумерует строки большими римскими цифрами:

```
ol { list-style-type: upper-roman }
```

- `lower-alpha` — нумерует строки малыми латинскими буквами:

```
ol { list-style-type: lower-alpha }
```

- `upper-alpha` — нумерует строки большими латинскими буквами:

```
ol { list-style-type: upper-alpha }
```

- `none` — никак не помечает позиции списка:

```
ol { list-style-type: none }
```

2.9.2. Изображение в качестве маркера списка

Атрибут `list-style-image` задает URL-адрес изображения, которое будет использовано в качестве маркера списка.

Относительные адреса указываются относительно расположения таблицы стилей, а не HTML-документа:

```
ol { list-style-image: url(foto1.gif) }
```

2.9.3. Компактное отображение списка

Атрибут `list-style-position` позволяет задать более компактное отображение списка. Может принимать следующие значения:

- `outside` — по умолчанию. Маркер отображается отдельно от текста:

```
ol { list-style-position: outside }
```

- `inside` — более компактное отображение списка. Маркер входит в состав текста:

```
ol { list-style-position: inside }
```

2.10. Вид курсора

Атрибут `cursor` задает форму курсора мыши при наведении на элемент страницы. Может принимать следующие значения:

- `auto` — Web-браузер сам определяет форму курсора мыши:

```
p { cursor: auto }
```

- `crosshair` — в виде креста:

```
p { cursor: crosshair }
```

- `default` — стрелка (курсор по умолчанию):

```
p { cursor: default }
```

- `pointer` — в виде руки:

```
p { cursor: pointer }
```

- `move` — стрелка, указывающая во все направления:

```
p { cursor: move }
```

- `n-resize`, `ne-resize`, `nw-resize`, `s-resize`, `se-resize`, `sw-resize`, `e-resize`, `w-resize` — стрелки, показывающие направление:

```
p { cursor: n-resize }
```

- `text` — текстовый курсор:

```
p { cursor: text }
```

- `wait` — песочные часы:

```
p { cursor: wait }
```

- `progress` — стрелка с песочными часами:

```
p { cursor: progress }
```

- `help` — стрелка с вопросительным знаком:

```
p { cursor: help }
```

2.11. Псевдостили гиперссылок. Отображение ссылок разными цветами

Большинство Web-браузеров позволяют отобразить посещенные и непосещенные ссылки разными цветами. Достигается это при помощи предопределенных стилей — псевдостилей:

- `a:link` — вид непосещенной ссылки;
- `a:visited` — вид посещенной ссылки;
- `a:active` — вид активной ссылки;
- `a:hover` — вид ссылки, на которую указывает курсор мыши.

> ***Внимание!***
> До и после двоеточия не должно быть пробелов.

Псевдостили аналогичны параметрам `link`, `vlink` и `alink` тега `<body>`:

```
<body link="#000000" vlink="#000080" alink="#FF0000">
```

эквивалентно заданию стиля

```
a:link { color: #000000 }
a:visited { color: #000080 }
a:active { color: #FF0000 }
```

С помощью псевдостилей можно менять не только цвет ссылки, но и задать, будет ли ссылка подчеркнута:

```
a:link { color: red; text-decoration: underline }
a:visited { color: blue; text-decoration: underline }
a:active { color: green; text-decoration: none }
a:hover { color: lime; text-decoration: none }
```

Кроме того, можно применить стиль гиперссылок не только ко всему документу, но и к определенному классу:

```
a.link1:link { color: black; text-decoration: none }
a.link1:visited { color: blue; text-decoration: none }
a.link1:active { color: red; text-decoration: underline }
a.link1:hover { color: red; text-decoration: underline }
```

В листинге 2.7 продемонстрирована возможность задания внешнего вида гиперссылок для всего документа, а также для определенного класса.

Листинг 2.7. Псевдостили гиперссылок

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Псевдостили гиперссылок</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
   a:link { color: red; text-decoration: underline }
   a:visited { color: blue; text-decoration: underline }
   a:active { color: green; text-decoration: none }
   a:hover  { color: lime; text-decoration: none }
```

```
   a.link1:link { color: black; text-decoration: none }
   a.link1:visited { color: blue; text-decoration: none }
   a.link1:active { color: red; text-decoration: underline }
   a.link1:hover { color: red; text-decoration: underline }
 </style>
</head>
<body>
 <p>
  <a href="doc1.html">Ссылка1</a><br>
  <a href="doc2.html" class="link1">Ссылка2</a>
 </p>
</body>
</html>
```

2.12. Форматирование блоков

Как вы уже знаете, любой элемент Web-страницы занимает в окне Web-браузера некоторую прямоугольную область. Эта область имеет как внутренние, так и внешние отступы, а также содержит реальную или воображаемую границу. Тип блочной модели зависит от формата документа. Если тег `<!DOCTYPE>` указан, то блочная модель соответствует стандартам консорциума W3C. Реальные размеры элемента вычисляются так:

```
Реальная ширина = margin-left + border-left-width + padding-left +
                  width + padding-right + border-right-width +
                  margin-right
Реальная высота = margin-top + border-top-width + padding-top + height +
                  padding-bottom + border-bottom-width + margin-bottom
```

Если тег `<!DOCTYPE>` не указан, то Web-браузер Internet Explorer переходит в режим совместимости (Quirks Mode). В этом режиме `padding` и `border` входят в состав `width` и `height`, а следовательно, реальные размеры будут другие:

```
Реальная ширина = margin-left + width + margin-right
Реальная высота = margin-top + height + margin-bottom
```

Поэтому при отсутствии тега `<!DOCTYPE>` разные Web-браузеры могут по-разному отображать Web-страницу.

> ***Примечание***
>
> Более подробную информацию о типах блочной модели можно получить в Интернете на странице консорциума W3C **http://www.w3.org/TR/CSS2/box.html** и на странице **http://www.quirksmode.org/css/box.html**.

2.12.1. Указание типа блока

Атрибут `display` предназначен для указания типа блока. Может принимать следующие значения:

- `block` — блок занимает всю ширину родительского элемента. Значение `block` по умолчанию имеют теги `<div>` и `<p>`;
- `inline` — блок занимает только необходимое для отображения содержимого пространство. Значение `inline` по умолчанию имеют теги `<span>`, `<b>` и др.;
- `inline-block` — аналогично `inline`, но дополнительно можно задать размеры и другое форматирование, применяемое для блочного элемента. Результат аналогичен встраиванию тега `<img>` в строку;
- `none` — содержимое блока не отображается.

Различные варианты использования атрибута `display` приведены в листинге 2.8.

Листинг 2.8. Атрибут `display`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Атрибут display</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
  div div { border: 2px solid #333 }
  label { display: inline-block; width: 100px }
 </style>
</head>
<body>
 <h2>Различные типы блоков</h2>
 <div>
   <div style="display: inline">display = inline</div>
   <div style="display: inline; width: 300px">display = inline</div>
   <div style="display: inline-block; width: 300px">
    display = inline-block
   </div>
  <div style="display: block">display = block</div>
  <div style="display: none">Этого блока не видно</div>
 </div>
 <h2>Выравнивание элементов формы</h2>
 <div><label for="login">Логин:</label>
  <input type="text" name="login" id="login"></div>
 <div><label for="pass">Пароль:</label>
  <input type="password" name="pass" id="pass"></div>
 <h2>Указание типа блока для ссылки</h2>
 <div>
  <div style="width: 300px"><a href="link1.html">Обычная ссылка</a></div>
  <div style="width: 300px">
   <a href="link.html" style="display: block">Ссылка занимает всю
   ширину блока</a>
```

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

2.12.2. Установка размеров

Атрибуты `width` и `height` задают соответственно ширину и высоту блока:

```
div { width: 100px; height: 100px }
```

Атрибуты `min-width` и `min-height` задают соответственно минимальные ширину и высоту блока:

```
div { min-width: 100px; min-height: 100px }
```

> ***Примечание***
>
> Web-браузер Internet Explorer поддерживает атрибуты `min-width` и `min-height`, начиная с версии 7.0.

Атрибуты `max-width` и `max-height` задают соответственно максимальную ширину и высоту блока:

```
div { max-width: 100px; max-height: 100px }
```

> ***Примечание***
>
> Web-браузер Internet Explorer поддерживает атрибуты `max-width` и `max-height`, начиная с версии 7.0.

Если размеры не заданы, то блок займет всю ширину родительского элемента, а его высота будет определена по содержимому блока. Если содержимое не помещается в блок заданного размера, то он будет отображаться в соответствии со значением атрибута `overflow`.

2.12.3. Атрибут *overflow*

Атрибут `overflow` задает поведение блока, чье содержимое выходит за его границы. Может принимать следующие значения:

- `visible` — блок увеличивается в размерах так, чтобы все его содержимое отобразилось полностью (значение по умолчанию). Если размеры заданы явным образом, то содержимое будет выходить за границы блока, а размеры самого блока останутся прежними;
- `hidden` — то, что не помещается в блоке, скрывается;
- `scroll` — у блока в любом случае отображаются полосы прокрутки;
- `auto` — если содержимое не помещается в блок, то добавляются полосы прокрутки. Если же содержимое полностью помещается, то полосы не отображаются.

Различные варианты использования атрибута `overflow` приведены в листинге 2.9.

Листинг 2.9. Атрибут `overflow`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Атрибут overflow</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
   body { font-size: 14px; font-family: Arial; color: black  }
   .div1 div { width: 100px; height: 100px }
   .div1 div, .div2 div {
     background-color: silver;
     border: black 2px solid;
     margin-bottom: 10px
   }
   .div2 { height: 600px }
   span { font-weight: bold }
 </style>
</head>
<body>
 <div class="div1">
  <span>overflow = hidden</span><br>
  <div style="overflow: hidden">
  этооченьдлиннаястрокабезпробелов
  То, что не влезает в границы блока, скрывается
  </div>
  <span>overflow = scroll</span><br>
  <div style="overflow: scroll">
  этооченьдлиннаястрокабезпробелов
  overflow = scroll. У блока в любом случае отображаются полосы прокрутки
  </div>
  <span>overflow = auto</span><br>
  <div style="overflow: auto">
  overflow = auto
  </div>
  <span>overflow = auto</span><br>
  <div style="overflow: auto">
  этооченьдлиннаястрокабезпробелов
  overflow = auto. Если содержимое не помещается в блок, то добавляются
  полосы прокрутки
  </div>
 </div>
 <div class="div2">
  <span>overflow = visible. Высота не задана</span><br>
  <div style="overflow: visible; width: 100px;">
  этооченьдлиннаястрокабезпробелов
```

```
  overflow = visible. Блок расширяется так, чтобы все его содержимое
  отобразилось полностью
  </div>
  <span>overflow = visible. Высота задана</span><br>
  <div style="overflow: visible; width: 100px; height: 100px;">
  этооченьдлиннаястрокабезпробелов
  overflow = visible. Если размеры заданы, то содержимое будет выходить
  за границы блока
  </div>
 </div>
</body>
</html>
```

2.12.4. Управление обтеканием

Атрибут `float` определяет, по какой стороне производится выравнивание блока. Может принимать следующие значения:

- `left` — блок выравнивается по левой стороне, а другие элементы обтекают его справа;
- `right` — блок выравнивается по правой стороне, а другие элементы обтекают его слева;
- `none` — выравнивание отсутствует.

Атрибут `clear` разрешает или запрещает обтекание. Может принимать следующие значения:

- `both` — запрещает обтекание по обеим сторонам;
- `left` — запрещает обтекание по левой стороне;
- `right` — запрещает обтекание по правой стороне;
- `none` — отменяет запрет на обтекание, который был установлен с помощью значений `both`, `left` и `right`.

Очень часто атрибуты `float` и `clear` применяются в блочной верстке Web-страницы для создания колонок. Рассмотрим это на примере (листинг 2.10).

Листинг 2.10. Блочная верстка страницы с помощью `float`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Блочная верстка страницы с помощью float</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
  * { margin: 0; padding: 0 } /* Убираем все отступы */
  body { font-family: Verdana, Tahoma, sans-serif; font-size: 14px }
```

```
  .header { background-color: silver; padding: 10px; height: 50px;
            text-align: center; line-height: 50px; }
  .menu { float: left; border: 1px solid black; width: 150px;
          padding: 5px; margin: 10px; min-height: 200px }
  .text { border: 1px solid black; padding: 5px;
           margin: 10px 10px 10px 185px; min-height: 500px }
  .footer { background-color: silver; padding: 5px; clear: both;
             height: 30px; line-height: 30px; }
 </style>
</head>
<body>
 <div class="header"><h2>Заголовок</h2></div>
 <div class="menu">Панель навигации</div>
 <div class="text">
  <h2>Основное содержимое страницы</h2>
  <p>Какой-то текст</p>
 </div>
 <div class="footer">Информация об авторских правах</div>
</body>
</html>
```

Итак, Web-страница состоит из четырех блоков. Первый блок (`header`) содержит заголовок и занимает всю ширину окна. Второй блок (`menu`) предназначен для вывода панели навигации. Для этого блока указано выравнивание по левой стороне и обтекание справа. В третьем блоке (`text`) располагается основное содержимое Web-страницы. Этот блок занимает всю ширину окна после панели навигации. Если изменить размер окна, то блок будет или расширяться, или уменьшаться. Четвертый блок (`footer`) предназначен для вывода информации об авторских правах, а также различных логотипов и счетчиков. Для этого блока с помощью атрибута `clear` отменяется обтекание с обеих сторон.

У блочной верстки Web-страницы с помощью `float` есть один недостаток. Если уменьшить ширину окна Web-браузера, то блоки будут выстроены по вертикали, один под другим. Чтобы избежать этого эффекта мы указали внешний отступ слева (185 px). Благодаря этому блоки всегда будут расположены по горизонтали, независимо от ширины окна Web-браузера.

Следует обратить внимание еще на один момент: содержимому тега `<div>` нельзя задать вертикальное выравнивание с помощью атрибута `vertical-align`. Чтобы добиться центрирования по вертикали, мы указали значение атрибута `line-height` равным высоте блока.

2.12.5. Позиционирование блока

Атрибут `position` позволяет задать способ позиционирования блока. Он может принимать одно из четырех значений:

- `static` — статическое позиционирование (значение по умолчанию). Положение элемента в окне Web-браузера определяется его положением в тексте HTML-документа;
- `relative` — относительное позиционирование. Координаты отсчитываются относительно позиции, в которую Web-браузер поместил бы элемент, будь он статически позиционированным;
- `absolute` — абсолютное позиционирование. Координаты отсчитываются относительно левого верхнего угла ближайшего родительского элемента, который имеет позиционирование, отличное от статического;
- `fixed` — фиксированное позиционирование. Координаты отсчитываются относительно левого верхнего угла окна Web-браузера. При прокрутке содержимого окна блок не смещается.

> ***Примечание***
>
> Web-браузер Internet Explorer поддерживает атрибут `fixed`, начиная с версии 7.0.

Для указания привязки предназначены следующие атрибуты:

- `left` — расстояние от левой границы;
- `top` — расстояние от верхней границы;
- `right` — расстояние от правой границы;
- `bottom` — расстояние от нижней границы.

Эти атрибуты могут иметь отрицательные значения. Статически позиционированные элементы не имеют атрибутов `left`, `top`, `right` и `bottom`.

Давайте рассмотрим все это на примере (листинг 2.11).

Листинг 2.11. Позиционирование блоков

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Позиционирование блоков</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
  body { font-family: Verdana, Tahoma, sans-serif; font-size: 14px }
  div { border: 1px solid black }
  .div1 { width: 10px; height: 2000px; left: 900px; top: 0;
          position: absolute }
  .div2 { height: 20px; position: static; background-color: silver }
  .div3 { height: 20px; position: relative; top: 30px;
          background-color: silver }
  .div4 { width: 150px; height: 150px; position: absolute; top: 30px;
          left: 400px; background-color: green }
  .div5 { width: 300px; height: 300px; position: absolute; top: 250px;
          left: 400px; }
```

```
  .div6 { width: 100px; height: 100px; position: absolute; top: 50px;
          left: 50px; background-color: #F5D8C1 }
  .div7 { width: 150px; height: 300px; position: fixed; top: 150px;
          left: 20px; background-color: #FF9600 }
  .div8 { width: 100%; height: 50px; left: 0; bottom: 0; margin: 0;
          position: fixed; background-color: #F4AB56 }
 </style>
</head>
<body>
 <div class="div1"></div>
 <div class="div2">Статическое позиционирование</div>
 <div class="div3">Относительное позиционирование</div>
 <div class="div4">Абсолютное позиционирование</div>
 <div class="div5">Абсолютное позиционирование внутри родительского блока
  <div class="div6">top: 50px; left: 50px;</div>
 </div>
 <div class="div7">Фиксированное позиционирование</div>
 <div class="div8">Фиксированное позиционирование относительно
                   нижней границы</div>
</body>
</html>
```

Итак, на странице восемь блоков.

Блок `div1` имеет абсолютное позиционирование и смещен на 900 px относительно левой границы окна Web-браузера. Для блока также указана большая высота (2000 px). Это позволит увидеть эффект фиксированного позиционирования для блоков `div7` и `div8`, т. к. Web-браузер отобразит вертикальную полосу прокрутки.

Блок `div2` имеет статическое позиционирование, а блок `div3` — относительное. Блок `div3` сдвинут на 30 px вниз относительно блока `div2`, а не от верхней границы окна Web-браузера.

Блоки `div4`, `div5` и `div6` имеют абсолютное позиционирование. Блок `div4` сдвинут на 400 px относительно левой границы окна Web-браузера и на 30 px — относительно верхней. Внутри блока `div5` расположен блок `div6`. Смещения этого блока отсчитываются относительно границ блока `div5`, а не границ окна Web-браузера.

Блоки `div7` и `div8` имеют фиксированное позиционирование. Блок `div7` демонстрирует возможность размещения панели навигации в определенном месте, а блок `div8` — прикрепление блока к нижней границе окна Web-браузера. Чтобы увидеть отличие от других видов позиционирования, переместите вертикальную полосу прокрутки вниз. Эти блоки всегда остаются на своих местах и не перемещаются при прокрутке. Однако не следует забывать, что Web-браузер Internet Explorer поддерживает атрибут `fixed`, начиная с версии 7.0.

2.12.6. Последовательность отображения слоев

Атрибут `z-index` устанавливает порядок, в котором свободно позиционированные элементы будут перекрываться друг другом. Элемент с бóльшим значением `z-index` перекрывает элементы с меньшим значением. Значение у родителя равно нулю.

Рассмотрим порядок перекрытия на примере, а заодно приведем вариант блочной верстки с использованием абсолютного позиционирования для панели навигации (листинг 2.12).

Листинг 2.12. Атрибут `z-index`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Атрибут z-index</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
  * { margin: 0; padding: 0 } /* Убираем все отступы */
  body { font-family: Verdana, Tahoma, sans-serif; font-size: 14px }
  .header { background-color: silver; padding: 10px; height: 50px;
            text-align: center; line-height: 50px; }
  .menu { border: 1px solid black; width: 150px; height: 200px;
          margin: 0; padding: 5px; position: absolute;
          left: 10px; top: 80px }
  .text { border: 1px solid black; padding: 5px;
          margin: 10px 10px 10px 185px; min-height: 500px }
  .footer { background-color: silver; padding: 5px; height: 30px;
            line-height: 30px }
  .div1 { width: 100px; height: 100px; position: absolute; top: 70px;
          left: 50px; z-index: 1; background-color: silver }
  .div2 { width: 100px; height: 100px; position: absolute; top: 120px;
          left: 100px; z-index: 2; background-color: red }
  .div3 { width: 100px; height: 100px; position: absolute; top: 170px;
          left: 150px; z-index: 3; background-color: green }
 </style>
</head>
<body>
 <div class="header"><h2>Заголовок</h2></div>
 <div class="text">
  <h2>Основное содержимое страницы</h2>
  <p>Какой-то текст</p>
  <div style="position: relative; top: 30px">
   <div class="div1">z-index = 1</div>
   <div class="div2">z-index = 2</div>
   <div class="div3">z-index = 3</div>
```

```
  </div>
 </div>
 <div class="footer">Информация об авторских правах</div>
 <div class="menu">Панель навигации с абсолютным позиционированием</div>
</body>
</html>
```

2.13. Управление отображением элемента

Атрибут `visibility` задает видимость элемента в окне Web-браузера. Он может принимать следующие значения:

- `inherit` — если родитель видим, то видим и элемент (значение по умолчанию);
- `visible` — элемент отображается независимо от видимости родителя;
- `hidden` — элемент скрывается независимо от видимости родителя.

Невидимый элемент все равно занимает место на Web-странице. Для того чтобы скрыть элемент и убрать его с Web-страницы, можно воспользоваться атрибутом `display` со значением `none`.

Атрибуты могут задавать только начальное поведение элемента. Отобразить же скрытый элемент можно только с помощью языка программирования JavaScript. Рассмотрим пример употребления атрибутов `visibility` и `display`, а заодно познакомимся с использованием языка программирования JavaScript (листинг 2.13).

Листинг 2.13. Скрытие и отображение элементов

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Скрытие и отображение элементов</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <script type="text/javascript">
 <!--
 function ChangeAbz1() {
   abz = document.getElementById("abz1");
   if (abz.style.display=="none") {
      abz.style.display = "block";
   }
   else {
      abz.style.display = "none";
   }
 }
 function ChangeAbz2() {
   abz = document.getElementById("abz2");
```

```
    if (abz.style.visibility=="hidden") {
       abz.style.visibility = "visible";
    }
    else {
       abz.style.visibility = "hidden";
    }
 }
 //-->
 </script>
</head>
<body>
 <div><a href="#" onclick="ChangeAbz1(); return false">Щелкните на
 ссылке, чтобы отобразить или скрыть абзац</a></div>
 <p id="abz1" style="display: none; background-color: silver">Скрытый
 абзац</p>
 <p>Демонстрация применения атрибута display.</p>
 <div><a href="#" onclick="ChangeAbz2(); return false">Щелкните
 на ссылке, чтобы отобразить или скрыть абзац</a></div>
 <p id="abz2" style="visibility: hidden; background-color: silver">Скрытый
 абзац</p>
 <p>Демонстрация применения атрибута visibility.</p>
</body>
</html>
```

Итак, первая ссылка демонстрирует применение атрибута `display`. При щелчке на ссылке отображается скрытый абзац, и все содержимое страницы сдвигается вниз. При повторном щелчке абзац скрывается, и все содержимое страницы сдвигается вверх на место скрытого абзаца.

Вторая ссылка демонстрирует применение атрибута `visibility`. При щелчке на ссылке скрытый абзац отображается, а при повторном щелчке — скрывается. Но в отличие от того, что происходит при изменении атрибута `display`, в этом случае все остальное содержимое страницы остается на своих первоначальных местах.

2.14. CSS 3

Как и HTML, язык CSS продолжает развиваться. Его очередная версия, носящая номер 3, в данный момент находится в разработке. Рабочая редакция CSS 3 предлагает множество впечатляющих новых возможностей в плане оформления страниц и в той или иной степени уже поддерживается всеми современными Web-обозревателями.

Давайте перечислим версии Web-обозревателей, в которых появилась поддержка HTML 5.

- Microsoft Internet Explorer — 9;
- Mozilla Firefox — 4.0;

- Google Chrome — 10.0;
- Opera — 10.5;
- Apple Safari — 5.1.

Если далее мы не укажем список Web-обозревателей и их версий, подразумевается, что описываемая возможность CSS 3 поддерживается версиями, перечисленными выше, и более поздними. Иначе мы явно укажем, в каких версиях каких программ поддерживается рассматриваемая возможность.

2.14.1. Новые селекторы

CSS 3 предоставляет новые селекторы, которые позволяют привязать стили к элементам страниц.

- `Селектор1 ~ Селектор2` — элемент, соответствующий параметру `Селектор2`, который является соседним для элемента, соответствующего параметру `Селектор1`, и следует после него, причем необязательно непосредственно:

  ```
  div ~ h6 { color: red }
  ```

  Цвет текста заголовка станет красным, если тег `<h6>` следует за элементом `<div>` и, возможно, отделяется от него другими элементами:

  ```
  <div>Текст</div>
  <p>Тоже текст</p>
  <h6>Красный заголовок</h6>
  ```

- `:empty` — пустой элемент, не имеющий никакого содержимого.

  ```
  p:empty { display: none; }
  ```

  Делаем все пустые абзацы невидимыми.

- `:first-child` — привяжет стиль к элементу, только если он относится к указанному типу и является первым элементом в контейнере.

  ```
  p:first-child { font-size: larger }
  ```

  Увеличиваем размер шрифта у абзацев, которые являются первыми элементами в контейнере:

  ```
  <div>
    <p>Этот абзац будет выведен увеличенным шрифтом, поскольку он
    является первым элементом в контейнере.</p>
    <p>А этот будет выведен шрифтом стандартного размера, поскольку он
    не является первым элементом в контейнере.</p>
  </div>
  <div>
    <h1>Заголовок</h1>
    <p>Этот абзац будет выведен шрифтом стандартного размера,
    поскольку он не является первым элементом в контейнере.</p>
  </div>
  ```

- `:first-of-type` — задает стиль для первого элемента в контейнере, относящегося к заданному типу.

```
p:first-of-type { font-size: larger }
```

Увеличиваем размер шрифта у всех абзацев — первых потомков контейнеров:

```
<div>
  <p>Этот абзац будет выведен увеличенным шрифтом, поскольку он
  является первым абзацем в контейнере.</p>
  <p>А этот будет выведен шрифтом стандартного размера.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>Этот абзац будет выведен увеличенным шрифтом, поскольку он
  является первым абзацем в контейнере.</p>
</div>
```

- `:last-child` — привяжет стиль к элементу, только если он относится к указанному типу и является последним элементом в контейнере.
- `:last-of-type` — задает стиль для последнего элемента в контейнере, относящегося к заданному тину.
- `:nth-child(<n>)` — привяжет стиль к элементу, только если он относится к указанному типу и является точно *n*-м по счету элементом в контейнере.

```
p:nth-child(2) { font-size: larger }
```

Увеличиваем размер шрифта у абзацев, которые являются вторыми потомками контейнеров:

```
<div>
  <p>Этот абзац будет выведен шрифтом стандартного размера.</p>
  <p>А этот будет выведен увеличенным шрифтом.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>Этот абзац также будет выведен увеличенным шрифтом.</p>
</div>
```

- `:nth-last-child(<n>)` — привяжет стиль к элементу, только если он относится к указанному типу и является точно *n*-м по счету элементом в контейнере, считая с конца.
- `:nth-of-type(<n>)` — задает стиль для *n*-го элемента в контейнере, относящегося к указанному типу.

```
p:nth-of-type(2) { font-size: larger }
```

Увеличиваем размер шрифта у каждого второго абзаца в контейнере:

```
<div>
  <p>Этот абзац будет выведен шрифтом стандартного размера.</p>
```

```
  <p>А этот будет выведен увеличенным шрифтом.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>Этот абзац будет выведен шрифтом стандартного размера.</p>
  <p>Этот абзац будет выведен увеличенным шрифтом.</p>
</div>
```

- `:nth-last-of-type(<n>)` — задает стиль для *n*-го элемента указанного типа в контейнере, считая с конца.
- `:only-child` — привяжет стиль к элементу, только если он относится к указанному типу и является единственным элементом в контейнере.

```
p:only-child { font-size: larger }
```

Увеличиваем размер шрифта у абзаца, который является единственным элементом в контейнере:

```
<div>
  <p>Этот абзац будет выведен увеличенным шрифтом.</p>
</div>
<div>
  <p>А этот будет выведен шрифтом стандартного размера.</p>
  <p>И этот тоже.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>И этот абзац будет выведен шрифтом стандартного размера.</p>
</div>
```

- `:only-of-type` — задает стиль для единственного элемента указанного типа в контейнере.

```
p:only-of-type { font-size: larger }
```

Увеличиваем размер шрифта у каждого единственного абзаца в контейнере; при этом там могут находиться элементы других типов:

```
<div>
  <p>Этот абзац будет выведен увеличенным шрифтом.</p>
</div>
<div>
  <p>А этот будет выведен шрифтом стандартного размера.</p>
  <p>И этот тоже.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>Этот абзац будет выведен увеличенным шрифтом.</p>
</div>
```

- `:target` — активный указатель (на который был выполнен переход щелчком на внутренней гиперссылке).
- `:checked` — установленный флажок или переключатель.
- `:disabled` — элемент управления, недоступный для пользователя.

```
input:disabled, select:disabled, textarea:disabled { color: #cccccc }
```

Задаем для текста недоступных элементов управления серый цвет.

- `:enabled` — элемент управления, доступный для пользователя.
- `:focus` — гиперссылка или элемент управления, имеющий фокус ввода.
- `:hover` — любой элемент, на который наведен курсор мыши.
- `:in-range` — поле ввода числового значения или регулятор, в котором указано корректное значение, укладывающееся в заданный диапазон.
- `:invalid` — элемент управления, в котором указано некорректное значение.

```
input:invalid { color: red }
```

- `:optional` — необязательный элемент управления (тег которого не имеет параметра `required`).
- `:out-of-range` — поле ввода числового значения или регулятор, в котором указано некорректное значение, не укладывающееся в заданный диапазон.
- `:read-only` — элемент управления, доступный только для чтения.

```
input:read-only { background-color: #cccccc }
```

- `:read-write` — элемент управления, доступный только для чтения и записи.
- `:required` — обязательный элемент управления (тег которого имеет параметр `required`).
- `:valid` — элемент управления, в котором указано корректное значение.

Псевдокласс `:not` стоит особняком. Он позволяет привязать стиль к любому элементу страницы, не удовлетворяющему заданным условиям. Вот формат записи этого псевдокласса:

<основной селектор>`:not(`*<селектор выбора>*`)`

Стиль будет привязан к элементу Web-страницы, удовлетворяющему *основному селектору* и не удовлетворяющему *селектору выбора*.

Пример:

```
p:not(:first-child) { font-size: larger }
```

Этот стиль будет применен лишь к абзацам, не являющимся первыми элементами в контейнерах.

2.14.2. Новые единицы измерения размеров и способы задания цвета

CSS 3 предлагает три новые единицы измерения размеров:

- `rem` — размер шрифта, указанный для корневого элемента (тега `<html>`);
- `vw` — 1% от ширины клиентской области окна Web-обозревателя;
- `vh` — 1% от высоты клиентской области окна Web-обозревателя.

Еще мы можем указать в качестве значения атрибута стиля результат какого-либо вычисления. Для этого предусмотрена функция `calc`, поддерживаемая CSS. Само выражение, которое требуется вычислить, записывается сразу же после ее имени, берется в скобки и имеет вид обычной алгебраической формулы:

```
#main { width: calc(100wh - 400px) }
```

Этот стиль устанавливает для контейнера с идентификатором `main` ширину, равную ширине клиентской области окна Web-обозревателя за вычетом 400 пикселов.

В выражениях, указанных в функции `calc`, допустимы следующие операции: сложение (обозначается символом `+`), вычитание (`-`), умножение (`*`) и деление (`/`).

Для указания цвета мы можем пользоваться расширенным форматом `rgba([R], [G], [B], [A])`, где `A` — уровень прозрачности цвета, представляющий собой значение от 0,0 (цвет полностью прозрачен) до 1,0 (цвет полностью непрозрачен). Например, `rgba(255, 0, 0, 0.5)` задает полупрозрачный красный цвет.

2.14.3. Параметры фона

Мы имеем возможность задать новые параметры фона, в частности, указать размеры фонового изображения, режим его позиционирования и заполнения им элемента страницы.

Размеры фонового изображения

Атрибут `background-size` указывает размеры фонового изображения в виде двух значений, разделенных пробелом; первое значение задает ширину изображения, второе — высоту.

Растягиваем фоновое изображение на всю ширину страницы и на половину ее высоты:

```
body { background-size: 100% 50% }
```

Также мы можем использовать следующие значения:

- `auto` — не изменяет размеры фонового изображения (поведение по умолчанию);
- `cover` — задает такие размеры фонового изображения, чтобы оно полностью покрывало элемент страницы; при этом какие-то фрагменты изображения могут выйти за границы элемента;

- `contain` — задает такие размеры фонового изображения, чтобы оно полностью помещалось в элементе страницы, занимая его целиком; при этом какие-то части элемента могут быть не покрыты изображением.

Пример:

```
body { background-image: url("/images.background.jpg");
background-repeat: no-repeat; background-size: cover }
```

Заполняем фоновым изображением все пространство страницы.

Режим позиционирования фонового изображения

Если мы задали местоположение фонового изображения, то можем также указать, относительно чего оно будет позиционироваться. Для этого мы применим атрибут `background-origin`, задав одно из следующих его значений:

- `padding-box` — изображение будет позиционироваться относительно границ внутренних отступов (поведение по умолчанию);
- `border-box` — позиционирование фонового изображения будет выполняться относительно рамки элемента;
- `content-box` — изображение будет позиционироваться относительно границ содержимого элемента.

Позиционируем фоновое изображение относительно содержимого контейнера с идентификатором `main`:

```
#main { background-position: center top; background-repeat: no-repeat;
background-origin: content-box }
```

Режим заполнения для фонового изображения

Иногда полезно указать, какую часть элемента будет заполнять фоновое изображения. Этим занимается атрибут `background-clip`, поддерживающий три значения:

- `content-box` — фоном будет заполнена часть элемента, занятая его содержимым;
- `padding-box` — фоном будет заполнена часть элемента, занятая содержимым и впутренними отступами;
- `border-box` — фоном будет заполнен весь элемент (поведение по умолчанию).

Заполняем черным фоном лишь ту часть контейнера `main`, что занята собственно содержимым:

```
#main { color: white; background-color: black;
background-clip: content-box }
```

2.14.4. Рамки со скругленными углами

Одна из наиболее долгожданных новых возможностей CSS 3 — скругление углов рамок, создаваемых у элементов страниц.

Задание радиуса скругления для разных углов по отдельности

Радиус скругления углов указывается с помощью атрибутов `border-top-left-radius` (левый верхний угол), `border-top-right-radius` (правый верхний угол), `border-bottom-right-radius` (правый нижний) и `border-bottom-left-radius` (левый нижний).

Если задать два значения раднуса скругления, то первое будет применено к горизонтальной четверти угла, а второе — к вертикальной четверти. Если задано одно значение, то оно будет применено к обеим четвертям.

Пример:

```
td, th { border-top-left-radius: 2px 1px;
border-top-right-radius: 2px; border-bottom-right-radius: 2px;
border-bottom-left-radius: 2px 1px }
```

Задаем для ячеек таблиц рамки со скругленными углами. У левого верхнего и левого нижнего углов рамок радиус скругления горизонтальной четверти равен двум пикселам, а раднус скругления вертикальной четверти — одному пикселу; у правого верхнего и правого нижнего углов раднус скругления обеих четвертей равен двум пикселам.

Задание радиуса скругления сразу для всех углов

Указать радиус скругления сразу для всех углов рамки можно атрибутом `border-radius`:

```
border-radius: <левый верхний> <правый верхний> <правый нижний>
<левый нижний> / <левый верхний> <правый верхний> <правый нижний>
<левый нижний>
```

Перед символом слэша задаются раднусы скругления для горизонтальной четверти угла, а после него — радиусы скругления для вертикальной четверти. Если последние не указать, заданные радиусы скругления будут применены и к горизонтальной, и к вертикальной четвертям.

Примеры:

```
td, th { border-radius: 2px 2px 2px 2px / 1px 2px 2px 1px }
```

Для углов с одинаковым радиусом скругления предусмотрен сокращенный формат записи этого атрибута:

```
td, th { border-radius: 2px / 1px }
```

Задаем для всех углов рамки радиус скругления горизонтальной четверти в два пиксела и радиус скругления вертикальной четверти в один пиксел.

```
td, th { border-radius: 2px }
```

А теперь рамка будет иметь углы с радиусом скругления в два пиксела в обеих четвертях.

2.14.5. Параметры таблиц

Для оформления таблиц CSS 3 предоставляет два новых атрибута. С их помощью мы можем указать величину просвета между ячейками и режим рисования рамок.

Просвет между ячейками

Поскольку параметр `cellspacing` тега `<table>` объявлен в HTML 5 устаревшим, величину просвета между ячейками следует задавать с помощью атрибута `border-spacing`.

- Если указаны два значения, первое задаст величину просвета по горизонтали, а второе — по вертикали:

```
table.special { border-spacing: 2px 3px }
```

- Если указано одно значение, оно задаст величину просвета со всех сторон:

```
table { border-spacing: 1px }
```

Режим рисования рамок

Атрибут `border-collapse` позволяет установить режим рисования рамок для самой таблицы и для ее ячеек. Поддерживаются два значения:

- `separate` — рисуется рамка вокруг самой таблицы и рамки вокруг каждой из ее ячеек, в результате каждая ячейка фактически будет заключена в двойную рамку (поведение по умолчанию);
- `collapse` — рисуются лишь рамки, разделяющие ячейки; тогда каждая ячейка таблицы будет заключена в одинарную рамку.

Пример:

```
table.minimalistic { border-collapse: collapse }
```

2.14.6. Параметры тени

Не менее долгожданной является возможность создания тени у текста или даже у целого контейнера.

Параметры тени у текста

Параметры тени для текста задает атрибут `text-shadow`:

```
text-shadow: <горизонтальное смещение> <вертикальное смещение>
<радиус размытия> <цвет>
```

Горизонтальное смещение тени может быть положительным (тогда тень будет расположена правее текста), отрицательным (левее) или пулевым (тень будет располагаться прямо под текстом). *Вертикальное смещение* также может быть положительным (тень будет расположена ниже текста), отрицательным (выше) или нулевым. Нулевое смещение имеет смысл только в том случае, если для тени задано размытие.

Если *радиус размытия* не указан или его значение равно нулю, тень не будет иметь эффекта размытия.

Значение none убирает тень у текста; это поведение по умолчанию.

Примеры:

```
h2 { text-shadow: 1mm 1mm rgba(0, 0, 0, 0.2) }
```

Здесь мы задали для заголовков второго уровня черную тень с уровнем прозрачности 0,2, расположенную правее и ниже текста на 1 мм и не имеющую эффекта размытия.

```
h1 { text-shadow: 0mm 0mm 10px red }
```

А для заголовков первого уровня задаем красную тень, расположенную непосредственно под текстом и имеющую раднус размытия 10 пикселов.

Атрибут text-shadow поддерживается Internet Explorer, начиная с версии 10.

Параметры тени у контейнера

Чтобы указать тень у контейнера, мы используем атрибут box-shadow:

```
box-shadow: <горизонтальное смещение> <вертикальное смещение>
<радиус размытия> <расширение> <цвет> inset
```

Расширение задает величину, на которую тень увеличится относительно своих изначальных размеров (они равны размерам блочного элемента, для которого она создается); если этот параметр не указан, расширение будет равно пулю. Если указать слово inset, будет создана "впутренняя" тень, визуально находящаяся впутри контейнера. Назначение остальных параметров нам уже знакомо.

Примеры:

```
h2 { box-shadow: 1px 1px 2px 1px rgba(0, 0, 0, 0.2) }
h1 { box-shadow: 1px 1px 0px 10px red inset }
```

2.14.7. Загружаемые шрифты

Помимо шрифтов, установленных на компьютере, и типовых семейств шрифтов, мы можем использовать для вывода текста на страницах *загружаемые шрифты*. Такой шрифт хранится в виде файла в составе сайта и загружается самим Web-обозревателем.

Загружаемый шрифт указывается с помощью директивы CSS под названием @font-face:

```
@font-face {
  font-family: <имя, под которым загружаемый шрифт будет доступен в
  таблице стилей>;
  src: url(<интернет-адрес файла шрифта>)
}
```

Интернет-адрес файла шрифта берется в кавычки.

Пример:

```
@font-face {
  font-family: MyriadPro;
  src: url("/fonts/myriadpro.woff")
}
```

Загружаем шрифт, хранящийся в файле myriadpro.woff, который находится в папке fonts корневой папки сайта, и задаем для него имя `MyriadPro`.

После этого мы можем использовать загруженный шрифт где угодно, указав заданное для него имя:

```
p { font-family: MyriadPro }
```

Все современные Web-обозреватели поддерживают загружаемые шрифты, сохраненные в формате WOFF. С другими форматами лучше не связываться, т. к. они либо поддерживаются не всеми Web-обозревателями, либо при их использовании могут возникнуть проблемы.

> ***Примечание***
>
> Для преобразования шрифтов в нужный формат можно воспользоваться онлайновым сервисом-перекодировщиком Font2Web, находящимся по интернет-адресу **http://www.font2web.com/**.

2.14.8. Режимы установки размеров

Еще мы можем, воспользовавшись средствами CSS 3, задать режимы установки размеров для контейнеров и таблиц.

Режим установки размеров для контейнеров

Атрибут `box-sizing` позволяет указать режим, в котором будут задаваться размеры контейнера. Здесь доступны два значения:

- `content-box` — размеры будут задаваться для содержимого контейнера без учета его внутренних отступов и рамки, как было показано в *разд. 2.12* (это поведение по умолчанию);
- `border-box` — размеры будут задаваться для самого контейнера с учетом его внутренних отступов и рамки.

Пример:

```
#main { width: 400px; padding: 5px; box-sizing: border-box }
```

Теперь контейнер `main` получит ширину в 400 пикселов, а его содержимое будет иметь ширину в 390 (400 – 2×5) пикселов.

Атрибут `box-sizing` поддерживается Firefox, начиная с версии 29.0.

Режим установки размеров для таблиц

Атрибут `table-layout` указывает режим задания размеров у таблиц. Здесь также доступны два значения:

- `auto` — заданные нами значения ширины таблицы и ее ячеек суть лишь рекомендация для Web-обозревателя. Он может изменить ширину таблицы и ее ячеек, если содержимое в них не помещается. Это поведение по умолчанию;
- `fixed` — ширина таблицы и ее ячеек ни в коем случае изменяться не будет. Если содержимое не помещается в ячейках, возникнет переполнение, параметры которого мы можем задавать с помощью атрибута `overflow`.

Пример:

```
table { width: 500px; overflow: auto; table-layout: fixed }
```

Теперь ширина таблицы не будет изменяться ни в коем случае.

2.14.9. Градиентные фоны

Градиент — это изображение, полученное в результате плавного перетекания одного цвета в другой. Как пример градиента можно привести заливку заголовков окон в стандартной теме Windows XP, Vista, 7 — там синий цвет плавно перетекает в светло-голубой.

Введение в градиенты

Градиенты бывают двух видов:

- В *линейном градиенте* переходящие друг в друга цвета распространяются по прямой из одной точки в другую. Точка, в которой градиент начинается, называется *начальной*, а точка, в которой он заканчивается, — *конечной*.
- В *радиальном градиенте* переходящие друг в друга цвета распространяются из начальной точки во все стороны, в конечном счете образуя своего рода эллипс (или, в его "вырожденном" случае, круг). За конечную точку такого градиента принимается любая точка, расположенная на охватывающем его воображаемом эллипсе.

Первое, что нам нужно сделать для создания градиента, — указать местоположение его начальной и конечной точек в координатах элемента, в качестве фона которого будет выступать этот градиент. Как вариант мы можем указать местоположение начальной точки и размер градиента.

Далее мы должны создать набор ключевых точек. *Ключевая точка* располагается на воображаемой прямой, проведенной между начальной и конечной точками градиента, и определяет, какой "чистый" цвет должен присутствовать в этом месте. (При этом цвета, которые будут присутствовать между ключевыми точками, сформируются переходом от одного "чистого" цвета в другой.) Ключевая точка характеризуется расстоянием между ней и начальной точкой градиента и, собственно, значением цвета.

Любой градиент должен иметь, по крайней мере, две ключевые точки, которые часто располагаются в его начальной и конечной точках. Более сложные градиенты имеют более двух ключевых точек; при этом цвет, указанный в первой ключевой точке, будет плавно перетекать в тот, что указан во второй ключевой точке, потом в цвет из третьей ключевой точки и т. д.

На данный момент CSS 3 позволяет создавать лишь градиентные фоны — градиенты, которые служат фоновыми заливками. Градиентный фон указывают либо с помощью атрибута `background-image`, либо в составе атрибута `background`.

Создание линейных градиентов

Линейный градиент формируется с применением поддерживаемой CSS функции `linear-gradient` (листинг 2.14).

Листинг 2.14. Описание линейного градиента

```
linear-gradient(
  [to <направление> ,]
  <цвет в ключевой точке 1> <местоположение ключевой точки 1>,
  <цвет в ключевой точке 2> <местоположение ключевой точки 2>,
  . . .
  <цвет в ключевой точке n-1> <местоположение ключевой точки n-1>,
  <цвет в ключевой точке n> <местоположение ключевой точки n>
)
```

Направление, по которому будет распространяться градиент, можно указать тремя способами:

- С помощью одного из значений `top` (вверх), `bottom` (вниз), `left` (налево) и `right` (направо). В этом случае градиент будет распространяться в указанном направлении по горизонтали или вертикали.
- С помощью комбинации из двух перечисленных выше значений, разделенных пробелом. В этом случае градиент будет распространяться по диагонали, в угол элемента, образованный смыканием указанных сторон.
- В виде значения угла между горизонталью и линией, по которой должен распространяться градиент; этот угол отсчитывается по часовой стрелке. Если требуется отсчитывать угол против часовой стрелки, нужно указать отрицательное значение.

  Для указания значения угла CSS 3 предлагает следующие единицы измерения: `deg` (градусы), `rad` (радианы), `grad` (грады) и `turn` (повороты).

Если направление не указано, градиент будет распространяться сверху вниз (как будто для него было задано значение направления `bottom`).

Местоположение ключевой точки обычно указывают в процентах, чтобы связать размеры градиента с размерами элемента, фоном которого он станет.

Если местоположение первой ключевой точки градиента не указано, она будет находиться в начальной точке; если не указать местоположение последней ключевой точки, она будет находиться в конечной точке. А если не указывать местоположения промежуточных ключевых точек, они выстроятся по градиенту на одинаковом расстоянии друг от друга.

Рассмотрим несколько примеров. Красно-синий градиент, распространяющийся по вертикали сверху вниз. Первая ключевая точка градиента будет находиться в его начальной точке, а вторая — в конечной:

```
background-image: linear-gradient(red, blue)
```

Аналогичный градиент, но распространяющийся по горизонтали слева направо:

```
background-image: linear-gradient(to right, red, blue)
```

Градиент, состоящий из трех цветов. Третья ключевая точка, задающая синий цвет, поместится на середине (50%) элемента; оставшаяся часть элемента будет закрашена чистым синим цветом. Вторая ключевая точка, задающая черный цвет, окажется между первой и третьей ключевыми точками (на 25% от ширины элемента):

```
background-image: linear-gradient(to right, red, black, blue 50%)
```

Аналогичный градиент, распространяющийся из нижнего левого в верхний правый угол элемента:

```
background-image: linear-gradient(to right top, red, black, blue 50%)
```

Диагональный градиент, распространяющийся по прямой, которая составляет угол 45° от горизонтали, если считать в направлении по часовой стрелке:

```
background-image: linear-gradient(45deg, red, black, blue)
```

Градиент, включающий пять цветов:

```
background-image: linear-gradient(red, black 10%, green 40%, white 85%,
blue)
```

Создание радиальных градиентов

Радиальный градиент описывается функцией `radial-gradient` (листинг 2.15).

Листинг 2.15. Описание радиального градиента

```
radial-gradient(
  <форма> <размер> at <местоположение начальной точки>,
  <цвет в ключевой точке 1> <местоположение ключевой точки 1>,
  <цвет в ключевой точке 2> <местоположение ключевой точки 2>,
  . . .
  <цвет в ключевой точке n-1> <местоположение ключевой точки n-1>,
  <цвет в ключевой точке n> <местоположение ключевой точки n>
)
```

Для параметра *форма* доступны значения `ellipse` (градиент эллиптической формы) и `circle` (градиент в форме круга). Значение по умолчанию — `ellipse`.

Для эллиптического градиента эллипс, на основе которого он формируется, будет иметь то же соотношение сторон, что и элемент, фоном которого он станет.

Размер градиента можно указать в виде одного из следующих значений:

- `closest-side` — градиент будет заканчиваться у самой близкой к его начальной точке стороны элемента страницы;
- `closest-corner` — градиент будет заканчиваться у самого близкого к его начальной точке угла элемента страницы;
- `farthest-side` — градиент будет заканчиваться у самой дальней от его начальной точки стороны элемента страницы;
- `farthest-corner` — градиент будет заканчиваться у самого дальнего от его начальной точки угла элемента страницы (значение по умолчанию).

Размер градиента также можно указать в виде одного или двух числовых значений, разделенных пробелом. Если указано одно значение, оно задаст и ширину, и высоту градиента. Если указаны два значения, первое задаст ширину градиента, а второе — его высоту.

Местоположение начальной точки радиального градиента задается в том же формате, что и значение атрибута `background-position`. Если местоположение не указано, начальная точка будет располагаться в центре элемента.

Набор ключевых точек задается точно так же, как и у линейного градиента.

Вот несколько примеров. Красно-синий эллиптический градиент; его начальная точка поместится в центре элемента, а формирующий градиент эллипс закончится у его дальнего угла:

```
background-image: radial-gradient(red, blue)
```

Бело-синий эллиптический градиент, начальная точка которого находится в нижнем правом углу элемента, а формирующий градиент эллипс заканчивается у его ближайшей стороны:

```
background-image: radial-gradient(closest-side at right bottom, white,
blue)
```

Аналогичный градиент, но круглой формы и заканчивающийся у дальнего угла элемента:

```
background-image: radial-gradient(circle farthest-corner at right bottom,
white, blue)
```

Сложный градиент с несколькими ключевыми точками, заполняющий четверть элемента:

```
background-image: radial-gradient(50% 50% at right bottom, white,
red 10%, black 45%, blue 60%)
```

Создание повторяющихся градиентов

Нам совсем не обязательно устанавливать последнюю ключевую точку на место конечной точки градиента. Мы можем установить ее на другое место, скажем, посередине элемента, задав для нее соответствующее местоположение:

```
background-image: linear-gradient(red, blue 50%)
```

Тогда оставшаяся часть элемента, не "охваченная" градиентом, будет закрашена тем цветом, который указан в последней ключевой точке, в нашем случае — "чистым" синим цветом.

Но CSS 3 позволяет нам создать *повторяющийся градиент*. Такой градиент будет повторяться за границей, обозначенной последней ключевой точкой, пока не заполнит элемент страницы целиком.

Для создания повторяющихся градиентов предусмотрены функции `repeating-linear-gradient` и `repeating-radial-gradient`:

```
repeating-linear-gradient(<параметры линейного градиента>)
repeating-radial-gradient(<параметры радиального градиента>)
```

Примеры. Повторяющийся линейный красно-синий градиент:

```
background-image: repeating-linear-gradient(red, blue 50%)
```

Повторяющийся радиальный градиент с несколькими ключевыми точками:

```
background-image: repeating-radial-gradient(circle closest-side
at right bottom, white, red 10%, black 45%, blue 60%)
```

Поддержка градиентов появилась в Internet Explorer 10, Firefox 16.0, Chrome 26.0, Opera 12.1 и Safari 6.1.

2.14.10. Анимация с двумя состояниями

Ранее, до появления CSS 3, для создания на страницах анимации, даже относительно простой, требовалось писать довольно сложные программы (о них будет рассказано в *главе 3*). Но сейчас задача Web-аниматоров существенно упростилась — чтобы заставить элемент страницы менять местоположение, размер или цвет, достаточно нескольких строчек CSS-кода.

Введение в анимацию с двумя состояниями

Анимация в стиле CSS 3 формируется путем изменения значения указанного нами атрибута в течение заданного нами времени и по определенному нами закону.

При создании CSS-анимации часто пользуются той же терминологией, как и при задании градиентных фонов (см. ранее). Здесь мы также имеем две ключевые точки: первая находится в начальной точке анимации и задает ее начальное состояние, а вторая — в конечной точке и задает конечное ее состояние.

Начальное состояние анимации задается стилем, который привязывается к анимируемому элементу страницы изначально. Конечное же состояние записывается в другом стиле, который привязывается к элементу в момент начала анимации. Привязать второй стиль можно как программно, так и декларативно, например, применив в нем селектор `:hover`, указывающий на элемент, на который наводится курсор мыши.

Пример:

```
#animated {
  / Задаем начальное состояние анимации
}
#animated:hover {
  / Задаем конечное состояние анимации
}
```

В результате анимация будет запущена при наведении курсора мыши на элемент `animated`.

Анимация, включающая две ключевые точки, носит название *анимации с двумя состояниями*.

Задание продолжительности анимации

Самый важный параметр, который мы должны указать, — это продолжительность анимации. Если мы его не зададим, элемент вообще не будет анимирован.

Продолжительность анимации задается с помощью атрибута `transition-duration`. Для указания значения времени мы можем использовать единицы измерения `s` (секунды) и `ms` (миллисекунды). Значение по умолчанию — 0 (т. е. элемент не анимирован).

Пример:

```
#animated { color: #ff0000; }
#animated:hover { color: #00ff00; transition-duration: 3s }
```

Создаем анимацию, которая запустится при наведении на элемент `animated` курсора мыши и плавно изменит цвет текста с красного на зеленый в течение трех секунд.

Задание анимируемых атрибутов

Атрибут `transition-property` позволяет задать атрибуты, значения которых будут меняться в процессе выполнения анимации ("анимируемые" атрибуты). Их наименования указываются без кавычек.

Также доступны два особых значения:

- `all` — в анимацию будут вовлечены все атрибуты, значения которых изменились (поведение по умолчанию);
- `none` — отключает анимацию.

Внимание!

Значения атрибутов, которые будут вовлечены в анимацию, должны быть указаны в абсолютных единицах измерения: пикселах, пунктах, миллиметрах и т. п. Если задать их в относительных единицах, например, в процентах, анимация выполнена не будет.

Пример:

```
#animated { color: #ff0000; background-color: #ffffff; }
#animated:hover { color: #00ff00; background-color: #000000;
transition-property: color; transition-duration: 3s }
```

Здесь мы вовлекаем в процесс анимации лишь цвет текста, не затрагивая цвет фона. В результате цвет текста будет меняться плавно, а цвет фона, напротив, изменится скачкообразно.

Задержка перед началом анимации

Атрибут `transition-delay` задает задержку перед началом анимации. Допустимы любые из знакомых нам единицы измерения времени. Значение по умолчанию — 0 (т. е. анимация начинается без задержки).

Пример:

```
#animated { color: #ff0000 }
#animated:hover { color: #00ff00; transition-duration: 3s;
transition-delay: 1s }
```

Теперь анимация при наведении курсора мыши на элемент `animated` начнется с задержкой в одну секунду.

Закон анимации

Атрибут `transition-timing-function` устанавливает закон, по которому будет изменяться значение "анимируемого" атрибута. CSS 3 поддерживает довольно много таких законов:

- `ease` — в начале скорость анимации слегка увеличивается, а в конце слегка уменьшается (поведение по умолчанию);
- `linear` — линейный закон; анимация будет выполняться с постоянной скоростью;
- `ease-in` — в начале скорость анимации заметно увеличивается, после чего анимация выполняется с постоянной скоростью;
- `ease-out` — анимация выполняется с постоянной скоростью, а в конце ее скорость заметно уменьшается;
- `ease-in-out` — в начале скорость анимации заметно увеличивается, а в конце заметно уменьшается;
- `cubic-bezier(`*<горизонтальная координата первой опорной точки>, <вертикальная координата первой опорной точки>, <горизонтальная координата второй опорной точки>, <вертикальная координата второй опорной точки>*`)` — закон, соответст-

вующий кубической кривой Безье. Значения *координат ее опорных точек* должны быть заданы в диапазоне между 0 и 1, где 0 обозначает начало анимации, а 1 — ее конец. Однако для всех атрибутов, за исключением `color`, вертикальная координата может выходить за указанный диапазон, что позволяет достичь эффекта "эластичной" анимации;

- `steps(`*<количество шагов анимации>*`, start|end)` — значение "анимируемого" атрибута изменяется не плавно, а скачкообразно — в начале или в конце одного из шагов анимации, количество которых мы задали. *Количество шагов анимации* указывается в виде числа без единицы измерения. Если вторым значением указано `start` изменение значения "анимируемого" атрибута выполняется в начале каждого шага, если `end` или если второе значение вообще не указано, — в его конце;
- `step-start` — "анимируемый" атрибут сразу получит свое конечное значение, которое в дальнейшем не изменится. Эквивалентно указанию `steps(1, start)`;
- `step-end` — "анимируемый" атрибут будет иметь свое начальное значение, которое в конце анимации мгновенно изменится на конечное. Эквивалентно указанию `steps(1, end)`.

Пример анимации, изменяющейся по линейному закопу:

```
#animated:hover { color: #00ff00; transition-duration: 3s;
transition-timing-function: linear }
```

Задаем закон анимации, соответствующий кубической кривой Безье:

```
#animated:hover { transition-timing-function: cubic-bezier(0.1, 0.5, 0.9, 0.5) }
```

А процесс этой анимации будет разбит на пять шагов, и значение "анимируемого" атрибута будет скачкообразно меняться в начале каждого шага:

```
#animated:hover { transition-timing-function: steps(5, start) }
```

Одновременное задание всех параметров анимации

Атрибут `transition` позволяет задать все параметры анимации одновременно:

```
transition: <"анимируемый" атрибут> <продолжительность>
<закон анимации> <задержка>
```

Пример:

```
#animated:hover { transition: color 3s ease-in 0.5s }
```

Создание обратной анимации

Мы создали анимацию, запускающуюся, как только на элемент страницы наводится курсор мыши. Если мы уберем курсор с анимируемого элемента, тот мгновенно вернется в свое изначальное состояние.

А можно ли сделать так, чтобы анимируемый элемент возвращался в изначальное состояние также с применением анимации, т. е. создать для него *обратную анимацию*? Разумеется, можно.

Давайте подумаем. В случае обычной, "прямой" анимации ее начальное состояние задает первый стиль, а конечное — второй. Тогда для создания обратной анимации начальное состояние задаст второй стиль, а конечное — первый. И нам просто нужно поместить описание обратной анимации в первый стиль.

Пример:

```
#animated { color: #ff0000; transition: color 1s }
#animated:hover { color: #00ff00; transition: color 3s }
```

Теперь при уводе курсора мыши с анимируемого элемента цвет его текста плавно вернется к изначальному значению в течение одной секунды.

Сложная анимация

Но что если нам понадобится изменять в процессе анимации значения сразу нескольких атрибутов? Тогда мы можем воспользоваться возможностями CSS по созданию сложной анимации.

Сначала мы перечислим все "анимируемые" атрибуты в атрибуте `transition-property`, разделив их запятыми:

```
transition-property: color, background-color, font-size, width, height;
```

Потом укажем параметры анимации для всех этих атрибутов, перечислив их в атрибутах `transition-duration`, `transition-delay` и `transition-timing-function` также через запятую:

```
transition-duration: 3s, 2s, 3s, 1s, 1.5s;
```

Первое значение параметра будет относиться к первому "анимируемому" атрибуту, второе — ко второму и т. д. Так, в нашем случае значение цвета текста будет изменяться в течение трех секунд, значение цвета фона — в течение двух секунд, а значение размера шрифта — также в течение трех секунд и т. д.

Если в атрибутах `transition-duration`, `transition-delay` или `transition-timing-function` указано меньше значений, чем атрибутов в `transition-property`, набор указанных значений будет повторяться столько раз, чтобы "покрыть" все указанные "анимируемые" атрибуты.

Пример:

```
transition-property: color, background-color, font-size, width, height;
transition-duration: 3s, 2s;
```

В этом случае:

- значение атрибута `color` будет изменяться в течение трех секунд;
- значение атрибута `background-color` — в течение двух секунд;
- `font-size` — трех секунд;
- `width` — двух;
- `height` — трех.

Если же указать в атрибутах `transition-duration`, `transition-delay` или `transition-timing-function` большее число значений, лишние значения будут проигнорированы.

Вот еще один пример CSS-кода, задающего анимацию для цвета текста и цвета фона.

Пример:

```
#animated { color: #ff0000; background-color: #ffffff }
#animated:hover { color: #00ff00; background-color: #000000;
transition-property: color, background-color;
transition-duration: 3s, 2s; transition-delay: 0s, 0.5s;
transition-timing-function: easy-in }
```

Отметим, что для цвета фона указана полусекундная задержка анимации, и для обоих анимируемых параметров задан закон анимации `easy-in`.

Поддержка анимации с двумя состояниями появилась в Internet Explorer 10, Firefox 16.0, Chrome 26.0, Opera 12.1 и Safari 6.1.

2.14.11. Анимация с несколькими состояниями

Анимация с несколькими состояниями, если вновь прибегнуть к терминологии градиентных фонов CSS 3, включает в свой состав несколько ключевых точек. Первая и последняя находятся в начальной и конечной точках анимации и задают ее начальное и конечное состояния. А все прочие ключевые точки располагаются между крайними точками и задают промежуточные состояния анимации.

Описание набора состояний для анимации

Первый наш шаг — описать все состояния такой анимации, сформировав так называемый *набор состояний*. Для этого применяется директива CSS под названием `@keyframes` (листинг 2.16).

Листинг 2.16. Шаблон описания набора состояний анимации

```
@keyframes <имя набора состояний> {
  <момент времени для описываемого состояния 1> {
    <параметры анимируемого элемента для описываемого состояния 1>
  }
  <момент времени для описываемого состояния 2> {
    <параметры анимируемого элемента для описываемого состояния 2>
  }
  . . .
  <момент времени для описываемого состояния n-1> {
    <параметры анимируемого элемента для описываемого состояния n-1>
  }
  <момент времени для описываемого состояния n> {
    <параметры анимируемого элемента для описываемого состояния n>
  }
}
```

Имя набора состояний указывается без кавычек и должно быть уникальным в пределах страницы. В нем допускается использовать буквы латиницы, цифры, дефисы и знаки подчеркивания, причем начинаться оно должно с буквы.

Момент времени для описываемого состояния указывается в процентах от значения продолжительности анимации (как ее задать, мы узнаем потом). Значение `0%` задает начало анимации, `100%` — ее конец, а, скажем, `50%` — ее середину. Вместо `0%` можно использовать предопределенное значение `from`, а вместо `100%` — `to`.

Параметры анимируемого элемента для описываемого состояния записываются в виде атрибутов CSS, задающих эти параметры.

Пример:

```
@keyframes sample {
  from { left: 100px; top: 50px; }
  25% { left: 200px; top: 50px; }
  50% { left: 200px; top: 150px; }
  75% { left: 100px; top: 150px; }
  to { left: 100px; top: 50px; }
}
```

Здесь мы описываем набор из пяти состояний, имеющий имя `sample`. В результате анимируемый элемент будет двигаться по траектории в виде квадрата.

Указание набора состояний и продолжительности анимации

Создав набор состояний, мы можем указать его для анимируемого элемента. Имя набора состояний (без кавычек) указывается в атрибуте `animation-name`. Предопределенное значение `none` отключает любую анимацию элемента (это, кстати, поведение по умолчанию).

Продолжительность анимации задается атрибутом `animation-duration` в том же формате, что применяется в уже знакомом нам атрибуте `transition-duration`.

Пример:

```
#animated { position: absolute; left: 100px; top: 50px; width: 50px;
height: 50px }
#animated:hover { animation-name: sample; animation-duration: 5s }
```

При наведении курсора мыши занускаем анимацию абсолютно позиционированного элемента, описываемую набором состояний `sample`.

Указание задержки перед началом анимации и ее закона

Атрибуты `animation-delay` и `animation-timing-function` задают, соответственно, задержку перед началом анимации и ее закон. Они указываются в тех же форматах, что и в знакомых нам атрибутах `transition-delay` и `transition-timing-function`.

Пример:

```
#animated:hover { animation-name: sample; animation-duration: 5s;
animation-delay: 1s; animation-timing-function: linear }
```

Задание количества повторений анимации

Атрибут `animation-iteration-count` задает количество повторений (проходов) анимации. Оно указывается в виде числа без единицы измерения. Значение `infinite` задает бесконечное повторение анимации. Значение по умолчанию — 1.

Пример анимации, которая будет выполнена нять раз:

```
#animated:hover { animation-name: sample; animation-duration: 5s;
animation-iteration-count: 5 }
```

А эта анимация будет выполняться бесконечно:

```
#animated:hover { animation-name: sample; animation-duration: 5s;
animation-iteration-count: infinite }
```

Направление анимации

Атрибут `animation-direction` позволит нам указать, в каком направлении будет воспроизводиться анимация: в прямом, как мы задали в наборе состояний, в обратном или то в прямом, то в обратном. Доступны четыре значения:

- `normal` — прямое направление анимации (значение по умолчанию);
- `reverse` — обратное направление анимации;
- `alternate` — нечетные проходы анимации воспроизводятся в прямом направлении, а четные — в обратном;
- `alternate-reverse` — нечетные проходы анимации воспроизводятся в обратном направлении, а четные — в прямом.

Понятно, что значения `alternate` и `alternate-reverse` имеет смысл указывать лишь в тех случаях, если анимация повторяется более одного раза:

```
#animated:hover { animation-name: sample; animation-duration: 5s;
animation-iteration-count: infinite; animation-direction: alternate }
```

Текущее состояние анимации

Атрибут `animation-play-state` указывает текущее состояние анимации — продолжает ли она воспроизводиться (значение `running`, используемое по умолчанию) или приостановлена (значение `paused`).

Пример:

```
#animated { position: absolute; left: 100px; top: 50px; width: 50px;
height: 50px; animation-name: sample; animation-duration: 5s;
animation-iteration-count: infinite; animation-play-state: running }
#animated:hover { animation-play-state: paused }
```

Создаем анимированный элемент, который начинает двигаться сразу после загрузки страницы и приостанавливается, если на него навести курсор мыши.

Положение анимированного элемента по окончанию анимации

Атрибут `animation-fill-mode` указывает положение, в котором окажется анимируемый элемент после завершения анимации. Здесь поддерживаются четыре значения:

- `none` — элемент окажется в положении, заданном в привязанном к нему первом стиле. Положение, описанное в наборе состояний (директиве `@keyframes`), в этом случае в расчет не принимается. Это значение по умолчанию;
- `forwards` — элемент окажется в положении, описанном последним состоянием набора;
- `backwards` — элемент окажется в положении, описанном в состоянии, с которого начался первый проход анимации. Это первое состояние для анимации, воспроизводящейся в прямом направлении (для атрибута `animation-direction` задано значение `normal` или `alternate`), или последнее, если анимация воспроизводится в обратном направлении (значения `reverse` или `alternate-reverse`);
- `both` — комбинация значений `forwards` и `backwards`.

Пример:

```
#animated:hover { position: absolute; left: 100px; top: 50px;
width: 50px; height: 50px; animation-name: sample;
animation-duration: 5s; animation-fill-mode: forwards }
```

Одновременное задание всех параметров анимации

Если мы хотим задать все параметры анимации сразу, то применим атрибут `animation`:

```
animation: <имя набора состояний> <продолжительность> <закон>
<задержка> <количество повторений> <направление>
<положение по окончанию анимации> <текущее состояние>
```

Пример

```
#animated:hover { animation: sample 5s linear infinite alternate }
```

Сложная анимация

И наконец, как и в случае анимации с двумя состояниями, мы можем указать для анимируемого элемента несколько наборов состояний и параметров к ним, создав тем самым более сложную анимацию (листинг 2.17).

Листинг 2.17. Шаблон сложной анимации

```
@keyframes anim1 {
  / Набор состояний 1
}
@keyframes anim2 {
  / Набор состояний 2
}
#animated { animation-name: anim1, anim2; animation-direction: 3s, 6s }
```

Здесь действуют те же правила, что мы изучили применительно к анимации с двумя состояниями.

Создание кроссплатформенной анимации

Поддержка анимации с несколькими состояниями появилась в Internet Explorer 10, Firefox 16.0, Chrome 4.0, Opera 12.1 и Safari 4.0.

Однако в Chrome и Safari необходимо предварять наименования атрибутов префиксом `-webkit-`, например `-webkit-animation-name`, `-webkit-animation-duration` и т. п. Стандартные наименования соответствующих атрибутов в этих Web-обозревателях не поддерживаются.

Обычно в таблицах стилей для одной анимации с несколькими состояниями указывают два задающих ее параметры набора стилей: один — с применением стандартных наименований атрибутов, другой — с наименованиями, предваренными префиксом `-webkit-`. Это позволяет создать кроссплатформенную анимацию, работающую во всех Web-обозревателях.

Пример:

```
#animated:hover { animation-name: sample; animation-duration: 5s;
-webkit-animation-name: sample; -webkit-animation-duration: 5s }
```

2.14.12. Двумерные преобразования

Преобразованиями в терминологии CSS 3 называются всякого рода смещения элементов страниц относительно их изначального местоположения, наклоны, повороты и масштабирование. Такие манипуляции можно выполнить без необходимости превращения элемента в свободно позиционированный.

Мы начнем с *двумерных преобразований*, которые выполняются в плоскости страницы.

Как задаются преобразования и их параметры

Для указания, какое, собственно, преобразование мы хотим применить к элементу страницы, и его параметров служит атрибут CSS `transform`. В качестве его значения указывается функция CSS, задающая само выполняемое преобразование; параметры последнего указываются в виде параметров функции, в круглых скобках, которые ставятся сразу же после ее имени:

```
transform: <функция, задающая преобразование>↳
(<параметры преобразования, перечисленные через запятую>)
```

Пример:

```
div#transformed { transform: translateY(200px) }
```

Смещаем контейнер с идентификатором `transformed` на 200 пикселов вниз.

В дальнейшем разговор в основном пойдет о функциях CSS, применяемых для указания преобразований и их параметров. Попутно мы рассмотрим еще несколько атрибутов CSS, которые могут нам пригодиться.

Смещение

Проще сместить элемент страницы относительно его изначального местоположения. Для этого предназначены три функции CSS, которые мы сейчас рассмотрим.

Функции `translateX` и `translateY` выполняют смещение, соответственно, по горизонтали и вертикали или, если пользоваться математическими терминами, по горизонтальной (`x`) и вертикальной (`y`) координатным осям. Величина смещения указывается в качестве единственного параметра функции.

Величина горизонтального смещения отсчитывается вправо, а величина вертикального смещения — вниз. Если нам потребуется сдвинуть элемент влево или вверх, мы укажем отрицательное значение соответствующей величины.

Сдвигаем контейнер `el1` на 100 пикселов вправо:

```
div#el1 { transform: translateX(100px) }
```

Сдвигаем контейнер `el2` на 200 пикселов вверх:

```
div#el2 { transform: translateY(-200px) }
```

Функция `translate` позволит нам сместить элемент сразу по обеим координатным осям. В качестве первого параметра в ней указывается величина смещения по горизонтали, а в качестве второго — величина смещения по вертикали.

Смещаем контейнер `el3` на 50 пикселов влево и на 200 пикселов вниз:

```
div#el3 { transform: translate(-50px, 200px) }
```

Масштабирование

Для масштабирования элемента примеияются функции CSS `scaleX` и `scaleY`. Первая выполняет масштабирование вдоль горизонтальной оси координат, вторая — вдоль вертикальной.

В качестве единственного параметра этих функций указывается относительная величина, на которую следует выполнить масштабирование элемента. Величина, бóльшая единицы, задает увеличение масштаба, а величина, меньшая единицы, — его уменьшение. Единица измерения при этом не указывается.

Точка, относительно которой элемент будет в результате масштабирования вырастать или уменьшаться, по умолчанию находится в его центре и носит название *точки начала координат*. (Мы можем сместить ее в другое место элемента и скоро узнаем, как это делается.)

Увеличиваем контейнер `el1` в ширину вдвое:

```
div#el1 { transform: scaleX(2) }
```

А высоту контейнера `el2` уменьшаем вдвое:

```
div#el1 { transform: scaleY(0.5) }
```

Если нам потребуется выполнить масштабирование элемента одновременно по обеим координатным осям, то воспользуемся функцией `scale`. Первый ее параметр задает величину масштаба по горизонтали, а второй — по вертикали.

Растягиваем контейнер `el3` вдвое в ширину и сжимаем вдвое в высоту:

```
div#el3 { transform: scale(2, 0.5) }
```

Наклон

Наклонить элемент страницы можно, использовав функции `skewX` и `skewY`. Первая выполняет наклон по горизонтальной оси координат, вторая — по вертикальной.

В качестве единственного параметра этих функций указывается угол, на который требуется выполнить наклон. Угол этот отсчитывается против часовой стрелки; чтобы выполнить наклон по часовой стрелке, следует указать отрицательное значение угла.

Элемент будет сдвигаться относительно точки начала координат. По умолчанию она находится в середине элемента.

Рассмотрим примеры. Наклоняем контейнер `el1` на 25° по горизонтали против часовой стрелки:

```
div#el1 { transform: skewX(25deg) }
```

А контейнер `el2` мы наклоним на 5° по вертикали по часовой стрелке:

```
div#el2 { transform: skewY(-5deg) }
```

Наклонить элемент сразу по горизонтали и вертикали удобно с помощью функции `skew`. В качестве первого параметра она принимает угол наклона по горизонтали, а в качестве второго — угол наклона по вертикали:

```
div#el3 { transform: skew(45deg, -60deg) }
```

Поворот

Поворот элемента на заданный угол выполняется с помощью функции `rotate`. Нужный угол задается в качестве ее единственного параметра.

Положительное значение угла задает поворот по часовой стрелке, отрицательное — против часовой стрелки. Точка начала координат, вокруг которой поворачивается элемент, по умолчанию находится в его центре.

Поворачиваем контейнер `el1` на 45° по часовой стрелке:

```
div#el1 { transform: rotate(45deg) }
```

Поворачиваем контейнер `el2` на 90° против часовой стрелки:

```
div#el2 { transform: rotate(-90deg) }
```

Если этот контейнер содержит текст, мы получим вертикальную надпись.

Позиционирование точки начала координат для двумерных преобразований

Ранее неоднократно говорилось, что по умолчанию точка начала координат располагается в середине элемента страницы, к которому примеияются преобразования; соответственно, относительно этой точки элемент будет масштабироваться, наклоняться и поворачиваться. Но у нас есть возможность установить точку начала координат в любое место элемента.

Для указания месторасположения этой точки CSS 3 предусматривает атрибут `transform-origin`:

```
transform-origin: <горизонтальная координата> <вертикальная координата>
```

Горизонтальную координату можно указать в виде числа, представляющего собой расстояние от левой границы элемента до самой точки, или одного из предопределенных значений: `left` (левая граница элемента), `center` (центр) или `right` (правая граница). *Вертикальная координата* также задается либо в виде числового расстояния от верхней границы элемента, либо одного из предопределенных значений: `top` (верхняя граница элемента), `center` (центр) и `bottom` (нижняя граница).

Значение атрибута CSS `transform-origin` по умолчанию — `50% 50%` (т. е. точка начала координат находится в центре элемента).

Указав местоположение точки начала координат в абсолютных единицах измерения, мы можем даже вынести эту точку за пределы элемента. Иногда такой прием может быть полезным.

Смещаем точку начала координат в верхний левый угол блока `el` и поворачиваем его на 45° по часовой стрелке:

```
div#el { transform: rotate(45deg); transform-origin: left top }
```

В результате блок будет повернут вокруг своего верхнего левого угла.

Сложные двумерные преобразования

Что ж, как сместить, отмасштабировать, наклонить или повернуть элемент страницы, мы выяснили. Но что если нам потребуется, скажем, одновременно сместить и повернуть его, т. е. выполнить *сложное преобразование*?

Для этого достаточно все функции, указывающие отдельные преобразования, записать в значении атрибута CSS `transform` друг за другом, разделив их пробелами.

Смещаем контейнер `el1` на 200 пикселов вниз и увеличиваем его ширину вдвое:

```
div#el1 { transform: translateY(200px) scaleX(2) }
```

Наклоняем контейнер `el2` на 15° против часовой стрелки и поворачиваем на 45° также против часовой стрелки:

```
div#el2 { transform: skewX(15deg) rotate(-45deg) }
```

Кроссплатформенные двумерные преобразования

Поддержка двумерных преобразований в том виде, в котором они определены в стандарте CSS 3, появилась в Microsoft Internet Explorer 10, Firefox 16.0, Chrome 36.0 и Opera 23.0. Safari их не поддерживает.

Однако двумерные преобразования можно использовать и в Safari. Для этого достаточно предварить названия задающих их атрибутов CSS префиксом `-webkit-`:

```
div#el3 { -webkit-transform: translate(-50px, 200px) }
```

Как правило, в реальном CSS-коде определение каждого преобразования записывают дважды: с применением стандартного атрибута CSS и атрибута, предваренного префиксом `-webkit-`. Такой код подходит для всех Web-обозревателей.

Пример:

```
div#el3 { transform: translate(-50px, 200px) }
div#el3 { -webkit-transform: translate(-50px, 200px) }
```

2.14.13. Трехмерные преобразования

Трехмерные преобразования выполняются в воображаемом трехмерном пространстве, проецируемом на плоскость страницы.

Двумерные преобразования выполняются по горизонтальной и вертикальной координатным осям, носящим в математике наименования `x` и `y`. В случае трехмерных преобразований к ним добавляется ось `z`, направленная из точки начала координат перпендикулярно воображаемой поверхности элемента. Положительные значения координаты по этой оси отсчитываются в направлении к посетителю, отрицательные — от посетителя.

Все трехмерные преобразования элементов страницы выполияются в ортогональной проекции на поверхность контейнера, в котором они находятся.

Перспектива

Перспектива (ее также называют *глубиной перспективы*) в CSS 3 — это расстояние, отсчитываемое по оси `z`, между воображаемым наблюдателем (посетителем) и поверхностью контейнера, в котором находится подвергаемый преобразованиям элемент. Фактически перспектива добавляет трехмерному изображению "глубину", благодаря чему мы получим на странице тот эффект, которого собираемся добиться, применяя трехмерное преобразование.

По умолчанию перспектива равна нулю, т. е. фактически отсутствует. Из-за этого при выполнении трехмерных преобразований мы получим не тот результат, на который рассчитываем.

Следовательно, сначала нам нужно указать перспективу. Сделать это можно с помощью атрибута CSS `perspestive`, указав в качестве его значения величину перспективы. Значение `none` убирает перспективу, делая ее равной нулю (значение по умолчанию).

Запомним, что данный атрибут и, соответственно, перспектива указывается для контейнера элемента, к которому применяется преобразование, но не для самого этого элемента.

Задаем перспективу в 300 пикселов для всех элементов, находящихся на странице:

```
body { perspective: 300px }
```

Выполнение трехмерных преобразований

В случае трехмерных преобразований для смещения, масштабирования и наклона элементов по горизонтальной и вертикальной координатным осям мы можем использовать те же функции, что примеияли для выполнения двумерных преобразований. Так что здесь нам остается лишь изучить функции, специфичные лишь для собственно трехмерных преобразований; в основном, они выполняют преобразования по координатной оси `z`.

Вот поддерживаемые CSS функции, задаваемые ими трехмерные преобразования и примеры использования:

- `translateZ` — смещение элемента по оси `z`. Единственный параметр задает величину смещения; положительные значения координат отсчитываются в направлении к посетителю, отрицательные — от посетителя:

```
div#el1 { transform: translateZ(50px) }
div#el2 { transform: translateZ(-1cm) }
```

- `translate3d` — смещение элемента одновременно по всем координатным осям. Принимает три параметра, задающие, соответственно, смещения по осям `x`, `y` и `z`:

```
div#el { transform: translate3d(100px, -200px, 1cm) }
```

- `scaleZ` — масштабирование по оси `z`. Единственный параметр задает масштаб элемента:

```
div#el { transform: scaleZ(1.5) }
```

  Масштабировать элемент по оси `z` имеет смысл только в том случае, если перед масштабированием к нему был применен поворот вокруг оси `x` или `y`. В противном случае визуально этот элемент никак не изменится.

- `scale3d` — масштабирование сразу по всем трем координатным осям. Принимает три параметра, задающие, соответственно, величины масштаба по осям `x`, `y` и `z`:

```
div#el { transform: scale3d(1.5, 0.5, 2) }
```

- `rotateX`, `rotateY` и `rotateZ` — поворот элемента по координатным осям `x`, `y` и `z` соответственно. Единственный параметр задает угол поворота:

```
div#el1 { transform: rotateX(45deg) }
div#el2 { transform: rotateY(-15deg) }
div#el3 { transform: rotateZ(0.5turn) }
```

Задание точки зрения

Мы уже знаем, что перед выполнением трехмерных преобразований необходимо указать перспективу; в противном случае мы можем вообще не получить никакого видимого результата. Также мы знаем, что перспектива в CSS 3 — это расстояние между наблюдателем и плоскостью контейнера элемента, над которым выполняется преобразование.

Но где же находится этот воображаемый наблюдатель? В особой точке, носящей название *точки зрения*. Ее Web-обозреватель и имеет в виду, когда рассчитывает проекцию преобразуемого элемента на двумерную плоскость его контейнера.

По умолчанию точка зрения находится в центре контейнера элемента, к которому применяется преобразование, и отстоит от него на величину перспективы по оси `z`. И, что очень важно, всегда остается в этом месте независимо от того, какое преобразование мы применяем к элементам, находящимся в этом контейнере.

Но при необходимости мы можем сместить точку зрения в другое место, тем самым заставив воображаемого наблюдателя "взглянуть" на подвергаемый преобразованиям элемент с другой стороны. Это можно сделать с помощью атрибута CSS `perspective-origin`:

```
perspective-origin: <горизонтальная координата>
[<вертикальная координата>]
```

Координаты проекции точки зрения указываются в том же формате, как и местоположение точки начала координат (атрибут CSS `transform-origin`). Как видим, здесь указываются лишь горизонтальная и вертикальная координаты (координата точки зрения по оси `z` — это фактически перспектива, задаваемая атрибутом стиля `perspective`). Если *вертикальная координата* не указана, она получит значение `50%`.

Значение атрибута CSS `perspective-origin` по умолчанию — `50% 50%` (т. е. проекция точки зрения находится в центре контейнера).

Атрибут `perspective-origin`, как и атрибут `perspective`, указывается для контейнера элементов, к которым будут применены трехмерные преобразования.

Пример:

```
body { perspective: 300px; perspective-origin: right top; }
div#transformed: { transform: rotateY(175deg) }
```

Задаем перспективу, местоположение точки зрения и трехмерное преобразование, которое будет выполнено с учетом нового расположения точки зрения.

Скрытие обратной стороны элемента

Давайте рассмотрим код листинга 2.18.

Листинг 2.18. Пример преобразования элемента

```
body { perspective: 300px; }
div#transformed { font-size: 36pt; width: 150px; background-color: black;
```

```
color: white; transform: rotateY(175deg) }
. . .
<div id="transformed">Блок 1</div>
```

Он создает контейнер с идентификатором `transformed` и поворачивает его относительно вертикальной оси на 175° по часовой стрелке. В результате контейнер окажется, если так можно сказать, перевернутым, и мы увидим его "обратную" сторону (рис. 2.1).

Рис. 2.1. Перевернутый элемент страницы с видимой "обратной" стороной

В некоторых случаях, например при создании с помощью преобразований трехмерных фигур, подобный результат неприемлем. Поэтому нам может потребоваться скрыть "обратную" сторону таких перевернутых элементов.

Для чего мы воспользуемся атрибутом CSS `backface-visibility`. Он поддерживает два значения:

- `visible` — сделать "обратную" сторону элемента видимой (значение по умолчанию);
- `hidden` — полностью скрыть элемент, если он окажется повернутым к посетителю своей "обратной" стороной.

Атрибут `backface-visibility` указывается непосредственно для элемента, к которому применяются трехмерные преобразования.

Пример:

```
div#transformed { font-size: 36pt; width: 150px; background-color: black;
color: white; transform: rotateY(175deg); backface-visibility: hidden }
```

Теперь Web-обозреватель вообще не выведет на экран контейнер `transformed`. Что вполне понятно — ведь этот элемент оказался повернутым к наблюдателю своей "обратной" стороной.

Режим проецирования элементов на контейнер

По умолчанию элементы, находящиеся в контейнере, к которому были применены трехмерные преобразования, отображаются в его плоскости. Если мы применим трехмерные преобразования и к ним, они так и останутся проекциями на плоскость контейнера.

Рассмотрим пример кода, приведенный в листинге 2.19.

Листинг 2.19. Трехмерные преобразования вложенных элементов

```
body { perspective: 300px; }
div#cont { width: 300px; height: 300px; background-color: yellow;
```

```
transform: rotateX(45deg) }
div#ch { width: 150px; height: 150px; background-color: red;
transform: translate3d(50px, 50px, 100px) }
. . .
<div id="cont">
  <div id="ch"></div>
</div>
```

Здесь мы создали два вложенных друг в друга контейнера и задали для обоих трехмерные преобразования: "внешний" контейнер повернули, а вложенный в него сместили, в том числе и по оси `z`, чтобы визуально приподнять его над "внешним".

Результат, который покажет нам Web-обозреватель, представлен на рис. 2.2. Видно, что вложенный контейнер так и находится в плоскости "внешнего".

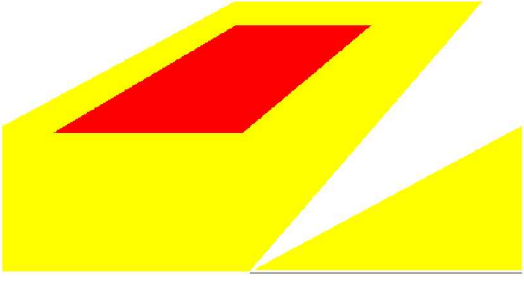


Рис. 2.2. Вложенный контейнер находится в плоскости "внешнего"

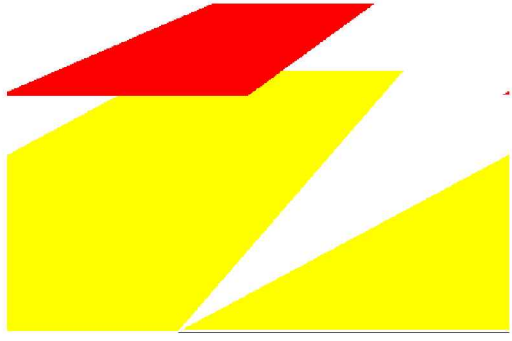


Рис. 2.3. Вложенный контейнер вышел из плоскости "внешнего"

Однако мы можем сделать так, чтобы при применении к ним трехмерных преобразований вложенные элементы вышли из плоскости контейнера и отображались в виде проекции непосредственно на странице. Так можно создавать трехмерные фигуры.

Режим проецирования элементов на контейнер, в который они вложены, задается с помощью атрибута CSS `transform-style`, поддерживающего два значения:

- `flat` — элементы выводятся в виде проекции на поверхность контейнера (значение по умолчанию);
- `preserve-3d` — элементы существуют в воображаемом трехмерном пространстве отдельно от контейнера и отображаются как проекции непосредственно на поверхности страницы.

Если мы добавим к приведенному в листинге 2.14 CSS-коду стиль `div#cont { transform-style: preserve-3d }`, то Web-обозреватель выведет нам изображение, показанное на рис. 2.3. Мы видим, что вложенный контейнер вышел за пределы "внешнего" и существует как бы отдельно от него.

Позиционирование точки начала координат для трехмерных преобразований

Для трехмерных преобразований предусмотрен расширенный формат атрибута CSS `transform-origin`, указывающего местоположение точки начала координат:

```
transform-origin: <горизонтальная координата> <вертикальная координата>
<координата по оси z>
```

По умолчанию значение *координаты по оси z* — 0.

Пример:

```
div#el { transform: rotateX(0.2rad); transform-origin: left top 200px }
```

Сложные трехмерные преобразования

Для создания сложных трехмерных преобразований мы можем пользоваться уже знакомым нам приемом, перечисляя функции, задающие отдельные преобразования, через пробел:

```
div#el1 { transform: translateZ(-200px) rotateY(30deg) }
```

Мы также можем смешивать двумерные и трехмерные преобразования:

```
div#el2 { transform: skewX(15deg) rotateZ(-45deg) }
```

Кроссплатформенные трехмерные преобразования

Трехмерные преобразования поддерживают те же версии Web-обозревателей, что и двумерные. И, как и двумерные, трехмерные преобразования в Safari поддерживаются атрибутами CSS с добавленным префиксом `-webkit-`:

```
div#el3 { transform: rotateZ(45deg) }
div#el3 { -webkit-transform: rotateZ(45deg) }
```

Следует также учесть, что Internet Explorer поддерживает атрибут CSS `transform-style`, начиная с версии 11.

2.15. Проверка CSS-кода на соответствие стандартам

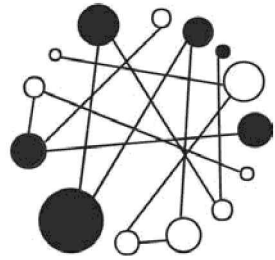
После создания документа, содержащего каскадные таблицы стилей, его необходимо проверить на отсутствие ошибок и соответствие стандартам. Ведь можно случайно допустить опечатку в названии атрибута или указать некорректное значение. Web-браузеры обычно не сообщают об ошибках, а пытаются их обработать. Поэтому о существовании ошибок можно узнать только в случае, если Web-браузер неправильно их обработал и это визуально видно.

Для проверки CSS-кода предназначен сайт **http://jigsaw.w3.org/css-validator/**. Чтобы проверить документ, размещенный в Интернете, достаточно ввести URL-адрес и нажать кнопку **Проверить**. Можно также загрузить файл или вставить CSS-код в поле ввода многострочного текста. Если после проверки были обнаружены ошибки, то будет выведено их подробное описание. После исправления ошибок следует повторно проверить CSS-код.

Внимание!

Если CSS-код встроен в HTML-документ, то сначала нужно проверить сам документ на отсутствие ошибок с помощью сайта **http://validator.w3.org/**.

ГЛАВА 3



Основы JavaScript. Создаем страницы, реагирующие на действия пользователей

3.1. Основные понятия

JavaScript — это язык программирования, позволяющий сделать Web-страницу интерактивной, т. е. реагирующей на действия пользователя.

Последовательность инструкций (называемая *программой*, *скриптом* или *сценарием*) выполняется *интерпретатором*, встроенным в сам Web-браузер. Иными словами, код программы внедряется в HTML-документ и выполияется на стороне клиента. Для выполнения программы даже не нужно перезагружать Web-страницу. Все программы выполняются в результате возникновения какого-то события. Например, перед отправкой данных формы можно проверить их на донустимые значения и, если значения не соответствуют ожидаемым, запретить отправку данных.

3.2. Первая программа на JavaScript

При изучении языков программирования приято начинать с программы, выводящей надпись "Hello, world". Не будем нарушать традицию и продемонстрируем, как это будет выглядеть на JavaScript (листинг 3.1).

Листинг 3.1. Первая программа

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Первая программа</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
```

```
<body>
<script type="text/javascript">
<!--
document.write("Hello, world");
//-->
</script>
<noscript>
   <p>Ваш Web-браузер не поддерживает JavaScript</p>
</noscript>
</body>
</html>
```

Набираем код в Блокноте и сохраняем в формате HTML, например, под именем test.html. Занускаем Web-браузер и открываем сохраненный файл.

Возможны следующие варианты:

- в окне Web-браузера отображена надпись "Hello, world" — значит, все нормально;
- отобразилась надпись "Ваш Web-браузер не поддерживает JavaScript" и Web-браузер задает вопрос "Запустить скрипты?" — значит, в настройках Web-браузера установлен флажок напротив пункта **Подтверждать заиуск скриитов**. Можно либо установить флажок напротив пункта **Разрешить заиуск сириитов**, либо каждый раз отвечать "Да" на этот вопрос;
- отобразилась надпись "Ваш Web-браузер не поддерживает JavaScript" и Web-браузер не задает никаких вопросов — значит, в настройках Web-браузера установлен флажок напротив пункта **Заиретить заиуск скриитов**. Надо установить флажок напротив пункта **Разрешить заиуск сириитов**;
- в окне Web-браузера нет никаких надписей — значит, донущена опечатка в коде программы. Следует иметь в виду, что в JavaScript регистр имеет важное значение. Строчные и прописные буквы считаются разными. Более того, каждая буква, каждая кавычка имеет значение. Достаточно ошибиться в одной букве, и вся программа работать не будет.

Итак, мы столкнулись с первой проблемой при использовании JavaScript — любой пользователь может отключить запуск скриптов в настройках Web-браузера. Но эта проблема не единственная. Разные Web-браузеры могут по-разному выполнять код программы. По этой причине приходится писать персональный код под каждый Web-браузер. Все примеры скриптов в этой книге написаны под Microsoft Internet Explorer и могут не работать в других Web-браузерах. Это следует помнить.

Вернемся к нашему примеру. Программа внедряется в HTML-документ с помощью парного тега `<script>`. В качестве значения параметра `type` указывается MIME-тип `text/javascript`. Кроме того, может быть указан параметр `language`, который задает название языка программирования (в нашем случае — JavaScript). Данный параметр использовался в ранних версиях HTML, а в настоящее время указывается только для совместимости, одновременно с параметром `type`:

```
<script type="text/javascript" language="JavaScript">
```

Если Web-браузер не поддерживает JavaScript или выполнение скриптов запрещено в настройках Web-браузера, то будет выведен текст между тегами `<noscript>` и `</noscript>`. По этой же причине код программы между тегами `<script>` и `</script>` заключается в теги HTML-комментария `<!--` и `-->`, иначе Web-браузеры, не поддерживающие JavaScript, выведут код скрипта в виде обычного текста:

```
<!--
document.write("Hello, world");
//-->
```

Интерпретатор JavaScript игнорирует открывающий тег HTML-комментария `<!--`, т. к. никакая строка программы JavaScript не может начинаться с "<". Но закрывающий тег HTML-комментария `-->`, начинающийся с двух минусов (`--`), распознается интерпретатором как ошибка, т. к. в JavaScript имеется предопределенный оператор `--`. По этой причине перед закрывающим тегом необходимо поставить символы комментария языка JavaScript (`//`):

```
//-->
```

> ***Примечание***
>
> В настоящее время практически все Web-браузеры распознают тег `<script>`. Поэтому особого смысла заключать программу в символы комментария нет.

Строка

```
document.write("Hello, world");
```

содержащая инструкцию отобразить надпись "Hello, world" в окне Web-браузера, называется *выражением*. Каждое выражение в JavaScript заканчивается точкой с запятой.

> ***Примечание***
>
> Необходимо заметить, что это необязательное требование. Тем не менее рекомендуется указывать точку с запятой в конце каждого выражения. Это позволит избежать множества ошибок в дальшейшем.

3.3. Комментарии в JavaScript

Все, что расположено после "`//`" до конца строки, в JavaScript считается *однострочным комментарием*:

```
// Однострочный комментарий
```

Однострочный комментарий можно записать после выражения:

```
document.write("Hello, world"); // Однострочный комментарий
```

Кроме того, существует *многострочный комментарий*. Он начинается с символов `/*` и заканчивается символами `*/`:

```
/*
Многострочный комментарий
*/
```

3.4. Вывод результатов работы программы и ввод данных

Прежде чем начинать изучение языка JavaScript, рассмотрим встроенные диалоговые окна, которые позволят нам выводить результаты работы программы или значения переменных, а также вводить данные.

3.4.1. Окно с сообщением и кнопкой *OK*

Метод `alert()` отображает диалоговое окно с сообщением и кнопкой **OK**. В листинге 3.2 демонстрируется вывод приветствия с помощью метода `alert()`.

Листинг 3.2. Метод `alert()`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Метод alert()</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
window.alert("Hello, world");
//-->
</script>
</body>
</html>
```

Сообщение можно разбить на строки с помощью последовательности символов `\n` (листинг 3.3).

Листинг 3.3. Разбиение сообщения на строки

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Разбиение сообщения на строки</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
window.alert("Строка1\nСтрока2\n\nСтрока4");
```

```
//-->
</script>
</body>
</html>
```

3.4.2. Окно с сообщением и кнопками *OK* и *Cancel*

Метод `confirm()` отображает диалоговое окно с сообщением и двумя кнопками **OK** и **Cancel** (листинг 3.4). Он возвращает значение `true`, если нажата кнопка **OK**, и `false` — если **Cancel**.

Листинг 3.4. Метод `confirm()`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Метод confirm()</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
if (window.confirm("Нажмите одну из кнопок")) {
   window.alert("Нажата кнопка OK");
}
else {
   window.alert("Нажата кнопка Cancel");
}
//-->
</script>
</body>
</html>
```

3.4.3. Окно с полем ввода и кнопками *OK* и *Cancel*

Метод `prompt()` отображает диалоговое окно с сообщением, полем ввода и двумя кнопками **OK** и **Cancel** (листинг 3.5). Он возвращает введенное значение, если нажата кнопка **OK**, или специальное значение `null`, если нажата кнопка **Cancel**.

Листинг 3.5. Метод `prompt()`

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

```
<head>
 <title>Метод prompt()</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var n = window.prompt("Введите ваше имя", "Это значение по умолчанию");
if (n==null) {
   document.write("Вы нажали Cancel");
}
else {
   document.write("Привет " + n);
}
//-->
</script>
</body>
</html>
```

3.5. Переменные

Переменные — это участки памяти, используемые программой для хранения данных. Каждая переменная должна иметь уникальное имя в программе, состоящее из латинских букв, цифр и знаков подчеркивания. Первым символом может быть либо буква, либо знак подчеркивания. В названии переменной может также присутствовать символ `$`. Имена переменных не должны совпадать с зарезервированными ключевыми словами языка JavaScript.

Правильные имена переменных:

```
x, strName, y1, _name, frame1
```

Неправильные имена переменных:

```
1y, ИмяПеременной, frame
```

Последнее имя неправильное, т. к. является ключевым словом.

При указании имени переменной важно учитывать регистр букв: `strName` и `strname` — разные переменные.

В программе переменные объявляются с помощью ключевого слова `var`.

```
var strName;
```

Можно объявить сразу несколько переменных в одной строке, указав их через запятую.

```
var x, strName, y1, _name, frame1;
```

3.6. Типы данных и инициализация переменных. Определение типа данных переменной

В JavaScript переменные могут содержать следующие типы данных:

- `number` — целые числа или числа с плавающей точкой (дробные числа);
- `string` — строки;
- `boolean` — логический тип данных. Может содержать значения `true` (истина) или `false` (ложь);
- `function` — функции. В языке JavaScript ссылку на функцию можно присвоить какой-либо переменной. Для этого название функции указывается без круглых скобок. Кроме того, функции имеют свойства и методы;
- `object` — массивы, объекты, а также переменная со значением `null`.

При инициализации переменной JavaScript автоматически относит переменную к одному из типов данных. Что такое *инициализация переменных*? Это операция присвоения переменной начального значения.

Значение переменной присваивается с помощью оператора =.

```
Number1 = 7; // Переменной Number1 присвоено значение 7
Number2 = 7.8;
// Переменной Number2 присвоено значение с плавающей точкой
String1 = "Строка"; // Переменной String1 присвоено значение Строка
String2 = 'Строка';
// Переменной String2 также присвоено значение Строка
Boolean1 = true;
// Переменной Boolean1 присвоено логическое значение true
Str1 = null; // Переменная Str1 не содержит данных
```

Переменной может быть присвоено начальное значение сразу при ее объявлении:

```
var str1 = "Строка";
var str2 = "Строка", Number1 = 7;
// Можно задать начальные значения сразу нескольким переменным.
```

Если в программе обратиться к переменной, которая не объявлена, то возникнет критическая ошибка. Если переменная объявлена, но ей не присвоено начальное значение, то значение предполагается равным `undefined`.

Оператор `typeof` возвращает строку, описывающую тип данных переменной. Продемонстрируем это на примере (листинг 3.6).

Листинг 3.6. Типы данных

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

```
<head>
 <title>Типы данных</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var Number1 = 7;
var Number2 = 7.8;
var String1 = "Строка";
var String2 = 'Строка';
var Boolean1 = true;
var Str1 = null, Str2;
document.write("Number1 - " + typeof (Number1) + "<br>");
document.write("Number2 - " + typeof (Number2) + "<br>");
document.write("String1 - " + typeof (String1) + "<br>");
document.write("String2 - " + typeof (String2) + "<br>");
// Скобки можно не указывать
document.write("Boolean1 - " + typeof Boolean1 + "<br>");
document.write("Str1 - " + typeof Str1 + "<br>");
document.write("Str2 - " + typeof Str2);
//-->
</script>
</body>
</html>
```

3.7. Операторы JavaScript

Операторы позволяют выполнить определенные действия с данными. Например, операторы присваивания служат для сохранения данных в переменной, математические операторы позволяют произвести арифметические вычисления, а оператор конкатенации строк используется для соединения двух строк в одну. Операторы берут одно или два значения, представляющих собой переменную, константу или другое выражение, содержащее операторы или функции, и возвращают одно значение, определяемое по исходным данным. Рассмотрим доступные в JavaScript операторы более подробно.

3.7.1. Математические операторы

- `+` — сложение:

```
Z = X + Y;
```

- `-` — вычитание:

```
Z = X - Y;
```

- `*` — умножение:

```
Z = X * Y;
```

- `/` — деление:

```
Z = X / Y;
```

- `%` — остаток от деления:

```
Z = X % Y;
```

- `++` — оператор инкремента. Увеличивает значение переменной на 1:

```
Z++; // Эквивалентно Z = Z + 1;
```

- `--` — оператор декремента. Уменьшает значение переменной на 1:

```
Z--; // Эквивалентно Z = Z - 1;
```

Операторы инкремента и декремента могут использоваться в постфиксной или префиксной формах:

```
Z++; Z--; // Постфиксная форма
++Z; --Z; // Префиксная форма
```

В чем разница? При постфиксной форме (`Z++`) возвращается значение, которое переменная имела перед операцией, а при префиксной форме (`++Z`) — вначале выполняется операция и только потом возвращается значение. Продемонстрируем разницу на примере (листинг 3.7).

Листинг 3.7. Постфиксная и префиксная форма

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Постфиксная и префиксная форма</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var X, Y;
X = 5;
Y = X++; // Y = 5, X = 6
var msg;
msg = "<b>Постфиксная форма (Y = X++;):<" + "/b><br> Y = ";
msg += Y + "<br>X = " + X + "<br><br>";
X = 5;
Y = ++X; // Y = 6, X = 6
msg += "<b>Префиксная форма (Y = ++X;):<" + "/b><br> Y = ";
msg += Y + "<br>X = " + X;
document.write(msg);
```

```
//-->
</script>
</body>
</html>
```

3.7.2. Операторы присваивания

- `=` — присваивает переменной значение:

```
Z = 5;
```

- `+=` — увеличивает значение переменной на указанную величину:

```
Z += 5; // Эквивалентно Z = Z + 5;
```

- `-=` — уменьшает значение переменной на указанную величину:

```
Z -= 5; // Эквивалентно Z = Z - 5;
```

- `*=` — умножает значение переменной на указанную величину:

```
Z *= 5; // Эквивалентно Z = Z * 5;
```

- `/=` — делит значение переменной на указанную величину:

```
Z /= 5; // Эквивалентно Z = Z / 5;
```

- `%=` — делит значение переменной на указанную величину и возвращает остаток:

```
Z %= 5; // Эквивалентно Z = Z % 5;
```

3.7.3. Двоичные операторы

- `~` — двоичная инверсия:

```
Z = ~X;
```

- `&` — двоичное И:

```
Z = X & Y;
```

- `|` — двоичное ИЛИ:

```
Z = X | Y;
```

- `^` — двоичное исключающее ИЛИ:

```
Z = X ^ Y;
```

- `<<` — сдвиг влево — сдвиг влево на один или более разрядов с заполнением младших разрядов нулями:

```
Z = X << Y;
```

- `>>` — сдвиг вправо — сдвиг вправо на один или более разрядов с заполнением старших разрядов содержимым самого старшего разряда:

```
Z = X >> Y;
```

- >>> — сдвиг вправо без учета знака — сдвиг вправо на один или более разрядов с заполнением старших разрядов нулями:

```
Z = X >>> Y;
```

Как следует из названия, двоичные операторы выполняют поразрядные действия с двоичным представлением целых чисел.

3.7.4. Оператор обработки строк

+ — оператор конкатенации строк:

```
var Str = "Строка1" + "Строка2";
// Переменная Str будет содержать значение "Строка1Строка2"
```

Часто необходимо сформировать строку, состоящую из имени переменной и ее значения. Если написать

```
var X = "Строка1";
var Z = "Значение равно X";
```

то переменная `Z` будет содержать значение `"Значение равно X"`, а если так:

```
var X = "Строка1";
var Z = "Значение равно " + X;
```

то переменная `Z` будет содержать значение `"Значение равно Строка1"`. Листинг 3.8 демонстрирует вывод значения переменной в диалоговом окне.

Листинг 3.8. Вывод значения переменной

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Вывод значения переменной</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var X = "Строка1";
window.alert("Переменная X содержит значение 'X'");
// Выведет "Переменная X содержит значение 'X'"
window.alert("Переменная X содержит значение '" + X + "'");
// Выведет "Переменная X содержит значение 'Строка1'"
//-->
</script>
</body>
</html>
```

3.7.5. Приоритет выполнения операторов

В какой последовательности будет вычисляться это выражение?

```
X = 5 + 10 * 3 / 2;
```

Это зависит от приоритета выполнения операторов. В данном случае последовательность вычисления выражения будет следующей:

1. Число 10 будет умножено на 3, т. к. приоритет операции умножения выше приоритета операции сложения.
2. Полученное значение будет поделено на 2, поскольку приоритет операции деления равен операции умножения, но выше операции сложения. При равных приоритетах операции выполняются слева направо.
3. К полученному значению будет прибавлено число 5, т. к. оператор присваивания = имеет наименьший приоритет.
4. Значение будет присвоено переменной X.

С помощью скобок можно изменить последовательность вычисления выражения. Следующее выражение будет вычислено в другом порядке:

```
X = (5 + 10) * 3 / 2;
```

1. К числу 5 будет прибавлено 10.
2. Полученное значение будет умножено на 3.
3. Полученное значение будет поделено на 2.
4. Значение будет присвоено переменной X.

Перечислим операторы в порядке убывания приоритета:

- !, ~, ++, -- — отрицание, двоичная инверсия, инкремент, декремент;
- *, /, % — умножение, деление, остаток от деления;
- +, - — сложение и вычитание;
- <<, >>, >>> — двоичные сдвиги;
- & — двоичное И;
- ^ — двоичное исключающее ИЛИ;
- | — двоичное ИЛИ;
- =, +=, -=, *=, /=, %= — присваивание.

3.8. Преобразование типов данных

Что будет, если к числу прибавить строку?

```
var Str = "5";
var Number1 = 3;
var Str2 = Number1 + Str; // Переменная содержит строку "35"
var Str3 = Str + Number1; // Переменная содержит строку "53"
```

В этом случае интерпретатор столкнется с несовместимостью типов данных и попытается преобразовать переменные к одному типу данных, а затем выполнить операцию. В нашем случае переменная `Number1`, имеющая тип `number` (число), будет преобразована к типу `string` (строка), а затем будет произведена операция конкатенации строк.

А что будет, если из числа вычесть строку, число умножить на строку или число разделить на строку?

```
var Number1 = 15;
var Str = "5";
var Str2 = Number1 - Str; // Переменная содержит число 10
var Str3 = Number1 * Str; // Переменная содержит число 75
var Str4 = Number1 / Str; // Переменная содержит число 3
```

Итак, интерпретатор попытается преобразовать строку в число, а затем вычислить выражение.

Причем не важно, в какой последовательности будут указаны число и строка:

```
var Str5 = Str * Number1; // Переменная все равно содержит число 75
```

Но что будет, если в строке будут одни буквы?

```
var Number1 = 15;
var Str = "Строка";
var Str2 = Number1 - Str; // Переменная содержит значение NaN
```

В этом случае интерпретатор не сможет преобразовать строку в число и присвоит переменной значение `NaN` (Not a Number, не число).

С одной стороны, хорошо, что интерпретатор делает преобразование типов данных за нас. Но с другой стороны, можно получить результат, который вовсе не планировался. По этой причине лучше оперировать переменными одного типа, а если необходимо делать преобразования типов, то делать это самим.

Для преобразования типов данных можно использовать следующие встроенные функции JavaScript:

- `parseInt(<Строка>, [<Основание>])` — преобразует строку в целое число. Строка считается заданной в системе счисления, указанной вторым необязательным параметром. Если основание не указано, то по умолчанию используется десятичная система. Если строка не может быть преобразована в число, возвращается значение `NaN`:

```
var Number1 = 15;
var Str = "5";
var Str5 = "FF";
var Str2 = Number1 - parseInt(Str);
// Переменная содержит число 10
var Str3 = Number1 - parseInt(Str5, 16);
```

```
// Переменная содержит число -240
var Str4 = Number1 + parseInt(Str);
// Переменная содержит число 20
```

- `parseFloat(<Строка>)` — преобразует строку в число с плавающей точкой:

```
var Str = "5.2";
var Str2 = parseFloat(Str); // Переменная содержит число 5.2
```

- `eval(<Строка>)` — вычисляет выражение JavaScript, хранящееся в строке:

```
var Str = "3 + 5";
var Str2 = eval(Str); // Переменная содержит число 8
```

Приведем пример использования преобразования типов данных. Просуммируем два числа, введенных пользователем в поля двух диалоговых окон (листинг 3.9).

Листинг 3.9. Вычисление суммы двух чисел

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Вычисление суммы двух чисел</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var Str1, Str2, Sum1, Sum2, msg;
Str1 = window.prompt("Вычисление суммы двух чисел\nВведите число 1", "");
if (Str1==null) {
   document.write("Вы нажали Отмена");
}
else {
Str2 = window.prompt("Вычисление суммы двух чисел\nВведите число 2", "");
   if (Str2==null) {
      document.write("Вы нажали Отмена");
   }
   else {
     Sum1 = Str1 + Str2;
     msg = "До преобразования типов:<br>Значение суммы чисел ";
     msg += Str1 + " и " + Str2 + " равно ";
     msg += Sum1 + "<br><br>";
     Sum2 = parseInt(Str1) + parseInt(Str2);
     msg += "После преобразования типов:<br>";
     msg += "Значение суммы чисел " + Str1 + " и ";
     msg += Str2 + " равно " + Sum2;
```

```
      document.write(msg);
    }
}
//-->
</script>
</body>
</html>
```

Если в обоих диалоговых окнах набрать число 5, то в окне Web-браузера отобразится следующий текст:

```
До преобразования типов:
Значение суммы чисел 5 и 5 равно 55

После преобразования типов:
Значение суммы чисел 5 и 5 равно 10
```

Итак, диалоговые окна возвращают в качестве типа значения строку. Чтобы получить сумму двух чисел, указанных в полях диалоговых окон, пужно обязательно провести преобразование типов, иначе мы получим еще одпу строку, а не сумму.

3.9. Специальные символы. Разбиение сообщения в диалоговом окне на несколько строк

Специальные символы — это комбинации знаков, обозначающих служебные или непечатаемые символы, которые невозможно вставить обычным способом.

Именно с помощью специального символа `\n` (перевод строки) мы разбиваем сообщение в диалоговом окне на строки (листинг 3.10).

Листинг 3.10. Специальные символы

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Специальные символы</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
window.alert("Строка1\nСтрока2\n\nСтрока4");
//-->
</script>
</body>
</html>
```

Перечислим специальные символы, доступные в JavaScript:

- `\n` — перевод строки;
- `\r` — возврат каретки;
- `\f` — перевод страницы;
- `\t` — знак табуляции;
- `\'` — апостроф;
- `\"` — кавычка;
- `\\` — обратная косая черта.

3.10. Массивы

Массив — это нумерованный набор переменных. Переменная в массиве называется *элементом* массива, а ее позиция в массиве задается *индексом*. Нумерация элементов массива начинается с 0, а не с 1. Это следует помнить. Общее количество элементов в массиве называется *размером* массива.

При инициализации массива переменные указываются через запятую в квадратных скобках:

```
Mass1 = [1, 2, 3, 4];
```

Получить значение элемента массива можно, указав его индекс в квадратных скобках:

```
Str = Mass1[0]; // Переменной Str будет присвоено значение 1
```

Листинг 3.11 демонстрирует создание массива и вывод значения элемента массива в окне Web-браузера.

Листинг 3.11. Массивы

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Массивы</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var Mass1, Mass2;
Mass1 = [1, 2, 3, 4];
Mass2 = ["", "Январь", "Февраль", "Март", "Апрель", "Май", "Июнь",
         "Июль", "Август", "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"];
document.write(Mass1[1] + " и " + Mass2[2]);
//-->
```

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

При желании можно добавить новый элемент массива или изменить значение существующего:

```
Mass1[5] = 6;
Mass1[0] = 0;
```

В этом примере было создано два элемента массива и изменено значение существующего. Почему создано два элемента массива? Первый элемент с индексом 5 создан нами, а элемент с индексом 4 был создан автоматически и ему присвоено значение `undefined` (не определен), т. к. наш массив состоял только из 4 элементов, и последний определенный элемент имел индекс 3.

Любому элементу массива можно присвоить другой массив:

```
Mass1[0] = [1, 2, 3, 4];
```

В этом случае получить значение массива можно, указав два индекса:

```
Str = Mass1[0][2]; // Переменной Str будет присвоено значение 3
```

Следует учитывать, что операция присваивания сохраняет в переменной ссылку на массив, а не все его значения. Например, если попробовать сделать так

```
var Mass1, Mass2;
Mass1 = [1, 2, 3, 4];
Mass2 = Mass1; // Присваивается ссылка на массив!!!
Mass2[0] = "Новое значение";
document.write(Mass1.join(", ") + "<br>");
document.write(Mass2.join(", "));
```

то изменение `Mass2` затронет `Mass1`, и мы получим следующий результат:

```
Новое значение, 2, 3, 4
Новое значение, 2, 3, 4
```

Чтобы сделать копию массива, можно, например, воспользоваться методом `slice()`, который возвращает срез массива:

```
var Mass1, Mass2;
Mass1 = [1, 2, 3, 4];
Mass2 = Mass1.slice(0);
Mass2[0] = "Новое значение";
document.write(Mass1.join(", ") + "<br>");
document.write(Mass2.join(", "));
```

Результат:

```
1, 2, 3, 4
Новое значение, 2, 3, 4
```

Необходимо заметить, что при использовании многомерных массивов метод `slice()` создает "поверхностную" копию, а не полную:

```
var Mass1, Mass2;
Mass1 = [[0, 1], 2, 3, 4];
Mass2 = Mass1.slice(0);
Mass2[0][0] = "Новое значение1";
Mass2[1] = "Новое значение2";
```

В результате массивы будут выглядеть так:

```
Mass1 = [["Новое значение1", 1], 2, 3, 4];
Mass2 = [["Новое значение1", 1], "Новое значение2", 3, 4];
```

Как видно из примера, изменение вложенного массива в `Mass2` привело к одновременному изменению значения в `Mass1`. Иными словами, оба массива содержат ссылку на один и тот же вложенный массив.

Более подробно мы рассмотрим массивы при изучении встроенного класса `Array` *(см. разд. 3.15.5)*.

3.11. Функции. Разделение программы на фрагменты

Функция — это фрагмент кода JavaScript, который можно вызвать из любого места программы. Функция описывается с помощью ключевого слова `function` по следующей схеме:

```
function <Имя функции> ([<Параметры>]) {
   <Тело функции>
   [return <Значение>]
}
```

3.11.1. Основные понятия

Функция должна иметь уникальное имя. Для имен действуют такие же правила, что и при указании имени переменной. Для наглядности все имена функций в этой книге начинаются с `f_`.

После имени функции в круглых скобках можно указать один или несколько параметров через запятую. Параметров может вообще не быть. В этом случае указываются только круглые скобки.

Между фигурными скобками располагаются выражения JavaScript. Кроме того, функция может возвращать значение в место вызова функции. Возвращаемое значение задается с помощью ключевого слова `return`.

Пример функции без параметров:

```
function f_Alert_OK() {
   window.alert("Сообщение при удачно выполненной операции");
}
```

Пример функции с параметром:

```
function f_Alert(msg) {
   window.alert(msg);
}
```

Пример функции с параметрами, возвращающей сумму двух переменных:

```
function f_Sum(x, y) {
   var z = x + y;
   return z;
}
```

В качестве возвращаемого значения в конструкции `return` можно указывать не только имя переменной, но и выражение:

```
function f_Sum(x, y) {
   return (x + y);
}
```

В программе функции можно вызвать следующим образом:

```
f_Alert_OK();
f_Alert("Сообщение");
Var1 = f_Sum(5, 2); // Переменной Var1 будет присвоено значение 7
```

Выражения, указанные после `return <значение>;`, никогда не будут выполнены:

```
function f_Sum(x, y) {
   return (x + y);
   window.alert("Сообщение"); // Это выражение никогда не будет выполнено
}
```

Имя переменной, передающей значение функции, может не совпадать с именем переменной внутри функции:

```
function f_Sum(x, y) {
   return (x + y);
}
var Var3, Var1 = 5;
var Var2 = 2;
Var3 = f_Sum (Var1, Var2);
```

Ссылку на функцию можно сохранить в какой-либо переменной. Для этого название функции указывается без круглых скобок:

```
function test() {
   window.alert("Это функция test()");
}
var x;
x = test; // Присваиваем ссылку на функцию
x(); // Вызываем функцию test() через переменную x
```

Кроме того, функция может вообще не иметь названия. В этом случае ссылку на анонимную функцию сохраняют в переменной:

```
var x = function() { // Присваиваем ссылку на анонимную функцию
   window.alert("Сообщение");
};
x(); // Вызываем анонимную функцию через переменную x
```

Ссылку на вложенную функцию можно вернуть в качестве значения в инструкции `return`. Чтобы вызвать вложенную функцию, круглые скобки указываются два раза:

```
var x = function() { // Присваиваем ссылку на анонимную функцию
   return function() { // Возвращаем ссылку на вложенную функцию
      window.alert("Это вложенная функция");
   };
};
x()(); // Вызываем вложенную функцию через переменную x
```

3.11.2. Расположение функций внутри HTML-документа

Обычно функции принято располагать в разделе `HEAD` HTML-документа (листинг 3.12) или в отдельном файле с расширением js (листинги 3.13 и 3.14). Хотя функции могут располагаться и в разделе `BODY`.

Листинг 3.12. Функция расположена в разделе `HEAD`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Функции</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_Sum(x, y) {
   return (x + y);
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
var Var3, Var1 = 5, Var2 = 3;
Var3 = f_Sum(Var1, Var2);
document.write(Var3);
//-->
```

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

Листинг 3.13. Функция вынесена в отдельный файл script.js

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Функции</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <script type="text/javascript" src="script.js"></script>
</head>
<body>
<script type="text/javascript">
<!--
var Var3, Var1 = 5, Var2 = 3;
Var3 = f_Sum(Var1, Var2);
document.write(Var3);
//-->
</script>
</body>
</html>
```

Листинг 3.14. Содержимое файла script.js

```
function f_Sum(x, y) {
   return (x + y);
}
```

Создать файл script.js можно с помощью Блокнота.

3.11.3. Рекурсия. Вычисление факториала

Рекурсия — это возможность функции вызывать саму себя. С одной стороны, это удобно, с другой стороны, если не предусмотреть условие выхода, происходит зацикливание. Для примера приведем вычисление факториала (листинг 3.15).

Листинг 3.15. Вычисление факториала

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Вычисление факториала</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
```

```
<script type="text/javascript">
<!--
function f_Factorial(x) {
   if (x == 0 || x == 1) return 1;
   else return (x * f_Factorial(x - 1));
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
var z;
z = window.prompt("Вычисление факториала\nВведите число", "");
if (z==null) {
   document.write("Вы нажали Отмена");
}
else {
   document.write("Факториал числа " + z + " = ");
   document.write(f_Factorial(parseInt(z)));
}
//-->
</script>
</body>
</html>
```

3.11.4. Глобальные и локальные переменные

Глобальные переменные — это переменные, объявленные вне функции. Глобальные переменные видны в любой части программы, включая функции.

Локальные переменные — это переменные, объявленные внутри функции. Локальные переменные видны только внутри тела функции. Если имя локальной переменной совпадает с именем глобальной переменной, то все операции внутри функции осуществляются с локальной переменной, а значение глобальной не изменяется.

Механизм, регулирующий такое поведение, называется областью видимости переменных. Он продемонстрирован в листинге 3.16.

Листинг 3.16. Глобальные и локальные переменные

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Глобальные и локальные переменные</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
```

```
<script type="text/javascript">
<!--
function f_Sum() {
   var Var1 = 5;
   var Num1 = 1;
   document.write("Локальная переменная Var1 = " + Var1 + "<br>");
   document.write("Локальная переменная Num1 = " + Num1 + "<br>");
   document.write("Глобальная переменная Var2 = " + Var2 + "<br>");
   return Var1+Var2;
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
var Var1, Var2, Var3;
Var1 = 10;
document.write("Глобальная переменная Var1  = " + Var1 + "<br>");
Var2 = 7;
Var3 = f_Sum();
document.write("Сумма Var1 + Var2 = " + Var3 + "<br>");
document.write("Глобальная переменная Var1 осталась = ");
document.write(Var1 + "<br>");
document.write("Локальная переменная Num1 = " + typeof Num1);
document.write(" , т. е. не видна вне тела функции");
//-->
</script>
</body>
</html>
```

В окне Web-браузера получим следующий результат:

```
Глобальная переменная Var1 = 10
Локальная переменная Var1 = 5
Локальная переменная Num1 = 1
Глобальная переменная Var2 = 7
Сумма Var1 + Var2 = 12
Глобальная переменная Var1 осталась = 10
Локальная переменная Num1 = undefined , т. е. не видна вне тела функции
```

Как видно из листинга 3.16, переменная `Num1`, объявленная внутри функции `f_Sum()`, не доступна вне функции. Глобальную переменную `Var1` не затронуло объявление внутри функции одноименной локальной переменной и ее изменение. А глобальная переменная `Var2` видна внутри функции `f_Sum()`.

3.12. Условные операторы. Выполнение блоков кода только при соответствии условию

Условные операторы позволяют в зависимости от значения логического выражения выполнить отдельный участок программы или, наоборот, не выполнять его. Логические выражения возвращают только два значения: `true` (истина) или `false` (ложь).

3.12.1. Операторы сравнения

Операторы сравнения используются в логических выражениях. Перечислим их:

- `==` — равно;
- `===` — строго равно;
- `!=` — не равно;
- `!==` — строго не равно;
- `<` — меньше;
- `>` — больше;
- `<=` — меньше или равно;
- `>=` — больше или равно.

В чем отличие оператора `==` (равно) от оператора `===` (строго равно)? Дело все в том, что если используется оператор `==`, интерпретатор пытается преобразовать разные типы данных к одному и лишь затем сравнивает их. Оператор `===`, встретив данные разных типов, сразу возвращает `false` (ложь).

Кроме того, значение логического выражения можно инвертировать с помощью оператора `!` таким образом:

```
!(Var1 == Var2)
```

Если переменные `Var1` и `Var2` равны, то возвращается значение `true`, но т. к. перед выражением стоит оператор `!`, выражение вернет `false`.

Несколько логических выражений можно объединить в одно большое с помощью следующих операторов:

- `&&` — логическое И;
- `||` — логическое ИЛИ.

```
(Var1 == Var2) && (Var2 != Var3)
(Var1 == Var2) || (Var3 == Var4)
```

Первое выражение возвращает `true` только в случае, если оба выражения вернут `true`, а второе — если хотя бы одно из выражений вернет `true`.

Оператор `||` также часто используется для создания необязательных параметров в функции. Если первое выражение не может быть преобразовано в `true`, то возвращается значение второго выражения:

```
function f_print(str) {
   str = str || "Значение по умолчанию";
   window.alert(str);
}
f_print(); // "Значение по умолчанию"
f_print("Значение указано"); // "Значение указано"
```

3.12.2. Оператор ветвления *if...else*. Проверка ввода пользователя

Оператор ветвления мы уже использовали ранее в наших примерах, например, чтобы проверить, какая из кнопок диалогового окна нажата. Так как при нажатии кнопки **OK** возвращается значение `true`, то можно узнать, какая кнопка нажата, используя оператор ветвления `if...else` (листинг 3.17).

Листинг 3.17. Проверяем, какая из кнопок диалогового окна нажата

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Окно с сообщением</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
if (window.confirm("Нажмите любую кнопку")) {
   window.alert("Нажата кнопка OK");
}
else {
   window.alert("Нажата кнопка Cancel");
}
//-->
</script>
</body>
</html>
```

Обратите внимание, что логическое выражение не содержит операторов сравнения:

```
if (window.confirm("Нажмите любую кнопку")) {
```

Такая запись эквивалентна записи:

```
if (window.confirm("Нажмите любую кнопку") == true) {
```

Проверка на равенство выражения значению `true` (истина) выполняется по умолчанию.

Оператор ветвления `if...else` имеет следующий формат:

```
if (<Логическое выражение>) {
   <Блок, выполняемый, если условие истинно>
}
[else {
   <Блок, выполняемый, если условие ложно>
}]
```

Для примера напишем программу (листинг 3.18), которая проверяет, является ли введенное пользователем число четным или нет. После проверки выводится соответствующее сообщение.

Листинг 3.18. Проверка числа на четность

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Проверка числа на четность</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var x = window.prompt("Введите число", "");
if (x==null) {
   document.write("Вы нажали Отмена");
}
else {
   if ((parseInt(x))%2==0) {
      document.write("Четное число");
   }
   else {
      document.write("Нечетное число");
   }
}
//-->
</script>
</body>
</html>
```

Как видно из примера, один условный оператор можно вложить в другой. Кроме того, если блок состоит из одного выражения, фигурные скобки можно не указывать:

```
if ((parseInt(x))%2==0) document.write("Четное число");
else document.write("Нечетное число");
```

Более того, блока `else` может не быть совсем:

```
if ((parseInt(x))%2==0) document.write("Четное число");
```

3.12.3. Оператор ? Проверка числа на четность

Оператор `?` имеет следующий формат:

```
<Переменная> = (<Лог. выражение>) ? <если Истина> : <если Ложь>;
```

Перепишем нашу программу (листинг 3.18) и используем оператор `?` вместо `if...else` (листинг 3.19).

Листинг 3.19. Проверка числа на четность с помощью оператора ?

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Проверка числа на четность</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var x = window.prompt("Введите число", "");
if (x==null) {
   document.write("Вы нажали Отмена");
}
else {
   var msg = ((parseInt(x))%2==0) ? "Четное число" : "Нечетное число";
   document.write(msg);
}
//-->
</script>
</body>
</html>
```

3.12.4. Оператор выбора *switch*

Оператор выбора `switch` имеет следующий формат:

```
switch (<Переменная или выражение>) {
   case <Значение 1>:
      <Выражение 1>;
      break;
```

```
   case <Значение 2>:
      <Выражение 2>;
      break;
...
   default:
      <Выражение>;
}
```

Перепишем нашу программу и используем оператор `switch` вместо `if...else` и `?` (листинг 3.20).

Листинг 3.20. Проверка числа на четность с помощью оператора `switch`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Проверка числа на четность</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var x = window.prompt("Введите число", "");
if (x==null) {
   document.write("Вы нажали Отмена");
}
else {
   switch ((parseInt(x))%2) {
   case 0:
      document.write("Четное число");
      break;
   case 1:
      document.write("Нечетное число");
      break;
   default:
      document.write("Введенное значение не является числом");
   }
}
//-->
</script>
</body>
</html>
```

Итак, оператор `switch` позволил сделать еще одну дополнительную проверку. Ведь пользователь вместо числа мог ввести строку. А в этом случае функция `parseInt()` вернет значение `NaN` (Not a Number). Любая арифметическая операция со значением `NaN` вернет в качестве значения `NaN`. В предыдущих примерах мы не выполняли эту

проверку, и в случае ввода строки, которую невозможно преобразовать в число, функция возвращала фразу "Нечетное число". Что, согласитесь, неверно.

Вернемся к оператору `switch`. Вместо логического выражения оператор `switch` принимает переменную или выражение. В зависимости от значения переменной (или выражения) выполняется один из блоков `case`, в котором указано это значение. Если ни одно из значений не описано в блоках `case`, то выполняется блок `default`. Оператор `break` позволяет досрочно выйти из оператора выбора `switch`. Зачем это нужно? Если не указать оператор `break` в конце блока `case`, то будет выполняться следующий блок `case` вне зависимости от указанного значения. Если убрать все операторы `break` из нашего примера, то в результате (при вводе четного числа) в окне Web-браузера отобразится следующая надпись:

```
Четное числоНечетное числоВведенное значение не является числом
```

Иными словами, оператор `break` следует обязательно указывать в конце каждого блока `case`.

3.13. Операторы циклов. Многократное выполнение блока кода

Предположим, нужно вывести все числа от 1 до 100 по одному на строке. Обычным способом пришлось бы писать 100 строк кода:

```
document.write("1<br>");
document.write("2<br>");
...
document.write("100<br>");
```

При помощи циклов то же действие можно выполнить одной строкой кода:

```
for (var i=1; i<101; i++) document.write(i + "<br>");
```

Иными словами, циклы позволяют выполнить одни и те же выражения многократно.

3.13.1. Цикл *for*

Цикл `for` используется для выполнения выражений определенное число раз. Имеет следующий формат:

```
for (<Начальное значение>; <Условие>; <Приращение>) {
   <Выражения>
}
```

Здесь используются следующие конструкции:

- `<Начальное значение>` — присваивает переменной-счетчику начальное значение;
- `<Условие>` — содержит логическое выражение. Пока логическое выражение возвращает значение `true`, выполняются выражения внутри цикла;
- `<Приращение>` — задает изменение переменной-счетчика при каждой итерации.

Более формально последовательность работы цикла `for` такова:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие — если оно истинно, выполняются выражения внутри цикла, а в противном случае осуществляется выход из цикла.
3. Переменная-счетчик изменяется на величину, указанную в <Приращении>.
4. Осуществляется переход к пункту 2.

Цикл выполняется до тех пор, пока <Условие> не вернет `false`. Если этого не случится, цикл будет бесконечным.

<Приращение> может не только увеличивать значение переменной-счетчика, но и уменьшать. Выведем все числа от 100 до 1:

```
for (var i=100; i>0; i--) document.write(i + "<br>");
```

<Приращение> может изменять значение переменной-счетчика не только на единицу. Выведем все четные числа от 1 до 100:

```
for (var i=2; i<101; i+=2) document.write(i + "<br>");
```

Следует заметить, что выражение, указанное в параметре <Условие>, вычисляется на каждой итерации. Рассмотрим вывод элементов массива:

```
var Mass = [1, 2, 3];
for (var i=0; i<Mass.length; i++) {
   if (i==0) {
      Mass.push(4); // Добавляем новые элементы
      Mass.push(5); // для доказательства
   }
   document.write(Mass[i] + " ");
} // Выведет: 1 2 3 4 5
```

В этом примере мы указываем свойство `length` в параметре <Условие>, а внутри цикла (чтобы доказать вычисление на каждой итерации) добавляем новые элементы в массив. В итоге получили все элементы массива, включая новые элементы. Чтобы этого избежать, следует вычисление размера массива указать в первом параметре:

```
var Mass = [1, 2, 3];
for (var i=0, c=Mass.length; i<c; i++) {
   if (i==0) {
      Mass.push(4); // Добавляем новые элементы
      Mass.push(5); // для доказательства
   }
   document.write(Mass[i] + " ");
} // Выведет: 1 2 3
```

3.13.2. Цикл *while*

Выполнение выражений в цикле `while` продолжается до тех пор, пока логическое выражение истинно. Имеет следующий формат:

```
<Начальное значение>;
while (<Условие>) {
   <Выражения>;
   <Приращение>;
}
```

Цикл `while` работает следующим образом:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие — если оно истинно, выполняются выражения внутри цикла, а в противном случае выполнение цикла завершается.
3. Переменная-счетчик изменяется на величину, указанную в <Приращении>.
4. Осуществляется переход к пункту 2.

Выведем все числа от 1 до 100, используя цикл `while` (листинг 3.21).

Листинг 3.21. Цикл `while`

```
var i = 1;
while (i<101) {
   document.write(i + "<br>");
   i++;
}
```

> ***Внимание!***
>
> Если <Приращение> не указано, то цикл будет бесконечным.

В <Приращении> не обязательно должна быть арифметическая операция. Например, при работе с базами данных в качестве <Приращения> будет перемещение к следующей строке, а условием выхода из цикла — отсутствие новых строк в базе данных. В этом случае <Начальным значением> будет первая строка базы данных.

3.13.3. Цикл *do...while*

Выполнение выражений в цикле `do...while` продолжается до тех пор, пока логическое выражение истинно. Но в отличие от цикла `while` условие проверяется не в начале цикла, а в конце. По этой причине выражения внутри цикла `do...while` один раз обязательно выполнятся. Конструкция имеет следующий формат:

```
<Начальное значение>;
do {
   <Выражения>;
```

```
   <Приращение>;
} while (<Условие>);
```

Последовательность работы цикла `do...while`:

1. Переменной-счетчику присваивается начальное значение.
2. Выполняются выражения внутри цикла.
3. Переменная-счетчик изменяется на величину, указанную в <Приращении>.
4. Проверяется условие и, если оно истинно, осуществляется переход к нункту 2, а если нет — цикл завершается.

Выведем все числа от 1 до 100, используя цикл `do...while` (листинг 3.22).

Листинг 3.22. Цикл `do...while`

```
var i = 1;
do {
   document.write(i + "<br>");
   i++;
} while (i<101);
```

> ***Внимание!***
>
> Если <Приращение> не указано, то цикл будет бесконечным.

3.13.4. Оператор *continue*. Переход на следующую итерацию цикла

Оператор `continue` позволяет перейти на следующую итерацию цикла еще до завершения выполнения всех выражений внутри цикла. Этот оператор можно применять в любых циклах.

Выведем все числа от 1 до 100, кроме чисел от 5 до 10 включительно (листинг 3.23).

Листинг 3.23. Использование оператора `continue`

```
for (var i=1; i<101; i++) {
   if (i>4 && i<11) continue;
   document.write(i + "<br>");
}
```

3.13.5. Оператор *break*. Прерывание цикла

Оператор `break` позволяет прервать выполнение цикла досрочно.

Выведем все числа от 1 до 100 еще одним способом (листинг 3.24).

Листинг 3.24. Прерывание цикла

```
for (var i=1; true; i++) {
   if (i>100) break;
   document.write(i + "<br>");
}
```

Здесь мы указываем условие продолжения цикла, которое всегда истинно, так что цикл продолжался бы бесконечно, если бы мы не вышли из него, используя оператор `break`.

Оператор `break` прерывает выполнение цикла, а не программы, т. е. далее будет выполнено выражение, следующее сразу за циклом.

3.14. Ошибки в программе

Существуют три типа ошибок в скриптах: синтаксические, логические и ошибки времени выполнения.

3.14.1. Синтаксические ошибки

Синтаксические — это ошибки в имени оператора или функции, отсутствие закрывающей или открывающей скобок и т. д. То есть ошибки в синтаксисе языка. Как правило, интерпретатор предупредит о наличии ошибки. А программа не будет выполняться совсем.

Например, если вместо

```
document.write(i + "<br>");
```

написать

```
doument.write(i + "<br>");
```

то Web-браузер отобразит нечто подобное:

```
Error:
name: ReferenceError
message: Statement on line 5: Reference to undefined variable: doument
Backtrace:
Line 5 of inline#1 script in test.html
doument.write(i + "<br>");
```

Итак, Web-браузер предупреждает нас, что в строке 5 файла test.html содержится ошибка. Достаточно отсчитать пятую строку в исходном коде и исправить опечатку с `doument` на `document`. А затем обновить страницу.

Перечислим часто встречающиеся синтаксические ошибки:

- опечатка в имени оператора или функции;
- буква набрана в русской раскладке клавиатуры вместо латинской;

- неправильный регистр букв;
- отсутствие открывающей или закрывающей скобки (или наоборот лишние скобки);
- в цикле `for` указаны параметры через занятую, а не через точку с занятой.

3.14.2. Логические ошибки

Логические ошибки — это ошибки в логике работы программы, которые можно выявить только по результатам работы скрипта. Как правило, интерпретатор не предупреждает о наличии ошибки, и программа будет выполняться, т. к. не содержит синтаксических ошибок. Такие ошибки достаточно трудно выявить и исправить.

Предположим, необходимо вывести первые три элемента массива. Программист, забыв, что индексация массивов начинается с нуля, пишет следующий код:

```
var Mass1 = [1, 2, 3, 4];
for (var i=1; i<4; i++) document.write(Mass1[i]+ "<br>");
```

В итоге возникает логическая ошибка, т. к. будут получены не первые элементы массива, а три элемента начиная со второго. Так как в данном примере нет синтаксических ошибок, интерпретатор сочтет код правильным.

Если в логическом выражении вместо оператора == (равно) указан оператор присваивания =, то это также приведет к логической ошибке:

```
var X = 5;
if (X=6) document.write("Переменная X равна 6");
else document.write("Переменная X НЕ равна 6");
```

Этот код выведет совсем не то, что хотел программист:

```
Переменная X равна 6
```

3.14.3. Ошибки времени выполнения

Ошибки времени выполнения — это ошибки, которые возникают во время работы скрипта. Причиной являются события, не предусмотренные программистом.

В некоторых языках (например, в PHP) ошибки времени выполнения возникают из-за деления на ноль или обращения к несуществующему элементу массива. В языке JavaScript в этих случаях программа прервана не будет. При попытке деления на ноль возвращается значение `Infinity`:

```
window.alert(5/0); // Infinity
```

При обращении к несуществующему элементу массива возвращается значение `undefined`:

```
var arr = [ 1, 2];
window.alert(arr[20]); // undefined
```

Очень часто ошибки времени выполнения возникают при использовании условий:

```
if (x>5) window.alert("x > 5");
else doument.write(x + "<br>"); // Строка с ошибкой
```

В этом примере никакой ошибки не будет, пока соблюдается условие "x>5". Как только условие перестанет выполняться, сразу возникнет ошибка, и выполнение программы будет прервано.

3.14.4. Обработка ошибок

Перехватить и обработать ошибки позволяет конструкция try/catch/finally. Конструкция имеет следующий формат:

```
try {
   <Выражения, в которых перехватываем ошибки>
}
[catch ([<Ссылка на объект Error>]) {
   <Обработка ошибки>
}]
[finally {
   <Выражения, которые будут выполнены в любом случае>
}]
```

Выражения, в которых могут возникнуть ошибки, размещаются в блоке try. Если внутри этого блока возникнет исключение, то управление будет передано в блок catch. В качестве параметра в блоке catch можно указать переменную, через которую будет доступен объект Error, содержащий описание ошибки. Если в блоке try ошибки не возникло, то блок catch не выполняется. Если указан блок finally, то выражения внутри этого блока будут выполнены независимо от того, возникла ошибка или нет. Блоки catch и finally являются необязательными, но хотя бы один из них должен быть указан.

В некоторых случаях требуется не обрабатывать ошибку, а, наоборот, указать программе, что возникла неисправимая ошибка, и прервать выполнение всей программы. Для этого предназначен оператор throw:

```
if (d < 0)
   throw new Error("Переменная не может быть меньше нуля");
```

3.14.5. Модуль Firebug для Web-браузера Firefox

Firebug — это модуль для Web-браузера Firefox, предназначенный для отладки Web-страниц и скриптов. Этот инструмент будет незаменимым помощником каждому Web-мастеру. Вы сможете отлаживать и просматривать структуру HTML, CSS и JavaScript. Загрузить модуль можно с сайта разработчика (**http://getfirebug.com/**) или со страницы **https://addons.mozilla.org/ru/firefox/addon/1843**.

На вкладке **HTML** отображается весь код страницы. При наведении курсора мыши на определенный тег элемент подсвечивается на Web-странице, а справа на вкладке **Макет** видна структура блочной модели со значениями атрибутов margin, border и padding (рис. 3.1). Значения этих атрибутов можно изменять и одновременно наблюдать за результатом произведенных изменений. Это очень удобно.

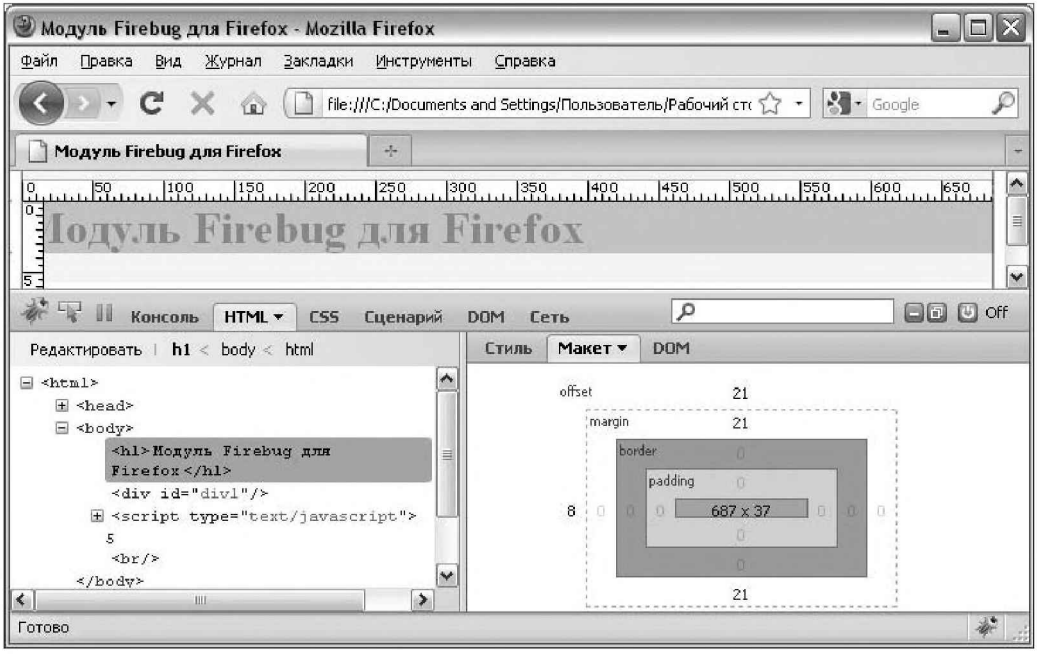


Рис. 3.1. Структура блочной модели, отображаемая на вкладке **Макет**

Следует обратить еще внимание на вкладку **Сеть**. Здесь отображается весь процесс загрузки Web-страницы. Можно узнать скорость загрузки отдельных компонентов, а также посмотреть HTTP-заголовки запроса Web-браузера и HTTP-заголовки ответа сервера.

Чтобы продемонстрировать возможности модуля для поиска ошибок в скриптах, вернемся к нашей строке с ошибкой:

```
doument.write(i + "<br>");
```

После загрузки страницы на вкладке **Консоль** появится сообщение об ошибке (рис. 3.2). Обратите внимание на то, что текст ошибки является ссылкой, при переходе по которой станет активной вкладка **Сценарнй**, а строка с ошибкой некоторое время будет подсвечена.

Вкладка **Сценарнй** является полноценным отладчиком скриптов на JavaScript. Здесь можно установить точки останова. Для этого необходимо щелкнуть мышью напротив нужной строки перед нумерацией строк. В итоге будет отображена жирная точка. Теперь после обновления Web-страницы программа прервется на отмеченной строке (рис. 3.3). В этот момент можно посмотреть текущие значения переменных, а также продолжить выполнение скрипта по шагам. Таким образом, можно полностью контролировать весь процесс выполнения программы.

Необходимо заметить, что в Web-браузере Internet Explorer 8.0 существует аналогичный инструмент. Он называется "Средства разработчика". Для запуска в меню **Сервнс** выбираем нункт **Средства разработчнка** или нажимаем клавишу <F12>. Здесь можно просматривать структуру HTML и CSS (рис. 3.4), а также отлаживать скрипты (рис. 3.5).

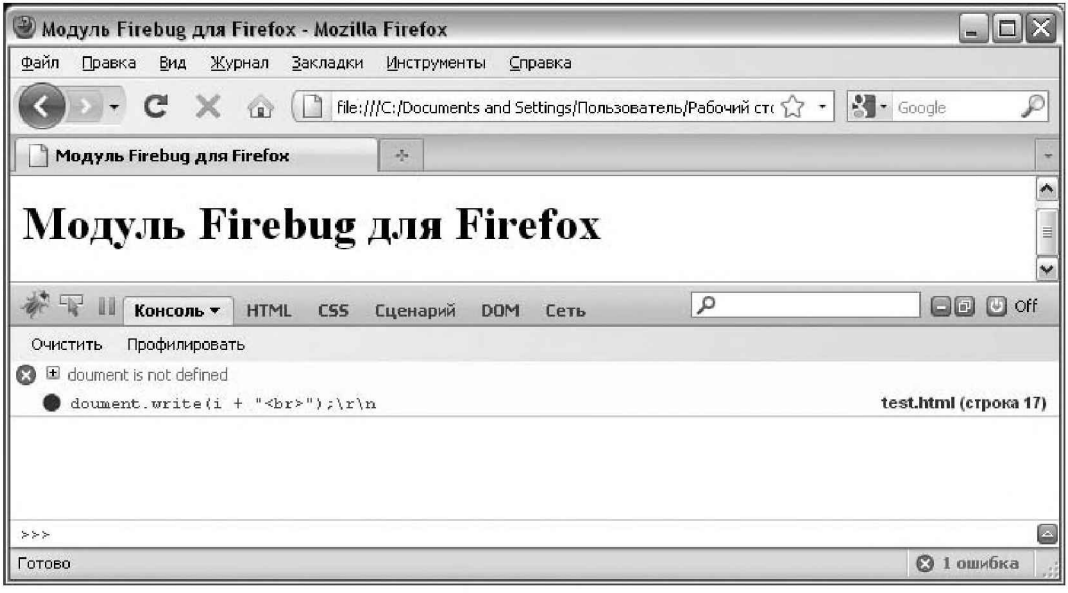


Рис. 3.2. Сообщение об ошибке, выводимое модулем Firebug

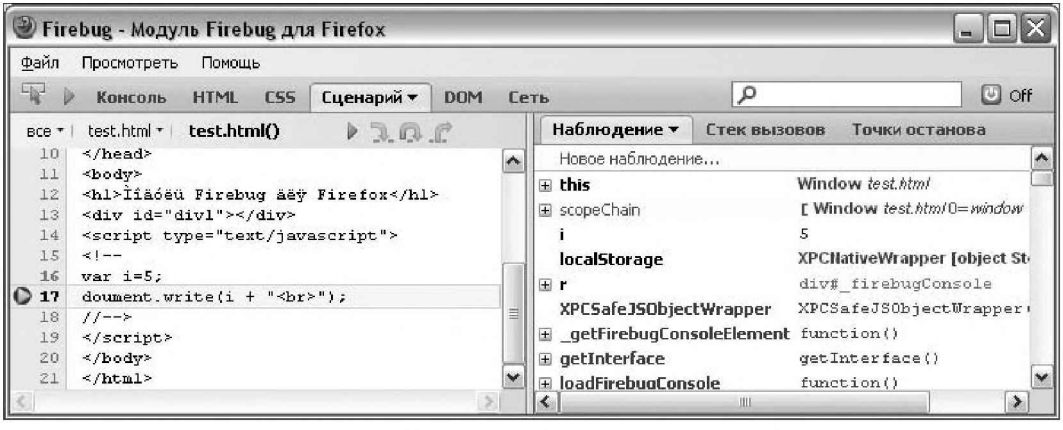


Рис. 3.3. Пошаговое выполнение программы в Firebug

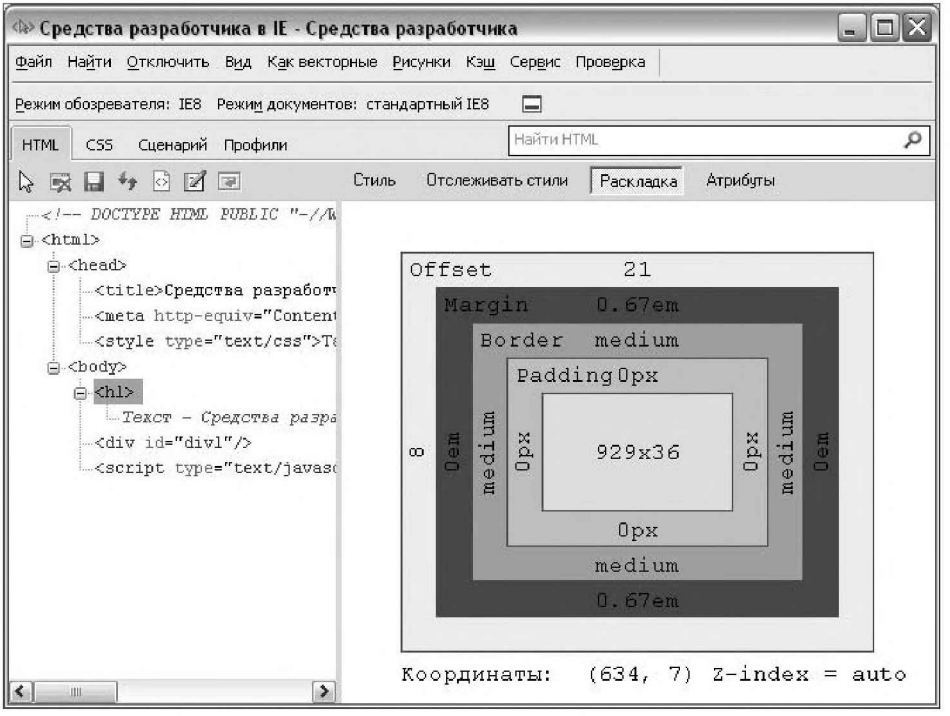


Рис. 3.4. Окно **Средства разработчика** в Web-браузере Internet Explorer 8.0

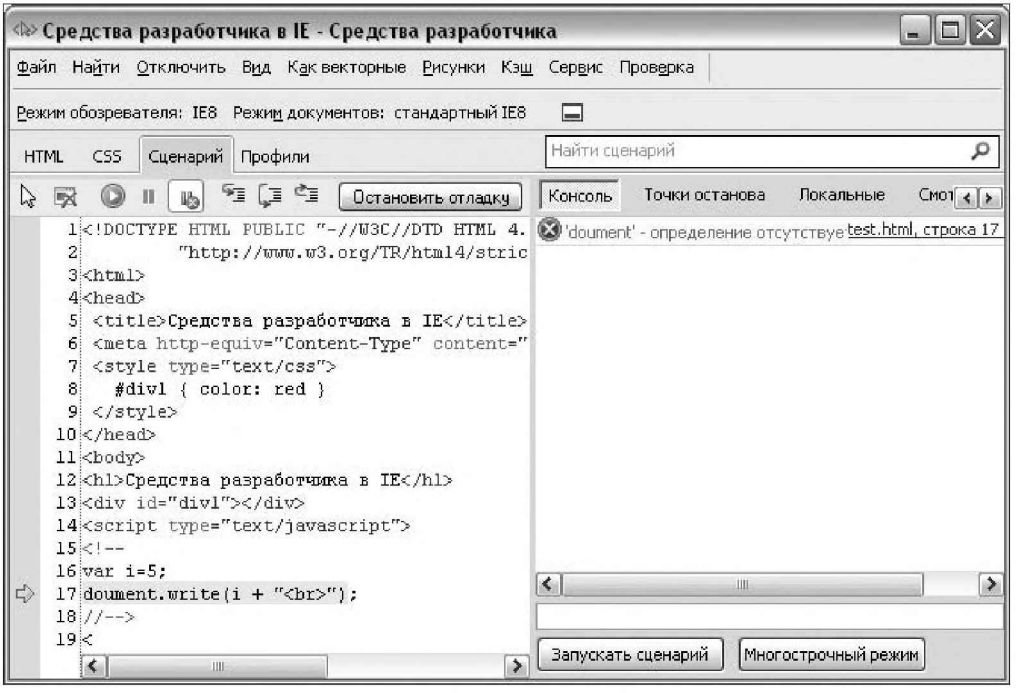


Рис. 3.5. Отладка скриптов в Web-браузере Internet Explorer 8.0

3.15. Встроенные классы JavaScript

Класс — это тип объекта, включающий в себя переменные и функции для управления этими переменными. Переменные называют *свойствами*, а функции — *методами*.

3.15.1. Основные понятия

Для использования методов и свойств класса чаще всего необходимо создать экземпляр класса. Для этого используется оператор `new`, после него указывается имя класса, к которому будет относиться данный экземпляр. После имени класса, в круглых скобках, можно передавать некоторые параметры, задавая таким образом начальные значения свойствам класса:

```
<Экземпляр класса> = new <Имя класса> ([<Параметры>]);
```

При создании экземпляра класса ссылка (указатель) сохраняется в переменной. Используя ссылку, можно обращаться к свойствам и методам созданного экземпляра класса.

При обращении к свойствам используется следующий формат:

```
<Экземпляр класса>.<Имя свойства>;
```

Обращение к методам осуществляется аналогично, только после имени метода необходимо указать круглые скобки:

```
<Экземпляр класса>.<Имя метода>();
```

В скобках часто указываются параметры метода.

3.15.2. Класс *Global*

Использование свойств и методов класса `Global` не требует создания экземпляра класса. Свойства и методы данного класса являются встроенными функциями JavaScript.

Свойства:

- `NaN` — содержит значение `NaN` (Not a Number, не число):

```
var x = NaN;
```

- `Infinity` — возвращает значение "плюс бесконечность":

```
var x = Infinity;
```

Методы:

- `parseInt(<Строка>, [<Основание>])` — преобразует строку в целое число системы счисления, заданной основанием. Если основание не указано, то по умолчанию используется десятичная система. Если строка не может быть преобразована в число, возвращается значение `NaN`.

Например:

```
var Number1 = 15;
var Str = "5";
var Str5 = "FF";
var Str2 = Number1 - parseInt(Str);
// Переменная содержит число 10
var Str3 = Number1 - parseInt(Str5, 16);
// Переменная содержит число -240
var Str4 = Number1 + parseInt(Str);
// Переменная содержит число 20
```

- `parseFloat(<Строка>)` — преобразует строку в число с плавающей точкой:

```
var Str = "5.2";
var Str2 = parseFloat(Str); // Переменная содержит число 5.2
```

- `eval(<Строка>)` — вычисляет выражение JavaScript, хранящееся в строке:

```
var Str = "3 + 5";
var Str2 = eval(Str); // Переменная содержит число 8
```

- `isNaN(<Выражение>)` — проверяет, является ли выражение правильным числом. Возвращает `true`, если значение выражения равно `NaN`, и `false`, если выражение возвращает число;
- `isFinite(<Выражение>)` — проверяет, является ли выражение конечным числом. Возвращает `true` или `false`;
- `escape(<Строка>)` — кодирует строку шестнадцатеричными кодами:

```
var Str = escape("Привет");
// Str = %u041F%u0440%u0438%u0432%u0435%u0442
```

- `unescape(<Строка>)` — декодирует строку, закодированную методом `escape()`:

```
var Str = unescape("%u041F%u0440%u0438%u0432%u0435%u0442");
// Str = Привет
```

> ***Примечание***
>
> **Функции** `escape()` **и** `unescape()` **являются устаревшими. Вместо них следует использовать функции** `encodeURI()` **и** `decodeURI()` **или** `encodeURIComponent()` **и** `decodeURIComponent()`.

- `encodeURI(<URL-адрес>)` — кодирует URL-адрес целиком:

```
var Str = "test.php?id=5&n=Николай";
window.alert(encodeURI(Str));
// test.php?id=5&n=%D0%9D%D0%B8%D0%BA%D0%BE%D0%BB%D0%B0%D0%B9
```

- `decodeURI(<Строка>)` — декодирует строку, закодированную методом `encodeURI()`;
- `encodeURIComponent(<Строка>)` — выполняет URL-кодирование строки:

```
var Str = encodeURIComponent("Строка");
// Str = %D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%B0
```

В отличие от функции `encodeURI()` заменяет все спецсимволы шестнадцатеричными кодами:

```
var Str = "test.php?name=Николай";
window.alert(encodeURIComponent(Str));
// test.php%3Fname%3D%D0%9D%D0%B8%D0%BA%D0%BE%D0%BB%D0%B0%D0%B9
```

- `decodeURIComponent(<Строка>)` — декодирует строку, закодированную методом `encodeURIComponent()`.

3.15.3. Класс *Number*. Работа с числами

Класс `Number` используется для хранения числовых величин, а также для доступа к константам. Экземпляр класса создается по следующей схеме:

```
<Экземпляр класса> = new Number (<Начальное значение>);
```

Свойства класса `Number` можно использовать без создания экземпляра класса:

- `MAX_VALUE` — максимально допустимое в JavaScript число:

```
var x = Number.MAX_VALUE; // 1.7976931348623157e+308
```

- `MIN_VALUE` — минимально допустимое в JavaScript число:

```
var x = Number.MIN_VALUE; // 5e-324
```

- `NaN` — значение `NaN`:

```
var x = Number.NaN; // NaN
```

- `NEGATIVE_INFINITY` — значение "минус бесконечность":

```
var x = Number.NEGATIVE_INFINITY; // -Infinity
```

- `POSITIVE_INFINITY` — значение "плюс бесконечность":

```
var x = Number.POSITIVE_INFINITY; // Infinity
```

Методы:

- `valueOf()` — возвращает числовое значение экземпляра класса:

```
var x = new Number (15);
var y = x.valueOf(); // 15
document.write(typeof y); // number
```

- `toString()` — возвращает строковое представление числа:

```
var x = new Number (15);
var Str = x.toString(); // "15"
document.write(typeof Str); // string
```

3.15.4. Класс *String*. Обработка строк

Класс `String` предоставляет доступ к множеству методов для обработки строк. Экземпляр класса создается по следующей схеме:

```
<Экземпляр класса> = new String (<Строка>);
```

Как вы уже знаете, создать строку можно с помощью двойных или одинарных кавычек:

```
var Str1 = "Строка 1";
var Str2 = 'Строка 2';
```

Строки, созданные этими способами, будут иметь тип данных `string`, а при создании экземпляра класса `String` тип данных будет `object`:

```
var Str1 = "Строка 1";
var Str2 = 'Строка 2';
var Str3 = new String ("Строка 3");
document.write(typeof Str1); // string
document.write(typeof Str2); // string
document.write(typeof Str3); // object !
```

Тем не менее к обычным строкам можно применять методы класса `String`:

```
var Str = "Строка".toUpperCase(); // Перевод символов в верхний регистр
document.write(Str); // "СТРОКА"
document.write(typeof Str); // string
```

При использовании метода `toUpperCase()` строка, имеющая тип данных `string`, автоматически преобразуется в экземпляр класса `String`. Затем производится изменение (в нашем случае перевод символов в верхний регистр) и возвращается строка, имеющая тип данных `string`. Таким образом, класс `String` является объектом-оберткой над элементарным типом данных `string`.

Свойство `length` возвращает длину строки в символах:

```
var Str = new String ("Hello, world");
document.write(Str.length); // 12
```

Методов у объектов класса `String` значительно больше:

- `toString()` и `valueOf()` — возвращают значение строки:

```
var Str = new String ("Hello, world");
var Str2 = Str.toString();
document.write(Str2); // "Hello, world"
document.write(typeof Str); // object
document.write(typeof Str2); // string
```

- `charAt(<Номер символа>)` — извлекает символ, номер которого указан в качестве параметра. Нумерация символов в строке начинается с нуля:

```
var Str = "Hello, world";
document.write(Str.charAt(0)); // "H"
```

- `charCodeAt(<Номер символа>)` — возвращает код символа, номер которого указан в качестве параметра. Нумерация символов в строке начинается с нуля:

```
var Str = "Hello, world";
window.alert(Str.charCodeAt(0)); // 72
```

- `fromCharCode(<Код1>,...,<КодN>)` — создает строку из указанных кодов:

```
var S = String.fromCharCode(1055, 1088, 1080, 1074, 1077, 1090);
window.alert(S); // "Привет"
```

- `toLowerCase()` — преобразует символы строки в символы нижнего регистра:

```
var Str = "Hello, world";
Str = Str.toLowerCase();
document.write(Str); // "hello, world"
```

- `toUpperCase()` — преобразует символы строки в символы верхнего регистра:

```
var Str = "Hello, world";
Str = Str.toUpperCase();
document.write(Str); // "HELLO, WORLD"
```

- `substr(<Начало фрагмента>, [<Длина фрагмента>])` — извлекает фрагмент строки заданной длины. Если второй параметр пропущен, возвращаются все символы до конца строки:

```
var Str = "Hello, world";
document.write(Str.substr(0, 5)); // "Hello"
document.write(Str.substr(7)); // "world"
```

- `substring(<Начало фрагмента>, <Конец фрагмента>)` — также извлекает фрагмент строки, заданный в этом случае номерами начального и конечного символов. Последний символ во фрагмент не включается:

```
var Str = "Hello, world";
document.write(Str.substring(7, 12)); // "world"
```

- `indexOf(<Подстрока>, [<Начальная позиция поиска>])` — возвращает номер позиции первого вхождения подстроки в текущей строке. Если второй параметр не задан, то поиск начинается с начала строки. Если подстрока не найдена, возвращается значение −1:

```
var Str = "Hello, world";
document.write(Str.indexOf("ll")); // 2
document.write(Str.indexOf("ll", 5)); // -1
```

- `lastIndexOf(<Подстрока>, [<Начальная позиция поиска>])` — определяет номер позиции последнего вхождения подстроки в текущей строке. Если второй параметр не задан, то поиск начинается с начала строки. Если подстрока не найдена, возвращается значение −1:

```
var Str = "Hello, world";
document.write(Str.lastIndexOf("o")); // 8
```

- `split(<Разделитель>, [<Лимит>])` — возвращает массив, полученный в результате разделения строки на подстроки по символу-разделителю. Если второй параметр присутствует, то он задает максимальное количество элементов в результирующем массиве:

```
var Str = "Hello, world";
var Mass = Str.split(",");
document.write(Mass[0]); // "Hello" — первый элемент массива
document.write(Mass[1]); // " world" — второй элемент массива
```

- `search(<Регулярное выражение>)` — определяет номер позиции первого вхождения подстроки, совпадающей с регулярным выражением;
- `match(<Регулярное выражение>)` — возвращает массив с результатами поиска, совпадающими с регулярным выражением;
- `replace(<Регулярное выражение>, <Текст для замены>)` — возвращает строку, которая является результатом поиска и замены в исходной строке с использованием регулярного выражения.

Примеры использования последних трех методов мы рассмотрим при изучении регулярных выражений и встроенного класса `RegExp` *(см. разд. 3.15.10)*.

3.15.5. Класс *Array*. Работа с массивами и их сортировка

Класс `Array` позволяет создавать массивы как объекты и предоставляет доступ к множеству методов для обработки массивов.

Экземпляр класса можно создать следующими способами:

```
<Экземпляр класса> = new Array (<Количество элементов массива>);
<Экземпляр класса> = new Array (<Элементы массива через запятую>);
```

Если в круглых скобках нет никаких параметров, то создается массив пулевой длины, т. е. массив, не содержащий элементов. Если указано одно число, то это число задает количество элементов массива. Если указано несколько элементов через запятую или единственное значение не является числом, то указанные значения записываются в создаваемый массив.

Обращение к элементам массива осуществляется с помощью квадратных скобок, в которых указывается индекс элемента. Нумерация элементов массива начинается с нуля:

```
var Mass = new Array("Один", "Два", "Три");
document.write(Mass[0]); // "Один"
Mass[3] = 4; // Создание нового элемента массива
document.write(Mass.join(", ")); // "Один, Два, Три, 4"
```

Свойство `length` возвращает количество элементов массива:

```
var Mass = [ "Один", "Два", "Три" ];
document.write(Mass.length + "<br>"); // 3
for (var i=0, c=Mass.length; i<c; i++) {
   document.write(Mass[i] + "<br>");
   // Выводим все элементы массива по одному на строку
}
```

Большое количество методов обеспечивают удобную работу с массивами:

- `push(<Список элементов>)` — добавляет в массив элементы, указанные в списке элементов. Элементы добавляются в конец массива. Метод возвращает новую длину массива:

```
var Mass = [ "Один", "Два", "Три" ];
document.write(Mass.push("Четвертый", "Пятый")); // 5
document.write(Mass.join(", "));
// "Один, Два, Три, Четвертый, Пятый"
```

- `unshift(<Список элементов>)` — добавляет в массив элементы, указанные в списке элементов. Элементы добавляются в начало массива:

```
var Mass = [ "Один", "Два", "Три" ];
Mass.unshift("Четвертый", "Пятый");
document.write(Mass.join(", "));
// "Четвертый, Пятый, Один, Два, Три"
```

- `concat(<Список элементов>)` — возвращает массив, полученный в результате объединения текущего массива и списка элементов. При этом в текущий массив элементы из списка не добавляются:

```
var Mass = [ "Один", "Два", "Три" ];
var Mass2 = []; // Пустой массив
Mass2 = Mass.concat("Четвертый", "Пятый");
document.write(Mass.join(", "));
// "Один, Два, Три"
document.write(Mass2.join(", "));
// "Один, Два, Три, Четвертый, Пятый"
```

- `join(<Разделитель>)` — возвращает строку, полученную в результате объединения всех элементов массива через разделитель:

```
var Mass = [ "Один", "Два", "Три" ];
var Str = Mass.join(" - ");

document.write(Str); // "Один - Два - Три"
```

- `shift()` — удаляет первый элемент массива и возвращает его:

```
var Mass = [ "Один", "Два", "Три" ];
document.write(Mass.shift()); // "Один"
document.write(Mass.join(", ")); // "Два, Три"
```

- `pop()` — удаляет последний элемент массива и возвращает его:

```
var Mass = [ "Один", "Два", "Три" ];
document.write(Mass.pop()); // "Три"
document.write(Mass.join(", ")); // "Один, Два"
```

- `sort([Функция сортировки])` — выполняет сортировку массива. Если функция не указана, будет выполнена обычная сортировка (числа сортируются по возрастанию, а символы — по алфавиту):

```
var Mass = [ "Один", "Два", "Три" ];
Mass.sort();
document.write(Mass.join(", ")); // "Два, Один, Три"
```

Если нужно изменить стандартный порядок сортировки, это можно сделать с помощью функции сортировки. Функция принимает две переменные и должна возвращать:

- `1` — если первый больше второго;
- `-1` — если второй больше первого;
- `0` — если элементы равны.

Например, стандартная сортировка зависит от регистра символов:

```
var Mass = [ "единица1", "Единый", "Единица2" ];
Mass.sort();
document.write(Mass.join(", ")); // "Единица2, Единый, единица1"
```

В результате мы получим неправильную сортировку, ведь `"Единица2"` и `"Единый"` должны стоять позже `"единица1"`. Изменим стандартную сортировку на свою сортировку без учета регистра (листинг 3.25).

Листинг 3.25. Сортировка без учета регистра

```
function f_sort(Str1, Str2) { // Сортировка без учета регистра
   var Str1_1 = Str1.toLowerCase(); // Преобразуем к нижнему регистру
   var Str2_1 = Str2.toLowerCase(); // Преобразуем к нижнему регистру
   if (Str1_1>Str2_1) return 1;
   if (Str1_1<Str2_1) return -1;
   return 0;
}

var Mass = [ "единица1", "Единый", "Единица2" ];
Mass.sort(f_sort); // Имя функции указывается без скобок
document.write(Mass.join(", ")); // "единица1, Единица2, Единый"
```

Для этого две переменные приводим к одному регистру, а затем производим стандартное сравнение. Обратите внимание, что мы не изменяем регистр самих элементов массива, т. к. работаем с их копиями.

Порядок сортировки можно изменить на противоположный (листинг 3.26), изменив возвращаемые функцией значения.

Листинг 3.26. Сортировка без учета регистра в обратном порядке

```
function f_sort(Str1, Str2) {
   // Сортировка без учета регистра в обратном порядке
   var Str1_1 = Str1.toLowerCase(); // Преобразуем к нижнему регистру
   var Str2_1 = Str2.toLowerCase(); // Преобразуем к нижнему регистру
```

```
    if (Str1_1>Str2_1) return -1;
    if (Str1_1<Str2_1) return 1;
    return 0;
}
var Mass = [ "единица1", "Единица2", "Единый" ];
Mass.sort(f_sort);
document.write(Mass.join(", ")); // "Единый, Единица2, единица1"
```

- `reverse()` — переворачивает массив. Элементы будут следовать в обратном порядке относительно исходного массива:

```
var Mass = [ "Один", "Два", "Три" ];
Mass.reverse();
document.write(Mass.join(", ")); // "Три, Два, Один"
```

- `slice(<Начало>, [<Конец>])` — возвращает срез массива, начиная от индекса `<Начало>` и заканчивая индексом `<Конец>`, но не включает элемент с этим индексом. Если второй параметр не указан, то возвращаются все элементы до конца массива:

```
var Mass1 = [ 1, 2, 3, 4, 5 ];
var Mass2 = Mass1.slice(1, 4);
window.alert(Mass2.join(", ")); // "2, 3, 4"
var Mass3 = Mass1.slice(2);
window.alert(Mass3.join(", ")); // "3, 4, 5"
```

- `splice(<Начало>, <Количество>, [<Список значений>])` — позволяет удалить, заменить или вставить элементы массива. Возвращает массив, состоящий из удаленных элементов:

```
var Mass1 = [ 1, 2, 3, 4, 5 ];
var Mass2 = Mass1.splice(2, 2);
window.alert(Mass1.join(", ")); // "1, 2, 5"
window.alert(Mass2.join(", ")); // "3, 4"
var Mass3 = Mass1.splice(1, 1, 7, 8, 9);
window.alert(Mass1.join(", ")); // "1, 7, 8, 9, 5"
window.alert(Mass3.join(", ")); // "2"
var Mass4 = Mass1.splice(1, 0, 2, 3, 4);
window.alert(Mass1.join(", ")); // "1, 2, 3, 4, 7, 8, 9, 5"
window.alert(Mass4.join(", ")); // Пустой массив
```

- `toString()` и `valueOf()` — преобразуют массив в строку. Элементы указываются через запятую без пробела:

```
var Mass = [ "Один", "Два", "Три" ];
document.write(Mass.toString()); // "Один,Два,Три"
```

Многомерные массивы

Многомерные массивы можно создать перечислением:

```
var Mass = new Array(new Array("Один", "Два", "Три"),
                     new Array("Четыре", "Пять", "Шесть"));
```

```
document.write(Mass[0][1]); // "Два"
var Mass2 = [ [ "Один", "Два", "Три" ],
              [ "Четыре", "Пять", "Шесть" ] ];
document.write(Mass2[1][1]); // "Пять"
```

или поэлементно:

```
var Mass = new Array();
Mass[0] = new Array();
Mass[1] = new Array();
Mass[0][0] = "Один";
Mass[0][1] = "Два";
Mass[0][2] = "Три";
Mass[1][0] = "Четыре";
Mass[1][1] = "Пять";
Mass[1][2] = "Шесть";
document.write(Mass[1][2]); // "Шесть"
var Mass2 = [];
Mass2[0] = [];
Mass2[1] = [];
Mass2[0][0] = "Один";
Mass2[0][1] = "Два";
Mass2[0][2] = "Три";
Mass2[1][0] = "Четыре";
Mass2[1][1] = "Пять";
Mass2[1][2] = "Шесть";
document.write(Mass2[0][0]); // "Один"
```

Обращение к элементу многомерного массива осуществляется с помощью двух индексов:

```
var Str = Mass[1][2];
```

Ассоциативные массивы. Перебор ассоциативных массивов

Основным отличием ассоциативных массивов от обычных является возможность обращения к элементу массива не по числовому индексу, а по индексу, представляющему собой строку.

```
var Mass = new Array();
Mass["Один"] = 1;
Mass["Два"] = 2;
Mass["Три"] = 3;
document.write(Mass["Один"]); // 1
```

Как вывести все элементы массива? Ни один из методов класса `Array` не позволяет вывести элементы ассоциативного массива. Кстати, свойство `length` также не работает. По этой причине перебрать все элементы массива с помощью стандартного цикла `for` не получится.

Для этой цели существует специальный цикл `for...in`. Он имеет следующий формат:

```
for (<Переменная> in <Экземпляр класса>) {
   <Тело цикла>
}
```

Цикл `for...in` на каждой итерации присваивает `<Переменной>` имя свойства, с помощью которого можно получить значение соответствующего элемента ассоциативного массива:

```
var Mass = new Array();
Mass["Один"] = 1;
Mass["Два"] = 2;
Mass["Три"] = 3;
for (var Name in Mass) {
   // Переменной Name на каждой итерации присваивается
   // строка-индекс ассоциативного массива
   document.write(Name + " = " + Mass[Name]  + "<br>");
}
```

В итоге мы получим следующий результат:

```
Один = 1
Два = 2
Три = 3
```

Ассоциативные массивы используются также для доступа к свойствам класса вместо классической точки. Для получения длины строки ранее мы обращались к свойству `length` класса `String` следующим образом:

```
var Str = "Hello, world ";
document.write(Str.length); // 13
```

С помощью ассоциативных массивов обращение к свойству `length` будет выглядеть так:

```
var Str = "Hello, world ";
document.write(Str["length"]); // 13
```

3.15.6. Класс *Math*. Использование математических функций

Класс `Math` содержит математические константы и функции. Его использование не требует создания экземпляра класса.

Свойства:

- `E` — *e*, основание натурального логарифма;
- `LN2` — натуральный логарифм 2;
- `LN10` — натуральный логарифм 10;

- `LOG2E` — логарифм по основанию 2 от *e*;
- `LOG10E` — десятичный логарифм от *e*;
- `PI` — число Пи:

```
document.write(Math.PI); // 3.141592653589793
```

- `SQRT1_2` — квадратный корень из 0,5;
- `SQRT2` — квадратный корень из 2.

Методы:

- `abs()` — абсолютное значение;
- `sin()`, `cos()`, `tan()` — стандартные тригонометрические функции (синус, косинус, тангенс). Значение указывается в радианах;
- `asin()`, `acos()`, `atan()` — обратные тригонометрические функции (арксинус, арккосинус, арктангенс). Значение возвращается в радианах;
- `exp()` — экспонента;
- `log()` — натуральный логарифм;
- `pow(<Число>, <Степень>)` — возведение `<Числа>` в `<Степень>`:

```
var x = 5;
document.write(Math.pow(x, 2)); // 25 (5 в квадрате)
```

- `sqrt()` — квадратный корень:

```
var x = 25;
document.write(Math.sqrt(x)); // 5 (квадратный корень из 25)
```

- `round()` — значение, округленное до ближайшего целого. Если первое число после запятой от 0 до 4, то округление производится к меньшему по модулю целому, а в противном случае — к большему:

```
var x = 2.499;
var y = 2.5;
document.write(Math.round(x)); // округлено до 2
document.write(Math.round(y)); // округлено до 3
```

- `ceil()` — значение, округленное до ближайшего большего целого:

```
var x = 2.499;
var y = 2.5;
document.write(Math.ceil(x)); // округлено до 3
document.write(Math.ceil(y)); // округлено до 3
```

- `floor()` — значение, округленное до ближайшего меньшего целого:

```
var x = 2.499;
var y = 2.5;
document.write(Math.floor(x)); // округлено до 2
document.write(Math.floor(y)); // округлено до 2
```

- `max(<Список чисел через запятую>)` — максимальное значение из списка:

```
document.write(Math.max(3, 10, 6)); // 10
```

- `min(<Список чисел через запятую>)` — минимальное значение из списка:

```
document.write(Math.min(3, 10, 6)); // 3
```

- `random()` — случайное число от 0 до 1:

```
document.write(Math.random()); // например, 0.9778613566886634
```

Для того чтобы получить случайное целое число от 0 до 9, нужно возвращаемое методом `random()` значение умножить на 9.9999, а затем округлить число до ближайшего меньшего целого при помощи метода `floor()`:

```
var x = Math.floor(Math.random()*9.9999);
document.write(x);
```

Попробуйте несколько раз обновить Web-страницу. Число будет меняться случайным образом в пределах от 0 до 9 включительно. Для чего это может пригодиться? Например, если есть четыре баннера 468×60, то их можно показывать случайным способом.

```
var x = Math.floor(Math.random()*3.9999);
document.write('<img src="banner' + x + '.gif" width="480" height="60">');
```

Четыре баннера с именами banner0.gif, banner1.gif, banner2.gif и banner3.gif должны быть расположены в одной папке с файлом, в котором находится исполняемый скрипт.

Названия файлов с баннерами можно сделать произвольными, добавив их в массив:

```
var Mass = [ "banner-red.gif", "banner-blue.jpeg",
             "banner-gray.gif", "banner-white.png" ];
var x = Math.floor(Math.random()*3.9999);
document.write('<img src="' + Mass[x]);
document.write('" width="480" height="60">');
```

3.15.7. Класс *Date*. Получение текущей даты и времени. Вывод даты и времени в окне Web-браузера

Класс `Date` позволяет работать с датой и временем. Создаются экземпляры класса так:

```
<Экземпляр класса> = new Date();
<Экземпляр класса> = new Date(<Число миллисекунд>);
<Экземпляр класса> = new Date(<Год>, <Месяц>, <День>, <Часы>, <Минуты>,
                              <Секунды>, <Миллисекунды>);
```

Класс поддерживает следующие методы:

- `toString()` — преобразует дату в строку и возвращает ее:

```
var d = new Date();
document.write(d.toString());
```

```
// В Opera: Fri, 30 Oct 2009 01:07:17 GMT+0300
// В Firefox: Fri Oct 30 2009 01:07:17 GMT+0300
// В IE: Fri Oct 30 01:07:17 UTC+0300 2009
```

- `toLocaleString()` — преобразует дату в строку, используя интернациональные установки системы, и возвращает ее:

```
var d = new Date();
document.write(d.toLocaleString());
// В Opera: 30.10.2009 1:11:27
// В Firefox: 30 Октябрь 2009 г. 1:11:27
// В IE: 30 октября 2009 г. 1:11:27
```

- `valueOf()` — позволяет определить число миллисекунд, прошедших с 01.01.1970 00:00:00:

```
var d = new Date();
document.write(d.valueOf());
// 1256854444062
```

- `getDate()` — возвращает день месяца (от 1 до 31):

```
var d = new Date();
document.write(d.getDate());
// 30
```

- `getDay()` — дает возможность узнать день недели (от 0 для воскресенья до 6 — для субботы):

```
var Mass = [ "воскресенье", "понедельник", "вторник",
             "среда", "четверг", "пятница", "суббота" ];
var d = new Date();
document.write(Mass[d.getDay()]);
// пятница
```

- `getMonth()` — возвращает месяц (от 0 для января до 11 — для декабря):

```
var Mass = [ "январь", "февраль", "март", "апрель", "май",
             "июнь", "июль", "август", "сентябрь", "октябрь",
             "ноябрь", "декабрь" ];
var d = new Date();
document.write(Mass[d.getMonth()]); // октябрь
```

Для получения номера текущего месяца к возвращаемому значению необходимо прибавить единицу:

```
var d = new Date();
var Month = d.getMonth() + 1;
document.write(Month); // 10
```

- `getFullYear()` — позволяет определить год:

```
var d = new Date();
document.write(d.getFullYear());
// 2009
```

- `getHours()` — возвращает час (от 0 до 23):

```
var d = new Date();
document.write(d.getHours());
// 1
```

- `getMinutes()` — позволяет получить минуты (от 0 до 59):

```
var d = new Date();
document.write(d.getMinutes());
// 23
```

- `getSeconds()` — возвращает секунды (от 0 до 59):

```
var d = new Date();
document.write(d.getSeconds());
// 20
```

- `getMilliseconds()` — возвращает миллисекунды (от 0 до 999):

```
var d = new Date();
document.write(d.getMilliseconds());
// 156
```

- `getTime()` — позволяет определить число миллисекунд, прошедших с 01.01.1970 00:00:00:

```
var d = new Date();
document.write(d.getTime());
// 1256855182843
```

Рассмотрим на примере работу с датой и временем (листинг 3.27).

Листинг 3.27. Текущая дата и время

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Текущая дата и время</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_Date(Str) {
   Str += ""; // Преобразуем число в строку
   if (Str.length==1) return ("0" + Str);
   else return Str;
}

function f_Year(Year) {
   Year += ""; // Преобразуем число в строку
   return Year.substr(2);
}
```

```
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
var d = new Date();
var msg;
var Day = [ "воскресенье", "понедельник", "вторник", "среда",
            "четверг", "пятница", "суббота" ];
var Month = [ "января", "февраля", "марта", "апреля", "мая",
              "июня", "июля", "августа", "сентября", "октября",
              "ноября", "декабря" ];
msg = "Сегодня <br>" + Day[d.getDay()] + " ";
msg += d.getDate() + " ";
msg += Month[d.getMonth()] + " ";
msg += d.getFullYear() + " ";
msg += f_Date(d.getHours()) + ":";
msg += f_Date(d.getMinutes()) + ":";
msg += f_Date(d.getSeconds()) + "<br>";
msg += f_Date(d.getDate()) + ".";
msg += f_Date(d.getMonth() + 1) + ".";
msg += f_Year(d.getFullYear());
document.write(msg);
//-->
</script>
</body>
</html>
```

В окне Web-браузера отобразится надпись

```
Сегодня
пятница 30 октября 2009 01:36:29
30.10.09
```

В другое время надпись будет иной, т. к. мы работаем с текущим временем.

В примере мы использовали две созданные нами функции:

- `f_Date(Str)` — если параметр состоит из одной цифры, то функция добавляет перед ним 0 и возвращает строку. Если не применить функцию, то дата 05.04.2008 будет выглядеть 5.4.2008, т. к. методы класса `Date` возвращают число;
- `f_Year(Year)` — функция возвращает последние две цифры года.

3.15.8. Класс *Function* (функции)

Класс `Function` позволяет использовать функцию как экземпляр класса. Делается это таким образом:

```
<Имя функции> = new Function(<Параметр1>, ... , <ПараметрN>, <Тело функции>);
```

Например, функцию суммирования двух чисел

```
function f_Sum(x, y) {
   return x + y;
}
```

можно переписать так:

```
var f_Sum = new Function ("x", "y", "return x + y");
```

Указывать тело функции в виде строки очень неудобно. По этой причине данным способом никто не пользуется.

Вместо него применяются анонимные функции:

```
var f_Sum = function(x, y) {
   return x + y;
};
```

Вызывать функцию можно так же, как и раньше:

```
document.write(f_Sum(5, 6)); // 11
```

При использовании анонимных функций следует учитывать, что при указании внутри функции глобальной переменной будет сохранена ссылка на эту переменную, а не на ее значение:

```
var x = 5;
var f_Sum = function() {
   return x; // Сохраняется ссылка, а не значение переменной x !
};
document.write(f_Sum()); // 5
x = 10; // Изменили значение
document.write(f_Sum()); // 10, а не 5
```

3.15.9. Класс *Arguments*. Функции с произвольным количеством аргументов

Класс массива аргументов позволяет получить доступ ко всем аргументам, переданным функции. Массив доступен только внутри тела функции. Получить доступ к аргументу можно, указав его индекс, а свойство `length` позволяет определить количество аргументов, переданных функции.

```
function f_Sum(x, y) {
   return arguments[0]+arguments[1];
}
document.write(f_Sum(5, 6)); // 11
```

Какой в этом смысл? Дело в том, что при использовании массива аргументов можно передать функции больше аргументов, чем первоначально объявлено. Например, можно просуммировать сразу несколько чисел, а не только два (листинг 3.28).

Листинг 3.28. Произвольное количество аргументов

```
function f_Sum(x, y) {
   var z = 0;
   for (var i=0, c=arguments.length; i<c; i++) {
      z += arguments[i];
   }
   return z;
}
document.write(f_Sum(5, 6, 7, 20)); // 38
```

3.15.10. Класс *RegExp*. Проверка значений с помощью регулярных выражений

Класс `RegExp` позволяет осуществить поиск в строке с помощью регулярных выражений — шаблонов для поиска определенных комбинаций метасимволов. Регулярные выражения позволяют осуществлять очень сложный поиск. Создать экземпляр класса `RegExp` можно двумя способами:

```
<Экземпляр класса> = new RegExp(<Регулярное выражение>[, <Модификатор>]);
<Экземпляр класса> = /<Регулярное выражение>/[<Модификатор>];
```

Необязательный параметр `<Модификатор>` задает дополнительные параметры поиска. Он может содержать следующие символы:

- `i` — поиск без учета регистра;
- `g` — глобальный поиск (поиск всех вхождений регулярного выражения в строке);
- `m` — многострочный режим. Символ `^` соответствует началу каждой подстроки, а `$` — концу каждой подстроки:

```
var p = new RegExp("^[0-9]$", "mg");
var Str = "1\n2\n3\nстрока\n4";
Mass = Str.match(p);
document.write(Mass.join(", ")); // Выведет: 1, 2, 3, 4
```

- `gi` — глобальный поиск без учета регистра символов.

При изучении класса `String` нами были оставлены без внимания три метода — `search()`, `match()` и `replace()`.

Рассмотрим способы их применения.

- `search(<Регулярное выражение>)` — возвращает номер позиции первого вхождения подстроки, совпадающей с регулярным выражением:

```
var p = new RegExp("200[14]");
var Str = "2000, 2001, 2002, 2003, 2004";
document.write(Str.search(p)); // 6
```

Шаблону `200[14]` соответствуют только два года: 2001 и 2004.

- `match(<Регулярное выражение>)` — возвращает массив с результатами поиска, совпадающими с регулярным выражением:

```
var p = new RegExp("200[14]");
var Str = "2000, 2001, 2002, 2003, 2004";
var Mass = [];
Mass = Str.match(p);
for (var i=0, c=Mass.length; i<c; i++)
   document.write(Mass[i] + "<br>");
```

Этот пример выведет только 2001, т. к. не указан модификатор глобального поиска `g`. Модифицируем его, чтобы получить все вхождения:

```
var p = new RegExp("200[14]", "g");
var Str = "2000, 2001, 2002, 2003, 2004";
var Mass = [];
Mass = Str.match(p);
for (var i=0, c=Mass.length; i<c; i++)
   document.write(Mass[i] + "<br>");
```

Теперь будут выведены все подстроки, совпадающие с регулярным выражением:

```
2001
2004
```

- `replace(<Регулярное выражение>, <Текст для замены>)` — возвращает строку, которая является результатом поиска и замены в исходной строке с использованием регулярного выражения:

```
var p = new RegExp("200[14]", "g");
var Str = "2000, 2001, 2002, 2003, 2004";
Str = Str.replace(p, "2007");
document.write(Str); // "2000, 2007, 2002, 2003, 2007"
```

В качестве второго параметра можно также указать ссылку на функцию. Через первый параметр в функции доступна строка, полностью соответствующая шаблону. Через остальные параметры доступны подвыражения, которые соответствуют фрагментам, заключенным в шаблоне в круглые скобки. В качестве примера найдем все числа в строке и прибавим к ним число 10:

```
var p = new RegExp("[0-9]([0-9]+)", "g");
var Str = "2000, 2001, 2002, 2003, 2004";
Str = Str.replace(p, function(s, x) {
   document.write(x + ", ");
   var n = parseInt(s);
   n += 10;
   return n + "";
});
document.write("<br>" + Str);
// "000, 001, 002, 003, 004, "
// "2010, 2011, 2012, 2013, 2014"
```

В строке для замены можно использовать специальные переменные `$1`, ..., `$N`, через которые доступны фрагменты, заключенные в шаблоне в круглые скобки. В качестве примера поменяем два тега местами:

```
var p = new RegExp("<([a-z]+)><([a-z]+)>");
var Str = "<br><hr>";
Str = Str.replace(p, "&lt;$2&gt;&lt;$1&gt;");
document.write(Str);
// Выведет в окне Web-браузера: "<hr><br>"
```

Метод `split(<Регулярное выражение>, [<Лимит>])` также поддерживает регулярные выражения. Он возвращает массив, полученный в результате разделения строки на подстроки по фрагменту, соответствующему регулярному выражению. Если второй параметр присутствует, то он задает максимальное количество элементов в результирующем массиве:

```
var Str = "1 2 3\n4\t5\r6";
var Mass = Str.split(/\s/);
document.write(Mass.join(", ")); // "1, 2, 3, 4, 5, 6"
var Mass2 = Str.split(/\s/, 3);
document.write(Mass2.join(", ")); // "1, 2, 3"
```

Вместо методов класса `String` можно воспользоваться методами класса `RegExp`:

- `test(<Строка>)` — возвращает `true` или `false` в зависимости от того, был поиск успешным или нет. В качестве примера проверим правильность введенной даты (листинг 3.29).

Листинг 3.29. Проверка правильности введенной даты

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Проверка вводимых данных</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var d = window.prompt("Введите дату в формате день.месяц.год", "");
if (d==null) {
   document.write("Вы нажали Отмена");
}
else {
   var p = /^[0-3]\d\.[01]\d\.\d{4}$/;
   if (p.test(d)) document.write("Дата введена правильно");
   else document.write("Вы неправильно ввели дату");
}
```

```
//-->
</script>
</body>
</html>
```

- `exec(<Строка>)` — позволяет получить массив с результатами поиска, совпадающими с регулярным выражением:

```
var p = new RegExp("(\\d{2}):(\\d{2}):(\\d{2})");
var Str = "Sun Apr 29 18:47:27 UTC+0400 2007";
var Mass = [];
Mass = p.exec(Str);
document.write(Mass.join("<br>"));
```

Эта программа выведет

```
18:47:27
18
47
27
```

Первая строка соответствует найденному фрагменту (элемент массива с индексом 0). Вторая, третья и четвертая строки содержат фрагменты, соответствующие группам метасимволов (`\\d{2}`), заключенных в круглые скобки. Номер скобок по порядку следования в регулярном выражении соответствует индексу фрагмента в массиве.

Метасимволы, используемые в регулярных выражениях. Проверка правильности ввода дат и адресов электронной почты

Как мы уже видели в приведенных ранее примерах, в регулярных выражениях присутствуют специальные символы, так называемые метасимволы. Они не всегда соответствуют отдельным символам строки, а управляют тем, как производится проверка строк. Два метасимвола позволяют осуществить привязку выражения к началу или концу строки:

- `^` — привязка к началу строки. Если указан модификатор `m`, то соответствует началу каждой подстроки;
- `$` — привязка к концу строки. Если указан модификатор `m`, то соответствует концу каждой подстроки.

Рассмотрим на примере, как действует привязка:

```
var p = new RegExp("^[0-9]+$"); // Строка может содержать только числа
var Str = "2";
if (p.test(Str)) document.write("Число"); // Выведет "Число"
else document.write("Не число");
Str = "Строка2";
```

```
if (p.test(Str)) document.write("Число");
else document.write("Не число"); // Выведет "Не число"
```

Если убрать привязку к началу и концу строки, то любая строка, содержащая число, вернет "Число":

```
var p = new RegExp("[0-9]+");
var Str = "Строка2";
if (p.test(Str)) document.write("Есть число");
// Выведет "Есть число"
else document.write("Нет числа");
```

Можно указать привязку только к началу или только к концу строки:

```
var p = new RegExp("[0-9]+$");
var Str = "Строка2";
if (p.test(Str)) document.write("Есть число в конце строки");
else document.write("Нет числа в конце строки");
// Выведет "Есть число в конце строки"
p = new RegExp("^[0-9]+");
if (p.test(Str)) document.write("Есть число в начале строки");
else document.write("Нет числа в начале строки");
// Выведет "Нет числа в начале строки"
```

Квадратные скобки `[]` позволяют указать несколько символов, которые могут встречаться на этом месте в строке. Можно перечислять символы подряд или указать диапазон через тире:

- `[09]` — соответствует числу 0 или 9;
- `[0-9]` — соответствует любому числу от 0 до 9;
- `[абв]` — соответствует буквам "а", "б" и "в";
- `[а-г]` — соответствует буквам "а", "б", "в" и "г";
- `[а-яё]` — соответствует любой букве от "а" до "я";
- `[АВС]` — соответствует буквам "А", "Б" и "С". Обратите внимание, если не указан модификатор `i`, регистр будет иметь значение;
- `[А-ЯЁ]` — соответствует любой букве от "А" до "Я";
- `[а-яёА-ЯЁ]` — соответствует любой русской букве в любом регистре;
- `[0-9а-яёА-ЯЁa-zA-Z]` — любая цифра и любая буква независимо от регистра и языка.

Значение можно инвертировать, если после первой скобки указать символ `^`. Таким способом можно указать символы, которых не должно быть на этом месте в строке:

- `[^09]` — не цифра 0 или 9;
- `[^0-9]` — не цифра от 0 до 9;
- `[^а-яёА-ЯЁa-zA-Z]` — не буква.

Вместо перечисления символов можно использовать стандартные метасимволы:

- `\d` — соответствует любой цифре;
- `\w` — соответствует любой латинской букве, цифре и знаку подчеркивания;
- `\s` — любой пробельный символ (пробел, табуляция, перевод страницы, новая строка или перевод каретки);
- . (точка) — любой символ, кроме символа перевода строки (`\n`);
- `\D` — не цифра;
- `\W` — не латинская буква, не цифра и не знак подчеркивания;
- `\S` — не пробельный символ.

> ***Внимание!***
>
> Метасимвол `\w` работает только с буквами латинского алфавита. С буквами русского языка он не работает.

Что же делать, если нужно найти точку, ведь символ "точка" соответствует любому символу, кроме символа перевода строки? Для этого перед специальным символом необходимо указать символ "\" (листинг 3.30).

Листинг 3.30. Проверка правильности введенной даты

```
var Str = "29,04.2007";
// Неправильная дата (вместо точки указана запятая)
var p = /^[0-3]\d.[01]\d.[12][09]\d\d$/;
// Символ "\" не указан перед точкой
if (p.test(Str)) document.write("Дата введена правильно");
else document.write("Дата введена неправильно");
// Выведет "Дата введена правильно", т. к. точка означает любой символ
p = /^[0-3]\d\.[01]\d\.[12][09]\d\d$/;
// Символ "\" указан перед точкой
if (p.test(Str)) document.write("Дата введена правильно");
else document.write("Дата введена неправильно");
// Выведет "Дата введена неправильно",
// т. к. перед точкой указан символ "\", а в дате присутствует запятая

p = new RegExp("^[0-3]\\d\\.[01]\\d\\.[12][09]\\d\\d$");
// Символ "\" указан перед точкой
if (p.test(Str)) document.write("Дата введена правильно");
else document.write("Дата введена неправильно");
// Выведет "Дата введена неправильно",
// т. к. перед точкой указан символ "\"
```

Обратите особое внимание на регулярное выражение в последнем примере:

```
^[0-3]\\d\\.[01]\\d\\.[12][09]\\d\\d$
```

В строке символ `\` должен заменяться на `\\`. Поэтому вместо `\d` указано `\\d`, а вместо `\.` — `\\.`. Если этого не сделать, в первом случае Web-браузер сообщит об

ошибке, а во втором случае — точка будет соответствовать любому символу, кроме символа перевода строки.

Напомним специальные символы, доступные в JavaScript, которые также доступны в регулярных выражениях:

- `\n` — перевод строки;
- `\r` — возврат каретки;
- `\f` — перевод страницы;
- `\t` — знак табуляции;
- `\v` — знак вертикальной табуляции.

Количество вхождений символа в строку задается с помощью *квантификаторов*:

- `{n}` — `n` вхождений предыдущего символа:

  `\d{2}` — последовательность из двух цифр;
- `{n,}` — `n` или более вхождений предыдущего символа:

  `\d{2,}` — последовательность из двух или более цифр;
- `{n,m}` — не менее `n` и не более `m` вхождений предшествующего символа. Цифры указываются через запятую без пробела:

  `\d{2,5}` — последовательность из двух, трех, четырех или пяти цифр;
- `*` — произвольное число вхождений предыдущего символа, в том числе ни одного вхождения:

  `\d*` — нустая строка или строка из цифр;
- `+` — одно или большее число вхождений предшествующего символа в строку:

  `\d+` — непустая строка, состоящая исключительно из цифр;
- `?` — ни одного или одно вхождение предыдущего символа в строку:

  `\d?` — цифра может встретиться один раз или не встретиться совсем.

Регулярное выражение можно разбить на подвыражения с помощью круглых скобок. Каждая группа символов, соответствующих подвыражению, сохраняется в памяти. В дальнейшем группу символов можно извлечь с помощью следующего синтаксиса:

```
\<Номер группы>
```

Нумерация групп символов осуществляется согласно их появлению в регулярном выражении. Рассмотрим пример:

```
var p = /<(.+)>(.*)<\/\1>/;
var Str = "<b><u>Подчеркнутый полужирный текст</u></b>";
var Mass = [];
Mass = p.exec(Str);
for (var i=0, c=Mass.length; i<c; i++)
   document.write(Mass[i] + "<br>");
```

Разберем регулярное выражение из этого примера:

- `<.+>` — соответствует любому открывающему тегу без параметров;
- `<(.+)>` — с помощью скобок запоминаем имя тега;
- `<\/\1>` — ищем соответствующий закрывающий тег, который был найден в первых скобках;
- `(.*)` — сохраняем группу символов между открывающим и закрывающим тегами.

В окне Web-браузера отобразится:

```
Подчеркнутый полужирный текст
b
Подчеркнутый полужирный текст
```

Первая строка соответствует найденной строке `<b><u>Подчеркнутый полужирный текст</u></b>`, вторая строка — символ в первых скобках, третья строка — группа символов во вторых скобках (`<u>Подчеркнутый полужирный текст</u>`).

С помощью круглых скобок можно объединять метасимволы в группы и применять квантификаторы ко всей группе. Рассмотрим это на примере (листинг 3.31) проверки правильности ввода адреса E-mail.

Листинг 3.31. Проверка корректности адреса электронной почты

```
var p = /^[a-z0-9_\.\-]+@([a-z0-9\-]+\.)+[a-z]{2,6}$/i;
var Str = "unicross@mail.ru";
if (p.test(Str)) document.write("E-mail правильный");
else document.write("E-mail не правильный");
```

Итак, этому шаблопу соответствует любой E-mail:

```
/^[a-z0-9_\.\-]+@([a-z0-9\-]+\.)+[a-z]{2,6}$/i
```

Сравнение производится без учета регистра. Метасимвол `^` указывает привязку к началу строки, а `$` — привязку к концу строки. E-mail разбивается на три части:

- `[a-z0-9_\.\-]+` — имя ящика, указанное до символа @;
- `([a-z0-9\-]+\.)+` — имя поддомена, указанное после символа @, но до названия зоны. Так как поддоменов может быть много, подвыражение `[a-z0-9\-]+\.` заключается в круглые скобки, после которых ставится метасимвол `+`, указывающий, что подвыражение может встречаться один и более раз;
- `[a-z]{2,6}` — название зоны может содержать только от 2 до 6 букв (ru, com, info, travel).

Логическое ИЛИ

Выражение `n|m` соответствует одному из символов `n` или `m`:

`красн(ая)|(oe)` — красная или красное, но не красный.

Глобальный класс *RegExp*. Составные части адресов электронной почты и URL-адресов

Получить результаты поиска можно с помощью свойств *глобального класса* `RegExp`:

- `$n` — возвращает `n`-пую группу символов в заданном подвыражении;
- `input` — возвращает строку, в которой был проведен поиск;
- `index` — возвращает позицию в строке найденной подстроки;
- `lastIndex` — возвращает последнюю позицию успешного поиска.

В качестве примера разберем E-mail (листинг 3.32) и URL-адрес (листинг 3.33) на составные части.

Листинг 3.32. Разбираем E-mail на составные части

```
var p =/^([a-z0-9_\.\-]+)@(([a-z0-9\-]+\.)+[a-z]{2,6})$/i;
var Str = "unicross@mail.ru";
p.exec(Str);
document.write("имя ящика - " + RegExp.$1 + "<br>");
document.write("имя сайта - " + RegExp.$2 + "<br>");
document.write("полный E-mail - " + RegExp.input + "<br>");
document.write(RegExp.index + "<br>");
document.write(RegExp.lastIndex + "<br>");
```

В итоге получим следующий результат:

```
имя ящика - unicross
имя сайта - mail.ru
полный E-mail - unicross@mail.ru
0
16
```

Листинг 3.33. Разбираем URL-адрес на составные части

```
var p = /^(\w+:\/\/)(([a-z0-9\-]+\.)+[a-z]{2,6})([a-z0-9\-\/]*\/)*([a-z0-9\-
]+\.[a-z]+)/i;
var Str = "http://www.mysite.ru/folder1/folder2/forder3/file.html";
p.exec(Str);
document.write("Полный URL - " + RegExp.input + "<br>");
document.write("Протокол - " + RegExp.$1 + "<br>");
document.write("Сайт - " + RegExp.$2 + "<br>");
document.write("Путь - " + RegExp.$4 + "<br>");
document.write("Имя файла - " + RegExp.$5 + "<br>");
```

В итоге получим результат:

```
Полный URL - http://www.mysite.ru/folder1/folder2/forder3/file.html
Протокол - http://
```

```
Сайт - www.mysite.ru
Путь - /folder1/folder2/forder3/
Имя файла - file.html
```

3.16. События

При взаимодействии пользователя с Web-страницей происходят события. События — это своего рода извещения системы о том, что пользователь выполнил какое-либо действие или внутри самой системы возникло некоторое условие.

3.16.1. Основные понятия

События возникают при щелчке на элементе, перемещении мыши, нажатии клавиши на клавиатуре, изменении размеров окна, окончании загрузки Web-страницы и т. д.

Зная, какие события может генерировать тот или иной элемент Web-страницы, можно написать функцию для обработки этого события. Например, при отправке данных формы возникает событие `onsubmit`. При наступлении этого события можно проверить данные, введенные пользователем, и, если они не соответствуют ожидаемым, прервать отправку данных.

Все названия событий начинаются с префикса `on`.

3.16.2. События мыши

Перечислим основные события мыши:

- `onmousedown` — при нажатии кнопки мыши на элементе Web-страницы или самой странице;
- `onmouseup` — при отпускании ранее нажатой кнопки мыши;
- `onclick` — при щелчке мыши на элементе Web-страницы или на самой Web-странице;
- `ondblclick` — при двойном щелчке мыши;
- `onmousemove` — при любом перемещении мыши;
- `onmouseover` — при наведении курсора мыши на элемент Web-страницы;
- `onmouseout` — при выведении курсора мыши с элемента Web-страницы;
- `onselectstart` — при начале выделения текста;
- `onselect` — при выделении элемента;
- `oncontextmenu` — при нажатии правой кнопки мыши для вывода контекстного меню.

3.16.3. События клавиатуры

Перечислим основные события клавиатуры:

- `onkeydown` — при нажатии клавиши на клавиатуре;
- `onkeypress` — аналогично событию `onkeydown`, но возвращает значение кода символа в кодировке Unicode. Наступает постоянно, пока пользователь не отпустит клавишу;
- `onkeyup` — при отпускании ранее нажатой клавиши клавиатуры;
- `onhelp` — при нажатии клавиши <F1>.

3.16.4. События документа

Перечислим основные события документа:

- `onload` — после загрузки Web-страницы;
- `onscroll` — при прокручивании содержимого элемента страницы, документа, окна или фрейма;
- `onresize` — при изменении размеров окна;
- `onbeforeunload` — перед выгрузкой документа;
- `onunload` — непосредственно перед выгрузкой документа. Наступает после события `onbeforeunload`;
- `onbeforeprint` — перед распечаткой документа или вывода его на предварительный просмотр;
- `onafterprint` — после распечатки документа или вывода его на предварительный просмотр.

3.16.5. События формы

Перечислим основные события формы:

- `onsubmit` — при отправке данных формы;
- `onreset` — при очистке формы;
- `onblur` — при потере фокуса элементом формы;
- `onchange` — при изменении данных в текстовом поле и перемещении фокуса на другой элемент формы либо при отправке данных формы (наступает перед событием `onblur`);
- `onfocus` — при получении фокуса элементом формы.

3.16.6. Последовательность событий

События возникают последовательно, например, последовательность событий при нажатии кнопкой мыши на элементе страницы будет такой:

```
onmousedown
onmouseup
onclick
```

При двойном нажатии последовательность будет такой:

```
onmousedown
onmouseup
onclick
ondblclick
```

Это значит, что событие `ondblclick` возникает после события `onclick`.

При нажатии клавиши на клавиатуре последовательность будет такой:

```
onkeydown
onkeypress
onkeyup
```

Продемонстрируем это на примере (листинг 3.34).

Листинг 3.34. Последовательность событий

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Последовательность событий</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_print(Str) {
   var div1 = document.getElementById("div1");
   div1.innerHTML += Str + "<br>";
}
//-->
</script>
</head>
<body onload="f_print('Событие onload - Страница загружена');"
 onmousedown="f_print('Событие onmousedown - Нажали');"
 onmouseup="f_print('Событие onmouseup - Отпустили');"
 onclick="f_print('Событие onclick - Щелчок');"
 onkeydown="f_print('Событие onkeydown - Нажали');"
 onkeypress="f_print('Событие onkeypress - Нажали');"
 onkeyup="f_print('Событие onkeyup - Отпустили');">
<p onmouseover="f_print('Событие onmouseover - Навели курсор');"
 onmouseout="f_print('Событие onmouseout - Убрали курсор');">
Щелкните мышью в любом месте страницы
</p><p></p>
```

```
<div id="div1"></div>
</body>
</html>
```

Данный пример позволяет наглядно увидеть последовательность событий. После загрузки возникнет событие `onload`. Щелкнем в любом месте окна, и, если не отпускать кнопку мыши, отобразится только событие `onmousedown`. Если отпустить, то возникают сразу два события: `onmouseup` и `onclick`. Если нажать любую клавишу клавиатуры и не отпускать, то возникнут сразу два события: `onkeydown` и `onkeypress`. Причем если продолжать удерживать клавишу нажатой, то событие `onkeypress` будет повторяться. Если отпустить, то возникнет событие `onkeyup`. Если навести курсор мыши на надпись "Щелкните мышью в любом месте страницы", то возникнет событие `onmouseover`. Если убрать курсор с надписи, то возникнет событие `onmouseout`.

> ***Примечание***
>
> Следует напомнить, что события возникают в такой последовательности в Web-браузере Microsoft Internet Explorer. В других Web-браузерах событийная модель может быть другой. Все примеры скриптов в этой книге написаны для Microsoft Internet Explorer, и в дальнейшем мы будем изучать объектную и событийную модель именно этого Web-браузера.

3.16.7. Всплывание событий

Что же такое "всплывание" событий? Давайте рассмотрим следующий пример (листинг 3.35).

Листинг 3.35. Всплывание событий

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Всплывание событий</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_print(Str) {
   var div1 = document.getElementById("div1");
   div1.innerHTML += Str + "<br>";
}
//-->
</script>
</head>
<body onclick="f_print('Событие onclick - Документ');">
<p onclick="f_print('Событие onclick - Абзац');">
Щелкните мышью
```

```
<span style="color: red" onclick="f_print('Событие onclick - SPAN');">
здесь</span>
</p>
<div id="div1"></div>
</body>
</html>
```

В этом примере мы написали обработчик события `onclick` для трех элементов страницы — тела документа, абзаца и тега `<span>`. Попробуем щелкнуть левой киопкой мыши на слове "здесь". В итоге вместо одного события `onclick` мы получим целую последовательность событий:

```
Событие onclick - SPAN
Событие onclick - Абзац
Событие onclick - Документ
```

Иными словами, событие `onclick` последовательно передается элементу-родителю. Для тега `<span>` элементом-родителем является абзац. А для абзаца элементом-родителем является само тело документа. Такое прохождение событий называется всплыванием событий.

Иногда всплывание событий необходимо прервать. Для этого свойству `cancelBubble` объекта `event` следует присвоить значение `true`. Кроме того, в некоторых Web-браузерах для прерывания всплывания событий можно воспользоваться методом `stopPropagation()` объекта `event`. Обратите внимание на то, что метод `stopPropagation()` не реализован в Web-браузере Internet Explorer. Продемонстрируем прерывание всплывания событий на примере (листинг 3.36).

Листинг 3.36. Прерывание всплывания событий

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Прерывание всплывания событий</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_print(Str, e) {
   var div1 = document.getElementById("div1");
   div1.innerHTML += Str + "<br>";
   e = e || window.event;
   if (e.stopPropagation) e.stopPropagation();
   else e.cancelBubble = true;
}
//-->
</script>
</head>
```

```
<body onclick="f_print('Событие onclick - Документ', event);">
<p onclick="f_print('Событие onclick - Абзац', event);">
Щелкните мышью
<span style="color: red" onclick="f_print('Событие onclick - SPAN', event);">
здесь</span>
</p>
<div id="div1"></div>
</body>
</html>
```

Попробуем теперь щелкнуть левой кнопкой мыши на слове "здесь". В итоге вместо трех событий мы получим только одно:

```
Событие onclick - SPAN
```

3.16.8. Действия по умолчанию и их отмена

Для многих событий назначены действия по умолчанию, т. е. действия, которые Web-браузер выполняет в ответ на возникшие в документе события. Например, при щелчке на гиперссылке действием по умолчанию будет переход по указанному URL-адресу, нажатие кнопки **Отправить** приводит к отправке данных формы и т. д.

Иногда действия по умолчанию необходимо прервать. Например, при отправке данных формы можно проверить их на соответствие ожидаемым и, если они не соответствуют, прервать отправку. Для этого необходимо вернуть значение `false`. Кроме возврата значения `false` для отмены действий по умолчанию можно воспользоваться методом `preventDefault()` объекта `event` или свойством `returnValue`.

В листинге 3.37 приведен пример проверки правильности ввода E-mail и прерывания перехода по гиперссылке.

Листинг 3.37. Прерывание действий по умолчанию

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Прерывание действий по умолчанию</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_test() {
   var p = /^[a-z0-9_\.\-]+@([a-z0-9\-]+\.)+[a-z]{2,6}$/i;
   // Получаем значение поля email
   var email = document.forms[0].email.value;
   if (p.test(email)) {
      if (window.confirm("Отправить данные формы?")) {
```

```
            return true; // Отправляем
        }
        else return false; // Прерываем
    }
    else {
        window.alert("E-mail введен неправильно");
        return false; // Прерываем
    }
}
function f_event(e) {
    e = e || window.event;
    if (e.preventDefault) e.preventDefault();
    else e.returnValue = false;
    window.alert("Перехода по ссылке не будет!");
}
//-->
</script>
</head>
<body>
<form action="file.php" method="GET" onsubmit="return f_test();">
<div>
E-mail:<br>
<input type="text" name="email"><br>
<input type="submit" value="Отправить">
</div>
</form>
<p>
<a href="file.html" onclick="window.alert('Перехода по ссылке не будет!');
return false;">Нажмите для перехода по ссылке</a><br><br>
<a href="file.html" onclick="f_event(event);">
Нажмите для перехода по ссылке</a>
</p>
</body>
</html>
```

3.16.9. Написание обработчиков событий

Как видно из предыдущих примеров, обработчики событий можно использовать как атрибуты тегов:

```
<span style="color: red" onclick="f_print('Событие onclick - SPAN', event);">
здесь</span>
```

Но это не единственный вариант написания обработчиков. Написать обработчик можно с помощью параметров `for` и `event` тега `<script>` (листинг 3.38). Для этого элемент Web-страницы должен иметь параметр `id`. Обратите внимание, что параметр `id` может иметь большинство тегов. В параметре `for` указывается `id` элемента

страницы, для которого создается обработчик, а в параметре `event` указывается обрабатываемое событие.

Листинг 3.38. Написание обработчиков событий

```
<!-- Работает только в Internet Explorer !!! -->
<html>
<head>
 <title>Обработчик события</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript" for="txt" event="onclick">
<!--
window.alert('Вы кликнули на слове "здесь"');
//-->
</script>
</head>
<body>
<p>Щелкните мышью
<span style="color: red" id="txt">здесь</span>
</p>
</body>
</html>
```

Можно назначить обработчик с помощью указателя функции (листинг 3.39). Нужно отметить, что имя функции обязательно должно быть указано без скобок и дополнительных атрибутов.

Листинг 3.39. Обработчик с помощью указателя функции

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Обработчик события</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_click(e) {
   e = e || window.event;
   window.alert('Вы кликнули на слове "здесь"');
   // this - это ссылка на элемент, вызвавший событие
   this.innerHTML = "новый текст";
   // Прерывание всплывания событий
   if (e.stopPropagation) e.stopPropagation();
   else e.cancelBubble = true; // Для IE
}
```

```
//-->
</script>
</head>
<body>
<p onclick="window.alert('Событие onclick - Абзац');">
Щелкните мышью
<span style="color: red" id="txt">здесь</span>
</p>
<script type="text/javascript">
<!--
// Обратите внимание: название функции указывается без скобок !!!
document.getElementById("txt").onclick = f_click;
//-->
</script>
</body>
</html>
```

Кроме того, обработчик можно написать, используя анонимную функцию (листинг 3.40).

Листинг 3.40. Обработчик с использованием анонимной функции

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Обработчик события</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<p onclick="window.alert('Событие onclick - Абзац');">
Щелкните мышью
<span style="color: red" id="txt">здесь</span>
</p>
<script type="text/javascript">
<!--
// Использование анонимной функции
document.getElementById("txt").onclick = function(e) {
   e = e || window.event;
   window.alert('Вы кликнули на слове "здесь"');
   // this - это ссылка на элемент, вызвавший событие
   this.innerHTML = "новый текст";
   // Прерывание всплывания событий
   if (e.stopPropagation) e.stopPropagation();
   else e.cancelBubble = true;
}
```

```
//-->
</script>
</body>
</html>
```

Назначить обработчик события в модели DOM Level 2 позволяет метод `addEventListener()`. Формат метода:

```
<Элемент>.addEventListener(<Событие>, <Ссылка на функцию>, <Перехват>);
```

Удалить обработчик события можно с помощью метода `removeEventListener()`. Формат метода:

```
<Элемент>.removeEventListener(<Событие>, <Ссылка на функцию>,
                              <Перехват>);
```

В параметре `<Событие>` указывается название события без префикса "on", например, `click` вместо `onclick`. Ссылка на функцию-обработчик указывается во втором параметре. В эту функцию в качестве аргумента передается ссылка на объект `event`, а внутри функции через ключевое слово `this` доступна ссылка на текущий элемент. Если в параметре `<Перехват>` указать значение `true`, то событие будет перехватываться на этапе всплывания от вложенных элементов, а если `false` — то обрабатывается событие самого элемента. Чтобы понять смысл этого параметра, рассмотрим пример:

```
<div><span id="span1">span1
   <span id="span2">Щелкните здесь (span2)</span></span>
</div>
<script type="text/javascript">
function f_click(e) { // e - ссылка на объект event
   window.alert("Элемент " + this.getAttribute("id") +
      ". Событие возникло в " + e.target.getAttribute("id"));
}
if (document.addEventListener) { // В IE не работает
   var span1 = document.getElementById("span1");
   var span2 = document.getElementById("span2");
   span1.addEventListener("click", f_click, true);
   span2.addEventListener("click", f_click, false);
}
</script>
```

При щелчке на фразе "Щелкните здесь" возникнет последовательность событий:

```
Элемент span1. Событие возникло в span2
Элемент span2. Событие возникло в span2
```

Таким образом, событие, возникшее во вложенном элементе, вначале обрабатывается элементом-родителем, а затем самим элементом. Если заменить `true` на `false`, то последовательность будет другой:

```
Элемент span2. Событие возникло в span2
Элемент span1. Событие возникло в span2
```

Это нормальная последовательность всплывания событий, которую мы рассматривали в *разд. 3.16.7*. Именно значение `false` используется в большинстве случаев.

В качестве еще одного примера рассмотрим назначение обработчика для всех кнопок (`type="button"`), а также реализацию обработчика, отрабатывающего только один раз:

```
<input type="button" id="btn1" value="Кнопка 1">
<input type="button" id="btn2" value="Кнопка 2">
<script type="text/javascript">
function f_click1(e) { // e - ссылка на объект event
   // Сработает при каждом щелчке на любой кнопке
   window.alert("Обработчик 1. Кнопка " + e.target.getAttribute("id"));
}
function f_click2() { // Сработает только 1 раз
   window.alert("Обработчик 2");
   // Удаление обработчика
   // this - ссылка на текущий элемент
   this.removeEventListener("click", f_click2, false);
}
if (document.addEventListener) { // В IE не работает
   var tags = document.getElementsByTagName("input");
   for (var i=0, len=tags.length; i<len; i++) {
      if (tags[i].type=="button")
         tags[i].addEventListener("click", f_click1, false);
   }
   var elem = document.getElementById("btn1");
   elem.addEventListener("click", f_click2, false);
}
</script>
```

Web-браузер Internet Explorer не поддерживает методы `addEventListener()` и `removeEventListener()`. Для назначения обработчика в этом Web-браузере, начиная с пятой версии, предназначен метод `attachEvent()`. Формат метода:

```
<Элемент>.attachEvent(<Событие>, <Ссылка на функцию>);
```

Удалить обработчик события можно с помощью метода `detachEvent()`. Формат метода:

```
<Элемент>.detachEvent(<Событие>, <Ссылка на функцию);
```

В параметре `<Событие>` указывается название события с префиксом "on", например, `onclick`. Ссылка на функцию-обработчик указывается во втором параметре. В эту функцию в качестве аргумента передается ссылка на объект `event`. Обратите внимание на то, что внутри функции ключевое слово `this` ссылается на объект `window`, а не на текущий элемент.

Переделаем наш предыдущий пример и используем методы `attachEvent()` и `detachEvent()` для назначения и удаления обработчиков:

```
<input type="button" id="btn1" value="Кнопка 1">
<input type="button" id="btn2" value="Кнопка 2">
<script type="text/javascript">
function f_click1(e) {
   // Сработает при каждом щелчке на любой кнопке
   window.alert("Обработчик 1. Кнопка " + e.srcElement.id);
}
function f_click2() { // Сработает только 1 раз
   window.alert("Обработчик 2");
   // Удаление обработчика
   var elem = document.getElementById("btn1");
   elem.detachEvent("onclick", f_click2);
}
if (document.attachEvent) { // Работает в IE 5+, Opera 9.02
   var tags = document.getElementsByTagName("input");
   for (var i=0, len=tags.length; i<len; i++) {
      if (tags[i].type=="button")
         tags[i].attachEvent("onclick", f_click1);
   }
   var elem = document.getElementById("btn1");
   elem.attachEvent("onclick", f_click2);
}
</script>
```

До пятой версии в Internet Explorer можно назначать обработчики только как параметры тегов или присваиванием ссылки на функцию свойству-обработчику элемента документа. В этом случае объект `event` не передается в качестве параметра. Вместо него следует использовать глобальное свойство `event` объекта `window`.

Для примера рассмотрим кроссбраузерный вариант назначения обработчика для события `onload`:

```
function f_load(e) {
   var e = e || window.event; // Объект event
   window.alert("Событие onload");
}
if (window.addEventListener) { // DOM Level 2
   window.addEventListener("load", f_load, false);
}
else if (window.attachEvent) { // IE 5+
   window.attachEvent("onload", f_load);
}
else window.onload = f_load; // IE 4-
```

3.16.10. Объект *event*. Вывод координат курсора и кода нажатой клавиши. Вывод сообщений при нажатии комбинации клавиш

Объект `event` позволяет получить детальную информацию о произошедшем событии и выполнить необходимые действия. Объект `event` доступен только в обработчиках событий. При наступлении следующего события все предыдущие значения свойств сбрасываются.

Объект `event` имеет следующие свойства:

- `srcElement` — ссылка на элемент, который является источником события. В модели DOM Level 2 используется свойство `target`;
- `currentTarget` — в модели DOM Level 2 возвращает ссылку на элемент, в котором обрабатывается событие. Ссылается на тот же элемент, что и ключевое слово `this` внутри обработчика события. Значение свойства `currentTarget` может не совпадать со значением свойства `target`;
- `type` — строка, содержащая тип события. Возвращается в нижнем регистре и без префикса `on`. Например, при событии `onclick` свойство `type` равно `click`;
- `clientX` и `clientY` — координаты события (по осям X и Y) в клиентских координатах;
- `screenX` и `screenY` — координаты события (по осям X и Y) относительно окна;
- `offsetX` и `offsetY` — координаты события (по осям X и Y) относительно контейнера;
- `x` и `y` — координаты события по осям X и Y. В модели DOM Level 2 этих свойств нет;
- `button` — число, указывающее нажатую кнопку мыши. Может принимать следующие значения:
  - `0` — кнопки не были нажаты;
  - `1` — нажата левая кнопка мыши;
  - `2` — нажата правая кнопка мыши;
  - `3` — левая и правая кнопки мыши были нажаты одновременно;
  - `4` — нажата средняя кнопка.

  В модели DOM Level 2 значения другие:
  - `0` — нажата левая кнопка мыши;
  - `1` — нажата средняя кнопка;
  - `2` — нажата правая кнопка мыши;
- `keyCode` — код нажатой клавиши клавиатуры. В Web-браузере Firefox при нажатии обычной клавиши в обработчике события `onkeypress` свойство `keyCode` имеет значение 0, а код символа доступен через свойство `charCode`. Если нажата только

функциональная клавиша, то ситуация другая — свойство `charCode` имеет значение 0, а код символа доступен через свойство `keyCode`;

- `altKey` — `true`, если в момент события была нажата клавиша <Alt>;
- `altLeft` — `true`, если была нажата левая клавиша <Alt>, и `false`, если правая. В модели DOM Level 2 этого свойства нет;
- `ctrlKey` — `true`, если была нажата клавиша <Ctrl>;
- `ctrlLeft` — `true`, если была нажата левая клавиша <Ctrl>, и `false`, если правая. В модели DOM Level 2 этого свойства нет;
- `shiftKey` — `true`, если была нажата клавиша <Shift>;
- `shiftLeft` — `true`, если была нажата левая клавиша <Shift>, и `false`, если правая. В модели DOM Level 2 этого свойства нет;
- `cancelBubble` — указывает, будет ли событие передаваться по иерархии объектов или нет. Для прерывания всплывания событий необходимо этому свойству присвоить значение `true`. Пример использования этого свойства мы рассматривали при изучении всплывания событий *(см. разд. 3.16.7)*. В модели DOM Level 2 используется метод `stopPropagation()`;
- `returnValue` — задает, будет ли выполняться действие по умолчанию для элемента страницы. Для прерывания действия по умолчанию необходимо этому свойству присвоить значение `false` (листинг 3.41). В модели DOM Level 2 используется метод `preventDefault()`.

Листинг 3.41. Прерывание действия по умолчанию

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Прерывание действия по умолчанию</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_print(Str, e) {
   window.alert(Str);
   e = e || window.event;
   if (e.preventDefault) e.preventDefault();
   else e.returnValue = false;
}
//-->
</script>
</head>
<body>
<p>
<a href="file.html"
```

```
onclick="f_print('Перехода по ссылке не будет!', event);">
Нажмите для перехода по ссылке</a><br><br>
<a href="file.html" onclick="window.alert('Перехода по ссылке не будет!');
return false;">Нажмите для перехода по ссылке</a></p>
</body>
</html>
```

В этом примере рассмотрены два метода прерывания действия по умолчанию. В первой ссылке прерывание действия по умолчанию реализовано с помощью свойства `returnValue` объекта `event`. Во второй ссылке прерывание осуществляется возвратом значения `false`;

- `fromElement` — ссылка на элемент, с которого переместился курсор мыши. В модели DOM Level 2 используется свойство `relatedTarget`;
- `toElement` — ссылка на элемент, на который пользователь перемещает курсор мыши. В модели DOM Level 2 используется свойство `relatedTarget`;
- `repeat` — `true`, если событие `onkeypress` наступило повторно в результате удержания клавиши нажатой. В модели DOM Level 2 этого свойства нет;
- `propertyName` — имя атрибута тега, стиля или свойства элемента страницы, значение которого изменилось. В модели DOM Level 2 этого свойства нет.

Пример использования свойств объекта `event` приведен в листинге 3.42.

Листинг 3.42. Выводим координаты курсора и код нажатой клавиши

```
<!-- Работает только в Internet Explorer !!! -->
<html>
<head>
 <title>Координаты курсора и код нажатой клавиши</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_unload() {
   event.returnValue = "Хотите покинуть документ?";
}
function f_body() {
   switch (event.type) {
   case "mousemove":
      var m1, m2;
      m1 = event.clientX;
      m2 = event.clientY;
      var div1 = document.getElementById("div1");
      div1.innerHTML = "clientX, clientY: x - " + m1 + " y - " + m2;
      m1 = event.screenX;
      m2 = event.screenY;
      var div2 = document.getElementById("div2");
```

```
      div2.innerHTML = "screenX, screenY: x - " + m1 + " y - " + m2;
      m1 = event.offsetX;
      m2 = event.offsetY;
      var div3 = document.getElementById("div3");
      var div4 = document.getElementById("div4");
      var div5 = document.getElementById("div5");
      div3.innerHTML = "offsetX, offsetY: x - " + m1 + " y - " + m2;
      div4.innerHTML = "x, y: x - " + event.x + " y - " + event.y;
      div5.innerHTML = "Тег: " + event.srcElement.tagName;
      break;
   case "keypress":
      var div6 = document.getElementById("div6");
      div6.innerHTML = "код нажатой клавиши - " + event.keyCode;
      if (event.ctrlLeft && event.keyCode==10) {
         window.alert("Нажата левая клавиша Ctrl + Enter");
      }
      if (!event.ctrlLeft && event.keyCode==10) {
         window.alert("Нажата правая клавиша Ctrl + Enter");
      }
      break;
   case "contextmenu":
      event.returnValue = false;
      break;
   case "selectstart":
      event.returnValue = false;
      break;
   }
}
//-->
</script>
</head>
<body onkeypress="f_body();" onmousemove="f_body();"
onbeforeunload="f_unload();">
<p oncontextmenu="f_body();">
Над этим абзацем нельзя вывести контекстное меню</p>
<div id="div5"></div>
<div id="div1"></div>
<div id="div2"></div>
<div id="div3"></div>
<div id="div4"></div>
<div id="div6">Нажмите клавишу на клавиатуре,
чтобы увидеть ее код</div>
<p>Нажмите левую или правую клавишу Ctrl вместе с Enter,
чтобы увидеть сообщение</p>
<p onselectstart="f_body();">Этот абзац нельзя выделить
отдельно от других абзацев</p>
```

```
<div><a href="file.html">Нажмите для перехода
по ссылке</a></div>
</body>
</html>
```

Итак, почти все события документа обрабатываются одной функцией `f_body()`. С помощью свойства `type` объекта `event` и оператора выбора `switch` можно определить, какое событие произошло, и обработать его. При щелчке правой кнопкой мыши на первом абзаце не выводится контекстное меню. При перемещении курсора мыши можно наблюдать изменение координат, а при нажатии клавиши на клавиатуре выводится ее код, зная который можно обработать нажатие. Например, если одновременно нажать клавишу <Ctrl> слева и <Enter>, выводится одно сообщение, а если <Ctrl> справа и <Enter> — другое. Последний абзац нельзя выделить отдельно от других абзацев, но если начать выделение с другого абзаца, то будет выделен и последний абзац.

Теперь попробуйте перейти по ссылке или закрыть окно Web-браузера. Появится окно с запросом. Событие `onbeforeunload`, возникающее перед выгрузкой документа, позволяет вывести стандартное диалоговое окно с двумя кнопками. Окно может содержать текст, указанный в качестве значения свойства `returnValue`. Да! Свойству `returnValue` можно присвоить не только значение `true` или `false`, но и строку, которая отобразится в диалоговом окне. Но это справедливо только для события `onbeforeunload`.

```
event.returnValue = "Хотите покинуть документ?";
```

Нужно еще раз напомнить, что это справедливо только для Microsoft Internet Explorer. Например, в Web-браузере Opera половина свойств не работает. Поэтому для написания скриптов, которые будут правильно работать во всех Web-браузерах, приходится писать код под каждый Web-браузер отдельно. Даже если брать Microsoft Internet Explorer, и то следует учитывать его версию. Как определить программно, какой Web-браузер использует пользователь, мы рассмотрим в *разд. 3.17.6*.

3.17. Объектная модель Microsoft Internet Explorer

Объектная модель браузера — это совокупность объектов, обеспечивающих доступ к содержимому Web-страницы и ряду функций Web-браузера. Доступ к объектам позволяет управлять содержимым Web-страницы уже после ее загрузки.

3.17.1. Структура объектной модели

Объектная модель представлена в виде иерархии объектов. То есть имеется объект верхнего уровня и подчиненные ему объекты. В свою очередь подчиненные объекты имеют свои подчиненные объекты. Кроме того, все объекты имеют свойства, а некоторые еще и методы.

Доступ к подчиненным объектам осуществляется путем указания пути от верхнего объекта к подчиненному через точку.

```
<Объект верхнего уровня>.<Подчиненный объект>.{Свойство или метод}
```

Часто объект верхнего уровня (и даже подчиненный объект) можно не указывать. Давайте в качестве примера рассмотрим выражение для вызова диалогового окна с сообщением. Это окно мы не раз использовали для вывода результата работы скрипта:

```
window.alert("Сообщение");
```

Здесь `window` — это объект самого верхнего уровня, представляющий сам Web-браузер, а `alert()` — это метод объекта `window`. В этом случае указывать объект не обязательно, т. к. объект `window` подразумевается по умолчанию:

```
alert("Сообщение");
```

Кстати, мы не раз опускали упоминание объекта верхнего уровня. Например, при печати сообщения в окне Web-браузера:

```
document.write("Сообщение");
```

Поскольку объект `document` является подчиненным объекту `window`, то нужно было бы написать так:

```
window.document.write("Сообщение");
```

Помимо уже упомянутого объекта самого высокого уровня — `window` в объектной модели имеются следующие основные объекты Microsoft Internet Explorer:

- `event` — предоставляет информацию, связанную с событиями. Мы уже рассматривали его при изучении событийной модели *(см. разд. 3.16.10)*;
- `frame` — служит для работы с фреймами (коллекция `frames`);
- `history` — предоставляет доступ к списку истории Web-браузера;
- `navigator` — содержит информацию о Web-браузере;
- `location` — содержит URL-адрес текущей Web-страницы;
- `screen` — служит для доступа к характеристикам экрана компьютера пользователя;
- `document` — служит для доступа к структуре, содержанию и стилю документа:
  - `all` — коллекция всех элементов;
  - `anchors` — коллекция "якорей", заданных тегом `<a>`;
  - `forms` — коллекция всех форм;
    - `elements` — коллекция элементов формы;
  - `frames` — все фреймы;
  - `images` — коллекция всех изображений;
  - `links` — коллекция ссылок;

- `scripts` — коллекция скриптов;
- `styleSheets` — коллекция стилей.

3.17.2. Объект *window*. Вывод сообщения в строку состояния Web-браузера

Объект `window` — это объект самого верхнего уровня, представляющий сам Web-браузер. Объект `window` подразумевается по умолчанию, поэтому указывать ссылку на объект не обязательно.

Свойства объекта `window`:

- `defaultStatus` — сообщение, выводимое по умолчанию в строке состояния;

```
window.defaultStatus = "Текст по умолчанию в строке состояния";
```

- `status` — сообщение, отображаемое в строке состояния;

```
window.status = "Текст в строке состояния";
```

- `length` — число фреймов, отображаемых в данном окне. Для примера создадим документ с фреймами и выведем в диалоговом окне количество фреймов. Для этого создадим основной документ с фреймовой структурой (листинг 3.43) и два файла — doc1.html (листинг 3.44) и doc2.html (листинг 3.45).

Листинг 3.43. HTML-документ, описывающий фреймовую структуру (test.html)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
                "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>Заголовок</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<frameset rows="100, *">
 <frame src="doc1.html" name="frame1">
 <frame src="doc2.html">
</frameset>
</html>
```

Листинг 3.44. HTML-документ фрейма 1 (doc1.html)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Заголовок</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
```

```
<body>
<h3>Фрейм 1</h3>
</body>
</html>
```

Листинг 3.45. HTML-документ фрейма 2 (doc2.html)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Заголовок</title>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<h3>Фрейм 2</h3>
<script type="text/javascript">
<!--
window.alert(top.length); // Выведет 2
window.alert(window.length); // Выведет 0
window.alert(top.frame1.length); // Выведет 0
//-->
</script>
</body>
</html>
```

Откроем в Web-браузере файл test.html. В итоге появится окно с количеством фреймов основного окна (2). Обратите внимание, как мы обратились к свойству `length` объекта `window` из документа фрейма 2:

```
top.length
```

Что же такое `top`? Это свойство объекта `window`, возвращающее ссылку на самое верхнее родительское окно. Закроем диалоговое окно с первым сообщением. После этого появится еще одно окно с числом 0, которое возвращается свойством `window.length`.

Но почему, указав объект `window`, представляющий окно Web-браузера, мы получили количество фреймов HTML-документа doc2.html? Все дело в том, что в этом случае `window` — это не объект, а лишь свойство одноименного объекта, которое возвращает ссылку на текущее окно. Все свойства, возвращающие ссылку, описаны далее. Последнее диалоговое окно, также отображающее 0, демонстрирует возможность доступа к фрейму 1 из фрейма 2:

```
top.frame1.length
```

- `parent` — ссылка на родительское окно;
- `self` — ссылка на текущее окно;

- top — ссылка на самое верхнее родительское окно;
- window — то же, что self, ссылка на себя;
- opener — ссылка на окно, открывшее данное;
- name — имя окна или фрейма;
- closed — позволяет определить, открыто или закрыто окно. Возвращает true — если открыто, и false — если закрыто;
- screenLeft — горизонтальная координата левого верхнего угла окна. В Web-браузере Firefox нет свойства screenLeft. Вместо него используется свойство screenX;
- screenTop — вертикальная координата левого верхнего угла окна. В Web-браузере Firefox нет свойства screenTop. Вместо него используется свойство screenY;
- clientInformation — браузер (объект navigator).

В некоторых Web-браузерах (например, в Firefox и Opera) определены дополнительные свойства:

- pageXOffset — число пикселов, на которое документ прокручен вправо. В Web-браузере Internet Explorer нет свойства pageXOffset. Вместо него можно использовать свойство scrollLeft объекта document;
- pageYOffset — число пикселов, на которое документ прокручен вниз. В Web-браузере Internet Explorer нет свойства pageYOffset. Вместо него можно использовать свойство scrollTop объекта document. Пример:

```
// Для Firefox и некоторых других Web-браузеров
var msg = "pageXOffset " + window.pageXOffset + "\n";
msg += "pageYOffset " + window.pageYOffset + "\n";
// Для IE >= 6, если есть тег <!DOCTYPE>
msg += "scrollLeft " + document.documentElement.scrollLeft;
msg += "\n"+"scrollTop "+document.documentElement.scrollTop;
// Для IE < 6 и для IE >= 6, если нет тега <!DOCTYPE>
msg += "\n" + "scrollLeft " + document.body.scrollLeft + "\n";
msg += "scrollTop " + document.body.scrollTop;
window.alert(msg);
```

- innerWidth — ширина окна в пикселах. Размер не включает ширину полос прокрутки. В Web-браузере Internet Explorer нет свойства innerWidth. Вместо него можно использовать свойство clientWidth объекта document;
- innerHeight — высота окна в пикселах. Размер не включает высоту строки меню и панели инструментов. В Web-браузере Internet Explorer нет свойства innerHeight. Вместо него можно использовать свойство clientHeight объекта document;
- outerWidth — полная ширина окна в пикселах, включая ширину полос прокрутки. В Web-браузере Internet Explorer нет свойства outerWidth;

- `outerHeight` — полная высота окна в пикселах, включая высоту строки меню и панели инструментов. В Web-браузере Internet Explorer нет свойства `outerHeight`.

Свойства, предназначенные для назначения обработчиков событий:

- `onload` — вызывается после загрузки Web-страницы;
- `onunload` — вызывается непосредственно перед выгрузкой документа;
- `onscroll` — вызывается при прокручивании содержимого окна или фрейма;
- `onresize` — вызывается при изменении размеров окна;
- `onblur` — вызывается, когда окно теряет фокус;
- `onfocus` — вызывается, когда окно получает фокус;
- `onerror` — вызывается при возникновении ошибки в коде JavaScript. В качестве значения указывается ссылка на функцию следующего формата:

```
function <Название>(<Описание>, <URL>, <Номер строки>) {
   // return true; // Если ошибка обработана
   // return false; // Если ошибка не обработана
}
```

  Внутри функции необходимо вернуть значение `true`, если ошибка обработана, или `false` — в противном случае. Пример:

```
window.onerror = function(e, u, n) {
   window.alert(e + "\n" + u + "\n" + n + "\n");
   return true; // Якобы обработали
}
```

Объекты, являющиеся свойствами объекта `window`:

- `document` — объект `document` *(см. разд. 3.17.10)*;
- `event` — объект `event`. Этот объект уже был рассмотрен нами при изучении событий *(см. разд. 3.16.10)*;
- `history` — объект `history` *(см. разд. 3.17.9)*;
- `location` — объект `location` *(см. разд. 3.17.8)*;
- `navigator` — объект `navigator` *(см. разд. 3.17.6)*;
- `screen` — объект `screen` *(см. разд. 3.17.7)*.

Методы объекта `window`:

- `alert()` — отображает окно сообщения *(см. разд. 3.4.1)*;
- `confirm()` — выдает окно подтверждения *(см. разд. 3.4.2)*;
- `prompt()` — показывает окно с полем ввода *(см. разд. 3.4.3)*;
- `open()` — открывает новое окно Web-браузера *(см. разд. 3.17.3)*;
- `showModalDialog()` — отображает модальное диалоговое окно *(см. разд. 3.17.4)*;

- `close()` — закрывает окно;
- `blur()` — удаляет фокус с окна и генерирует событие `onblur`;
- `focus()` переносит фокус на текущее окно и генерирует событие `onfocus`;
- `navigate(<URL-адрес>)` — загружает в окно страницу, адрес которой указан в параметре. В Web-браузере Firefox нет метода `navigate()`;
- `stop()` — прерывает загрузку страницы. В Web-браузере Internet Explorer нет метода `stop()`;
- `resizeBy(<x>, <y>)` — изменяет размеры окна на заданное число пикселов;
- `resizeTo(<Ширина>, <Высота>)` — устанавливает размеры окна в пикселах;
- `moveBy(<x>, <y>)` — перемещает окно (по `x` — вправо, по `y` — вниз, а если указаны отрицательные значения, то наоборот);
- `moveTo(<x>, <y>)` — перемещает окно, чтобы верхний левый угол попал в заданную точку с координатами `x` и `y`;
- `scrollBy(<x>, <y>)` — прокручивает окно на заданные расстояния;
- `scrollTo(<x>, <y>)` — прокручивает содержимое окна в точку с координатами `x` и `y`.

Кроме того, имеются четыре метода для работы с таймерами, которые мы рассмотрим в *разд. 3.17.5*.

3.17.3. Работа с окнами. Создание нового окна без строки меню, адресной строки и панели инструментов

Метод `open()` объекта `window` позволяет открыть дополнительное окно и поместить в него Web-страницу:

```
[<Переменная> = ]window.open(<URL>, [<Имя окна>], [<Свойства окна>]);
```

Здесь используются следующие компоненты:

- `<URL>` — URL-адрес страницы, которая будет загружена в новое окно;
- `<Имя окна>` — имя нового окна;
- `<Свойства окна>` — определяет отображаемые элементы в новом окне;
- `<Переменная>` — ссылка на объект вновь созданного окна, которую можно использовать для работы с ним.

Свойства окна, передаваемые методу `open()`:

- `width` и `height` — задают ширину и высоту создаваемого окна в пикселах (минимум 100);
- `left` и `top` — указывают горизонтальную и вертикальную координаты левого верхнего угла создаваемого окна;

- `fullscreen` — {`yes` | `no` | `1` | `0`} — включает (`yes` или `1`) или отключает (`no` или `0`) полноэкранный режим для создаваемого окна;
- `resizable` — {`yes` | `no` | `1` | `0`} — включает или отключает возможность изменения размера создаваемого окна;
- `location` — {`yes` | `no` | `1` | `0`} — указывает наличие или отсутствие адресной строки;
- `menubar` — {`yes` | `no` | `1` | `0`} — включает или отключает отображение строки меню;
- `scrollbars` — {`yes` | `no` | `1` | `0`} — задает, отображать или нет полосы прокрутки;
- `status` — {`yes` | `no` | `1` | `0`} — указывает наличие или отсутствие строки состояния;
- `titlebar` — {`yes` | `no` | `1` | `0`} — включает или отключает отображение заголовка у создаваемого окна;
- `toolbar` — {`yes` | `no` | `1` | `0`} — включает или отключает отображение панели инструментов;
- `replace` — {`yes` | `no` | `1` | `0`} — задает режим сохранения адресов в истории: если указано `yes` или `1`, то адрес открываемого документа заменит в списке истории адрес документа, находящегося в текущем окне;
- `channelmode` — {`yes` | `no` | `1` | `0`} — задает режим отображения панели каналов: если указано `yes` или `1`, то создаваемое окно будет отображаться с панелью каналов.

Приведем пример создания нового окна и управления его местоположением и размерами. Создадим два файла: test.html (листинг 3.46) — страницу, открывающую новое окно, и doc1.html (листинг 3.47) — открываемый в новом окне документ.

Листинг 3.46. Содержимое файла test.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Открытие нового окна</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
var myWindow;
function f_open() { // Открываем окно
   var str = "menubar=0,location=0,resizable=0,scrollbars=0,";
   str += "status=0,titlebar=no,toolbar=0,left=0,";
   str += "top=0,width=500,height=500";
   myWindow = window.open("doc1.html", "window1", str);
}
```

```
function f_close() { // Закрываем окно
   var div1 = document.getElementById("div1");
   if (!myWindow.closed) {
      myWindow.close();
      div1.style.display = "none";
   }
   else {
      window.alert("Окно уже было закрыто");
      div1.style.display = "none";
   }
}
function f_resize() { // Задаем размер окна
   var div1 = document.getElementById("div1");
   var txt1 = document.getElementById("txt1");
   var txt2 = document.getElementById("txt2");
   var p = /^[0-9]{3}$/;
   if (p.test(txt1.value) && p.test(txt2.value)) {
      if (!myWindow.closed) {
         myWindow.resizeTo(txt1.value, txt2.value);
      }
      else {
         window.alert("Окно было закрыто раньше");
         div1.style.display = "none";
      }
   }
   else {
      window.alert("Необходимо ввести число из 3 цифр");
   }
}
function f_move() {
   // Изменяем местоположение относительно текущего положения окна
   var div1 = document.getElementById("div1");
   var txt3 = document.getElementById("txt3");
   var txt4 = document.getElementById("txt4");
   var p = /^\-?[0-9]+$/;
   if (p.test(txt3.value) && p.test(txt4.value)) {
      if (!myWindow.closed) {
         myWindow.moveBy(txt3.value, txt4.value);
      }
      else {
         window.alert("Окно уже было закрыто");
         div1.style.display = "none";
      }
   }
   else {
      window.alert("Необходимо ввести число");
   }
}
```

```
//-->
</script>
</head>
<body>
<div>
 <input type="button" value="Создать окно" onclick="f_open();"><br>
 <div style="display: none" id="div1">
 <input type="button" value="Закрыть созданное окно"
 onclick="f_close();"><br>
 X: <input type="text" id="txt1"><br>
 Y: <input type="text" id="txt2"><br>
 <input type="button" value="Задать размеры окна"
 onclick="f_resize();"><br>
 X: +- <input type="text" id="txt3" value="0"><br>
 Y: +- <input type="text" id="txt4" value="0"><br>
 <input type="button" value="Изменить местоположение окна"
 onclick="f_move();">
 </div>
</div>
</body>
</html>
```

Листинг 3.47. Содержимое файла doc1.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Новое окно</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_closed() {
   window.close();
}
function f_unload() {
   opener.document.getElementById("div1").style.display = "none";
}
function f_load() {
   opener.document.getElementById("div1").style.display = "block";
}
//-->
</script>
</head>
<body onunload="f_unload();" onload="f_load();">
<div><h1>Заголовок нового окна</h1>
```

```
<input type="button" value="Закрыть окно" onclick="f_closed();">
</div>
</body>
</html>
```

Откроем в Web-браузере файл test.html и нажмем кнопку **Создать окно**. В итоге отобразится новое окно и стапут доступными дополнительные возможности:

- кнопка, позволяющая закрыть открытое окно;
- кнопка **Задать размеры окна** и два поля, позволяющие задать новый размер открытого окна;
- кнопка **Изменнть местоноложенне окна** и два поля, позволяющие задать смещение окна относительно текущего местоположения (может принимать отрицательные значения).

3.17.4. Модальные диалоговые окна. Использование модальных окон вместо встроенных диалоговых окон

Модальное окно — это окно, которое будет активным до тех пор, пока пользователь его не закроет. Такие окна могут применяться вместо метода `prompt()` для ввода данных.

Для их отображения выполняется такой код:

```
[<Переменная> = ]window.showModalDialog(<URL>, [<Аргументы>],
[<Свойства окна>]);
```

Здесь используются следующие параметры:

- `<URL>` — определяет URL-адрес страницы, которая будет загружена в окно;
- `<Аргументы>` — позволяют передать в диалоговое окно произвольный набор параметров;
- `<Свойства окна>` — определяют внешний вид окна.

В параметре `<Свойства окна>` могут быть указаны следующие свойства:

- `dialogWidth` и `dialogHeight` — задают ширину и высоту окна в абсолютных (`px`, `mm`) или относительных (`em`, `ex`) величинах;
- `dialogTop` и `dialogLeft` — задают вертикальпую и горизонтальпую координаты левого верхнего угла создаваемого окна;
- `center` — {`yes | no | 1 | 0`} — устанавливает выравнивание окна по центру экрана;
- `edge` — {`sunken | raised`} — задает вид границы окна: вдавленный (`sunken`) или вынуклый (`raised`);
- `resizable` — {`yes | no | 1 | 0`} — включает или отключает возможность изменения размера создаваемого окна;

- `scroll` — {`yes | no | 1 | 0`} — устанавливает режим отображения полос прокрутки;
- `status` — {`yes | no | 1 | 0`} — включает или отключает отображение строки состояния;
- `dialogArguments` — переменная или массив переменных, передаваемых в модальное диалоговое окно;
- `returnValue` — возвращает значение из модального окна.

Приведем пример использования модальных диалоговых окон. Создадим два файла: основной документ test.html (листинг 3.48) и документ doc1.html (листинг 3.49), загружаемый в модальное окно.

Листинг 3.48. Содержимое файла test.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<!-- Корректно работает в Internet Explorer и Firefox/3.5.1
     В Opera 9.02 не работает -->
<html>
<head>
 <title>Модальные диалоговые окна</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_open() { // Открываем окно
   var obj = window.showModalDialog("doc1.html", ["Имя", "Фамилия"],
         "dialogWidth:300px; dialogHeight:150px; center=yes; status=no");
   if (obj != null) {
      var msg = "Имя: " + obj.pole1;
      msg += "<br>Фамилия: " + obj.pole2;
      document.getElementById("div1").innerHTML = msg;
   }
}
//-->
</script>
</head>
<body>
<p><input type="button" value="Открыть окно" onclick="f_open();"></p>
<div id="div1"></div>
</body>
</html>
```

Листинг 3.49. Содержимое файла doc1.html (модальное окно)

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

```
<head>
 <title>Введите имя и фамилию</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_click() { // Возвращаем значения
   var obj1 = {};
   obj1.pole1 = document.forms[0].pole1.value;
   obj1.pole2 = document.forms[0].pole2.value;
   window.returnValue = obj1;
   window.close();
}
function f_load() { // Задаем значения
   document.forms[0].pole1.value = window.dialogArguments[0];
   document.forms[0].pole2.value = window.dialogArguments[1];
}
//-->
</script>
</head>
<body onload="f_load();">
<form action="" id="frm">
<div style="text-align: center">
Имя:<br><input type="text" name="pole1"><br>
Фамилия:<br><input type="text" name="pole2"><br>
<input type="reset" value="Очистить">
<input type="button" value="OK" onclick="f_click();">
</div>
</form>
</body>
</html>
```

Идеальным решением для получения значений полей из модального диалогового окна является применение объектов. С помощью свойства `returnValue` мы передаем данные формы обратно в наш документ и можем с ними делать все, что захотим.

3.17.5. Таймеры. Создание часов на Web-странице

Таймеры позволяют однократно или многократно выполнять указанпую функцию через определенный интервал времени.

Для управления таймерами используются следующие методы объекта `window`:

- `setTimeout()` — создает таймер, однократно выполняющий указанпую функцию или выражение спустя заданный интервал времени:

  `<Идентификатор> = setTimeout(<Функция или выражение>, <Интервал>);`

- `clearTimeout(<Идентификатор>)` — останавливает таймер, установленный методом `setTimeout()`.

- `setInterval()` — создает таймер, многократно выполняющий указанную функцию или выражение через заданный интервал времени.

  `<Идентификатор> = setInterval(<Функция или выражение>, <Интервал>);`

- `clearInterval(<Идентификатор>)` — останавливает таймер, установленный методом `setInterval()`.

Здесь `<Интервал>` — это промежуток времени, по истечении которого выполняется `<Функция или выражение>`. Значение указывается в миллисекундах.

Приведем пример использования таймеров. В листинге 3.50 созданы часы на Web-странице, отображающие время вплоть до секунды. Добавим также возможность остановки и зануска часов.

Листинг 3.50. Часы на Web-странице

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Часы на Web-странице</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
var clock1;
function f_start() { // Запускаем таймер
   clock1 = setInterval("f_time();", 1000);
   document.getElementById("div_start").style.display = "none";
   document.getElementById("div_end").style.display = "block";
}
function f_time() { // Считываем текущее время
   var d = new Date();
   var msg = (d.getHours()<10) ? "0" : "";
   msg += d.getHours();
   msg += (d.getMinutes()<10) ? ":0" : ":";
   msg += d.getMinutes();
   msg += (d.getSeconds()<10) ? ":0" : ":";
   msg += d.getSeconds();
   document.getElementById("div1").innerHTML = msg;
}
function f_end() { // Останавливаем таймер
   clearInterval(clock1);
   document.getElementById("div_start").style.display = "block";
   document.getElementById("div_end").style.display = "none";
}
//-->
</script>
</head>
```

```
<body onload="f_start();">
<div id="div1"></div>
<div id="div_start">
  <input type="button" value="Запустить часы" onclick="f_start();">
</div>
<div id="div_end">
  <input type="button" value="Остановить часы" onclick="f_end();">
</div>
</body>
</html>
```

3.17.6. Объект *navigator*. Получение информации о Web-браузере пользователя. Перенаправление клиента на разные страницы в зависимости от Web-браузера

Объект `navigator` предоставляет информацию о самом Web-браузере.

Свойства объекта `navigator`:

- `appName` — имя Web-браузера;
- `appCodeName` — кодовое имя версии Web-браузера;
- `appVersion` — версия Web-браузера;
- `appMinorVersion` — вторая цифра в номере версии Web-браузера;
- `userAgent` — комбинация свойств `appCodeName` и `appVersion`;
- `cpuClass` — тип процессора клиентского компьютера;
- `platform` — название клиентской платформы;
- `systemLanguage` — код языка операционной системы клиента;
- `browserLanguage` — код языка Web-браузера;
- `userLanguage` — код языка Web-браузера;
- `onLine` — режим подключения: `true`, если клиент в настоящее время подключен к Интернету, и `false`, если отключен;
- `cookieEnabled` — режим работы `cookie`: возвращает `true`, если прием `cookie` разрешен.

Продемонстрируем все на примере (листинг 3.51).

Листинг 3.51. Информация о Web-браузере

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

```
<head>
 <title>Информация о Web-браузере</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<h1 style="text-align: center">Информация о Web-браузере</h1>
<div>
<script type="text/javascript">
<!--
document.write(navigator.appName + " - имя Web-браузера.<br>");
document.write(navigator.appCodeName);
document.write(" - кодовое имя версии Web-браузера.<br>");
document.write(navigator.appVersion + " - версия Web-браузера.<br>");
document.write(navigator.appMinorVersion);
document.write(" - вторая цифра в номере версии Web-браузера.<br>");
document.write(navigator.userAgent + " - комбинация свойств ");
document.write("appCodeName и appVersion.<br><br>");
document.write(naviga*tor.cpuClass);
document.write(" - тип процессора клиентского компьютера.<br>");
document.write(navigator.platform);
document.write(" - название клиентской платформы.<br>");
document.write(navigator.systemLanguage);
document.write(" - код языка операционной системы клиента.<br>");
document.write(navigator.browserLanguage);
document.write(" - код языка Web-браузера (browserLanguage).<br>");
document.write(navigator.userLanguage);
document.write(" - код языка Web-браузера (userLanguage).<br><br>");
if (navigator.onLine) {
   document.write("Клиент в настоящее время подключен к Интернету.");
}
else {
   document.write("Клиент в настоящее время отключен от Интернета.");
}
document.write("<br>");
if (navigator.cookieEnabled) document.write("Прием cookie разрешен.");
else document.write("Прием cookie запрещен.");
//-->
</script>
</div>
</body>
</html>
```

Вывести все поддерживаемые свойства и методы объекта `navigator` позволяет следующий код:

```
for (var p in navigator) {
   document.write(p + " ==&gt; " + navigator[p] + "<br>");
}
```

Мы уже не раз говорили, что разные Web-браузеры могут по-разному выполнять программный код. По этой причине часто приходится писать персональный код под каждый Web-браузер. В листинге 3.52 приведен пример скрипта, перенаправляющего клиента на разные страницы в зависимости от названия Web-браузера.

Листинг 3.52. Перенаправляем клиента на разные страницы в зависимости от названия Web-браузера

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Перенаправляем клиента на разные страницы</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<script type="text/javascript">
<!--
var name1 = navigator.appName;
if (name1.indexOf("Explorer") != -1) {
   var Version1 = navigator.appVersion;
   var p =/compatible\;\s+\w+\s+(([0-9]+)\.[0-9]+)\;/i;
   p.exec(Version1);
   if (RegExp.$2=="6") {
     window.location.href = "explorer6.html";
   }
   else {
      if (RegExp.$2=="7") {
         window.location.href = "explorer7.html";
      }
      else {
         if (parseInt(RegExp.$2)>=8) {
            window.location.href = "explorer8.html";
         }
         else {
            window.location.href = "explorer.html";
         }
      }
   }
}
if (navigator.userAgent.indexOf("Firefox") != -1) {
   window.location.href = "firefox.html";
}
if (name1.indexOf("Opera") != -1) {
   window.location.href = "opera.html";
}
```

```
//-->
</script>
<p>Информация для других Web-браузеров</p>
</body>
</html>
```

В случае с Microsoft Internet Explorer мы проверяем также версию Web-браузера с помощью регулярных выражений. Таким образом, можно написать Web-страницу не только под определенный Web-браузер, но и под определенную его версию.

> ***Примечание***
>
> Так как поисковые машины не умеют обрабатывать код JavaScript, на практике лучше не использовать перенаправление, а обрабатывать различия на одной Web-странице. Кроме того, вместо определения названия и версии лучше использовать метод проверки функциональных возможностей Web-браузера. Например, проверить наличие необходимого метода, указав его без круглых скобок в операторе `if`.

В *разд. 1.16* мы рассмотрели специальный тег, который можно использовать в Web-браузере Internet Explorer. Для языка JavaScript Internet Explorer также предоставляет условные комментарии. Они начинаются с комбинации символов `/*@cc_on` и заканчиваются комбинацией `@*/`:

```
/*@cc_on
   window.alert("Это инструкция для Internet Explorer");
@*/
```

Внутри такой конструкции могут быть указаны ключевые слова `@if`, `@else` и `@end`. Например, выполнить один блок выражений только в Internet Explorer, а другой блок в остальных Web-браузерах позволяет следующий код:

```
/*@cc_on
   @if (@_jscript)
     window.alert("Это сообщение выведет Internet Explorer");
   @else*/
      // Этот блок в IE выполнен не будет
      window.alert("А это сообщение в другом Web-браузере");
   /*@end
@*/
```

3.17.7. Объект *screen*. Получение информации о мониторе пользователя

Объект `screen` содержит информацию о характеристиках видеосистемы компьютера клиента.

Свойства объекта `screen`:

- `width` — полная ширина экрана в пикселах;
- `height` — полная высота экрана в пикселах;

- `availWidth` — ширина, доступная для окна Web-браузера;
- `availHeight` — высота, доступная для окна Web-браузера;
- `colorDepth` возвращает глубину цвета: 4 — для 16 цветов, 8 — для 256 цветов, 32 — для 16,7 млн цветов.

Для примера выведем информацию о мониторе пользователя (листинг 3.53).

Листинг 3.53. Информация о видеосистеме

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Информация о видеосистеме</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<h1 style="text-align: center">Информация о видеосистеме</h1>
<div>
<script type="text/javascript">
<!--
document.write(screen.width + " - полная ширина экрана в пикселах.<br>");
document.write(screen.height);
document.write(" - полная высота экрана в пикселах.<br>");
document.write(screen.availWidth);
document.write(" - ширина, доступная для окна Web-браузера.<br>");
document.write(screen.availHeight);
document.write(" - высота, доступная для окна Web-браузера.<br>");
document.write(screen.colorDepth + " - глубина цвета.");
//-->
</script>
</div>
</body>
</html>
```

3.17.8. Объект *location*. Разбор составляющих URL-адреса документа. Создание многостраничных HTML-документов

Объект `location` содержит информацию об URL-адресе текущей Web-страницы.

Свойства объекта `location`:

- `href` — полный URL-адрес документа;
- `protocol` — идентификатор протокола;
- `port` — номер порта;

- `host` — имя компьютера в сети Интернет, вместе с номером порта;
- `hostname` — имя компьютера в сети Интернет;
- `pathname` — нуть и имя файла;
- `search` — строка параметров, указанная после знака "?" (включая этот знак);
- `hash` — имя "якоря" (закладки) в документе, указанное после знака "#" (включая этот знак).

Методы объекта `location`:

- `assign()` — загружает документ, URL-адрес которого указан в качестве параметра;
- `reload()` — перезагружает документ;
- `replace()` — загружает документ, URL-адрес которого указан в качестве параметра. При этом информация об URL-адресе предыдущего документа удаляется из объекта `history`.

Загрузить новый документ можно не только с помощью методов `assign()` или `replace()`, но и присвоив новый URL-адрес свойству `href` объекта `location`:

```
window.location.href = "newURL.html";
```

Если нужно загрузить документ в какой-либо фрейм, то сначала необходимо указать имя фрейма:

```
frameName.location.href = "newURL.html";
frameName.location.assign("newURL.html");
```

Для примера разберем URL-адрес документа на составляющие (листинг 3.54).

Листинг 3.54. Информация об URL-адресе

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Информация об URL-адресе</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<h1 style="text-align: center">Информация об URL-адресе</h1>
<div>
<script type="text/javascript">
<!--
document.write(window.location.href);
document.write(" - полный URL-адрес документа.<br>");
document.write(window.location.protocol);
document.write(" - идентификатор протокола.<br>");
document.write(window.location.port + " - номер порта.<br>");
```

```
document.write(window.location.host);
document.write(" - имя компьютера в сети Интернет, ");
document.write("вместе с номером порта.<br>");
document.write(window.location.hostname);
document.write(" - имя компьютера в сети Интернет.<br>");
document.write(window.location.pathname + " - путь и имя файла.<br>");
document.write(window.location.search);
document.write(" - строка параметров.<br>");
document.write(window.location.hash + " - имя якоря (закладки)");
document.write(" в документе.<br>");
//-->
</script>
</div>
</body>
</html>
```

Через строку URL-адреса могут передаваться некоторые параметры для программы, расположенной на сервере. Например, для просмотра многостраничного каталога может передаваться номер страницы. Но параметры могут быть переданы и HTML-документу. В листинге 3.55 приведен пример создания многостраничного HTML-документа.

Листинг 3.55. Многостраничный HTML-документ

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Многостраничный HTML-документ</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<h1 style="text-align: center">Многостраничный HTML-документ</h1>
<div>
<a href="test.html?page=1&amp;param1=5&amp;param2=Hello">Страница 1</a><br>
<a href="test.html?page=2&amp;param1=5">Страница 2</a><br>
<a href="test.html?page=3">Страница 3</a><br><br>
<script type="text/javascript">
<!--
var arr1, arr2;
var obj = {};
if (window.location.search != "") {
   var Str = window.location.search.substr(1);
   if (Str.indexOf("&") != -1) {
      arr1 = Str.split("&");
      for (var i=0, c=arr1.length; i<c; i++) {
         arr2 = arr1[i].split("=");
```

```
            obj[arr2[0]] = arr2[1];
         }
      }
      else {
         arr2 = Str.split("=");
         obj[arr2[0]] = arr2[1];
      }
}
else {
   obj.page = "1";
}
if (obj.page=="1") {
   document.write("Содержание страницы 1.<br><br>");
}
if (obj.page=="2") {
   document.write("Содержание страницы 2.<br><br>");
}
if (obj.page=="3") {
   document.write("Содержание страницы 3.<br><br>");
}
if (obj.param1!=undefined) {
   document.write("Параметр param1 равен " + obj.param1 + "<br>");
}
if (obj.param2!=undefined) {
   document.write("Параметр param2 равен " + obj.param2 + "<br>");
}
//-->
</script>
</div>
</body>
</html>
```

У этого метода есть большой минус. HTML-документ будет всегда содержать все фрагменты. При этом посетителю будет показываться только тот фрагмент, который задан параметрами. Однако при каждом переходе по ссылкам документ будет заново загружаться с сервера. Если документ имеет большой размер, то он каждый раз будет долго загружаться только ради одного фрагмента. А зачем? Ведь все нужные фрагменты уже есть в документе! По этой причине лучше воспользоваться свойством `display` объекта `style`, а не передавать параметры через URL-адрес. В листинге 3.56 показан пример использования свойства `display` объекта `style`.

Листинг 3.56. Использование свойства `display` объекта `style`

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

```
<head>
 <title>Многостраничный HTML-документ</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_go(page) {
   switch (page) {
     case 1:
       document.getElementById("page1").style.display = "block";
       document.getElementById("page2").style.display = "none";
       document.getElementById("page3").style.display = "none";
       break;
     case 2:
       document.getElementById("page1").style.display = "none";
       document.getElementById("page2").style.display = "block";
       document.getElementById("page3").style.display = "none";
       break;
     case 3:
       document.getElementById("page1").style.display = "none";
       document.getElementById("page2").style.display = "none";
       document.getElementById("page3").style.display = "block";
   }
}
//-->
</script>
<style type="text/css">
   div div { border: #FFE9B3 1px solid; background-color: #FFFBEF;
             min-height: 100px; margin: 10px 5px 10px 5px }
</style>
</head>
<body>
<h1 style="text-align: center">Многостраничный HTML-документ</h1>
<div><a href="#" onclick="f_go(1); return false;">Страница 1</a>
<div id="page1">Содержание страницы 1.</div></div>
<div><a href="#" onclick="f_go(2); return false;">Страница 2</a>
<div id="page2" style="display: none">Содержание страницы 2.</div></div>
<div><a href="#" onclick="f_go(3); return false;">Страница 3</a>
<div id="page3" style="display: none">Содержание страницы 3.</div></div>
</body>
</html>
```

Этот пример и выглядит гораздо проще, и не требует никакой перезагрузки страницы.

3.17.9. Объект *history*. Получение информации о просмотренных страницах. Реализация перехода на предыдущую просмотренную страницу

Объект `history` предоставляет доступ к списку всех просмотренных Web-страниц за последнее время.

Свойство `length` объекта `history` возвращает размер списка истории.

Методы объекта `history`:

- `back()` — загружает в окно Web-браузера предыдущий документ из списка истории;
- `forward()` — загружает в окно Web-браузера следующий документ из списка истории;
- `go(<Позиция>)` — перемещается в списке истории на позицию, номер которой указан в качестве параметра.

Листинг 3.57 демонстрирует, как реализовать ссылку на предыдущую просмотренную Web-страницу.

Листинг 3.57. Переход на предыдущую Web-страницу

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Переход на предыдущую Web-страницу</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<h1 style="text-align: center">Переход на предыдущую Web-страницу</h1>
<p><a href="javascript:history.back();">Перейти</a></p>
</body>
</html>
```

3.17.10. Объект *document*. Получение полной информации о HTML-документе

Объект `document` предоставляет доступ ко всем элементам Web-страницы.

Свойства объекта `document`:

- `activeElement` — ссылка на активный элемент документа;
- `documentElement` — ссылка на корневой элемент (тег `<html>`);
- `body` — ссылка на все содержимое тега `<body>`;
- `title` — название документа, указанное в теге `<title>`;

- `URL` — URL-адрес документа;
- `referrer` — URL-адрес, с которого перешел посетнтель по ссылке;
- `parentWindow` — окно, которому принадлежит документ;
- `cookie` — объект cookie, который позволяет сохранять данные на компьютере клиента *(см. разд. 3.17.18)*;
- `readyState` — статус документа. Возвращает следующие строковые значення:
  - `complete` — документ полностью загружен;
  - `interactive` — документ загружен не полностью, но доступен для просмотра и управления;
  - `loading` — документ загружается;
  - `uninitialized` — документ не доступен;
- `location` — объект `location`;
- `selection` — объект `selection`;
- `fileCreatedDate` — дата создания файла документа в виде строки;
- `fileModifiedDate` — дата последнего изменения файла документа в виде строки;
- `fileUpdatedDate` — дата последнего изменения файла сервером в кэше компьютера пользователя;
- `lastModified` — дата и время последнего изменения файла документа в виде строки;
- `fileSize` — размер файла в виде строки;
- `bgColor` — цвет фона страницы;
- `fgColor` — цвет текста страницы;
- `linkColor` — цвет гиперссылок документа;
- `alinkColor` — цвет активных гиперссылок;
- `vlinkColor` — цвет посещенных гиперссылок.

Методы объекта `document`:

- `getElementById(<Идентификатор>)` — возвращает ссылку на элемент с указанным идентификатором;
- `getElementsByName(<Название элемента>)` — возвращает ссылку на коллекцию элементов, у которых параметр `name` равен значению аргумента;
- `getElementsByTagName(<Тег>)` — возвращает ссылку на коллекцию дочерних элементов, созданных с использованием тега, переданного в качестве параметра;
- `elementFromPoint(<x>, <y>)` — возвращает ссылку на элемент, находящийся по координатам `<x>` и `<y>`;
- `write(<Текст>)` — записывает текст, переданный как параметр, в текущее место документа;

- `writeln(<Текст>)` — записывает текст, переданный как параметр, в текущее место документа, а в конце строки добавляет символы возврата каретки и перевода строки. Добавляемые символы полностью игнорируются Web-браузерами, поэтому результат будет таким же, как и в случае с методом `write()`.

Коллекции объекта `document`:

- `all` — коллекция всех элементов;
- `anchors` — коллекция "якорей", заданных тегом `<a>`;
- `forms` — коллекция всех форм;
- `frames` — все фреймы;
- `images` — коллекция всех изображений;
- `links` — коллекция ссылок;
- `scripts` — коллекция скриптов;
- `styleSheets` — коллекция стилей.

Общие свойства и методы элементов Web-страницы

Все элементы Web-страницы также имеют свойства и методы. Помимо свойств, специфических для конкретных элементов, все они имеют следующие общие свойства:

- `all` — ссылка на коллекцию дочерних элементов;
- `id` — имя элемента, заданное параметром `id`;
- `className` — имя класса, заданное параметром `class` элемента Web-страницы;
- `sourceIndex` — порядковый номер элемента, который можно использовать для ссылки на элемент из коллекции `all`;
- `tagName` — имя тега элемента;
- `parentElement` — ссылка на элемент-родитель;
- `length` — число элементов в коллекции;
- `height` и `width` — высота и ширина элемента;
- `clientHeight` и `clientWidth` — высота и ширина элемента без учета рамок, границ, полос прокрутки и т. п.;
- `clientLeft` — смещение левого края элемента относительно левого края элемента-родителя без учета рамок, границ, полос прокрутки и т. п.;
- `clientTop` — смещение верхнего края элемента относительно верхнего края элемента-родителя без учета рамок, границ, полос прокрутки и т. п.;
- `offsetHeight` и `offsetWidth` — высота и ширина элемента относительно элемента-родителя;
- `offsetLeft` — смещение левого края элемента относительно левого края элемента-родителя;

- `offsetParent` — ссылка на элемент-родитель, относительно которого определяются свойства `offsetHeight`, `offsetWidth`, `offsetLeft` и `offsetTop`;
- `offsetTop` — смещение верхнего края элемента относительно верхнего края элемента-родителя;
- `innerText` — содержимое элемента, исключая теги HTML. Если присвоить свойству новое значение, то содержимое элемента изменится;
- `outerText` — содержимое элемента, исключая теги HTML. Если присвоить свойству новое значение, то содержимое элемента заменится новым, и сам элемент будет заменен;
- `innerHTML` — содержимое элемента, включая внутренние теги HTML. Если присвоить свойству новое значение, то содержимое элемента также изменится;
- `outerHTML` — содержимое элемента, включая теги HTML. Если присвоить свойству новое значение, то содержимое элемента заменится новым, а сам элемент будет заменен;
- `scrollHeight` и `scrollWidth` — полная высота и ширина содержимого элемента;
- `scrollLeft` и `scrollTop` — положение горизонтальной и вертикальной полос прокрутки.

Общие методы элементов Web-страницы:

- `getAdjacentText(<Местонахождение>)` — возвращает текстовую строку в зависимости от заданного местонахождения;
- `insertAdjacentHTML(<Местонахождение>, <Текст>)` — позволяет вставить текст в место, заданное местонахождением. Текст может содержать HTML-теги;
- `insertAdjacentText(<Местонахождение>, <Текст>)` — дает возможность вставить текст в место, заданное местонахождением. Текст не должен содержать HTML-тегов;
- `replaceAdjacentText(<Местонахождение>, <Текст>)` — позволяет заменить текст другим текстом в месте, заданном местонахождением.

  В этих методах `<Местонахождение>` может принимать следующие значения:

  - `BeforeBegin` — текст, находящийся перед открывающим тегом элемента;
  - `AfterBegin` — текст, находящийся после открывающего тега элемента, но перед всем содержимым текущего элемента;
  - `BeforeEnd` — текст, находящийся перед закрывающим тегом элемента, но после всего содержимого элемента;
  - `AfterEnd` — текст, находящийся после закрывающего тега элемента;
- `getAttribute(<Имя параметра>, true | false)` — возвращает значение параметра с именем `<Имя параметра>` тега текущего элемента. Если второй параметр равен `false`, то поиск параметра тега производится без учета регистра символов;

- `setAttribute(<Имя параметра>, <Значение>, true | false)` — присваивает `<Значение>` параметру с именем `<Имя параметра>` тега текущего элемента. Если третий параметр равен `false`, то при поиске параметра тега регистр символов не учитывается;
- `removeAttribute(<Имя параметра>, true | false)` — удаляет параметр тега текущего элемента. Если второй параметр равен `false`, то поиск параметра тега производится без учета регистра символов. Возвращает значение `true`, если удаление было выполнено успешно;
- `clearAttributes()` — удаляет все параметры тега элемента, кроме параметров `id` и `name`;
- `contains(<Имя элемента>)` — возвращает `true`, если элемент с этим именем содержится внутри текущего элемента;
- `getElementsByTagName(<Тег>)` — возвращает ссылку на коллекцию дочерних элементов, созданных с использованием тега, переданного в качестве параметра;
- `scrollIntoView(true | false)` — вызывает прокрутку страницы в окне так, чтобы текущий элемент оказался в поле зрения. Если параметр равен `true`, то текущий элемент окажется у верхнего края окна, а если `false` — то у нижнего края.

Получение информации о HTML-документе

В качестве примера использования свойств документа и его элементов рассмотрим листинг 3.58. Он демонстрирует получение полной информации о HTML-документе.

Листинг 3.58. Объект `document`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Объект document</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<div><a href="doc1.html" id="link1">Ссылка</a></div>
<script type="text/javascript">
<!--
document.write(document.URL + " - URL-адрес документа.<br>");
document.write(document.location.href + " - URL-адрес документа.<br>");
document.write(document.title + " - название документа.<br>");
document.title = "Новое название";
document.write(document.title + " - измененное название документа.<br>");
document.write(document.readyState + " - статус документа.<br><br>");
document.write(document.fileCreatedDate + " - дата ");
```

```
document.write("создания файла документа.<br>");
document.write(document.fileModifiedDate + " - дата ");
document.write("последнего изменения файла документа.<br>");
document.write(document.lastModified + " - дата и ");
document.write("время последнего изменения файла документа.<br>");
document.write(document.fileSize);
document.write(" - размер файла.<br><br>");
document.write("Цвет элементов до изменения параметров:<br><br>");
document.write(document.bgColor + " - цвет фона страницы;<br>");
document.write(document.fgColor + " - цвет текста страницы;<br>");
document.write(document.linkColor);
document.write(" - цвет гиперссылок документа;<br>");
document.write(document.alinkColor);
document.write(" - цвет активных гиперссылок;<br>");
document.write(document.vlinkColor);
document.write(" - цвет посещенных гиперссылок.<br><br>");
document.bgColor = "#009000";
document.fgColor = "#FFFFFF";
document.linkColor = "#FFFFFF";
document.alinkColor = "red";
document.vlinkColor = "black";
document.write("Цвет элементов после изменения параметров:<br><br>");
document.writeln(document.bgColor + " - цвет фона страницы;<br>");
document.writeln(document.fgColor + " - цвет текста страницы;<br>");
document.writeln(document.linkColor);
document.write(" - цвет гиперссылок документа;<br>");
document.writeln(document.alinkColor);
document.write(" - цвет активных гиперссылок;<br>");
document.writeln(document.vlinkColor);
document.write(" - цвет посещенных гиперссылок.<br><br>");
if (document.all) {
   document.write(document.all.link1.tagName);
   document.write(" - имя тега с id = link1.<br>");
}
document.write(document.getElementById("link1").innerHTML);
document.write(" - содержимое тега с id = link1.<br>");
document.getElementById("link1").innerHTML = "Новый текст ссылки";
document.write(document.getElementById("link1").innerHTML);
document.write(" - новое содержимое тега с id = link1.<br>");
//-->
</script>
<div style="margin-top: 300px">
<input type="button" value="Нажмите, чтобы увидеть ссылку"
onclick="document.getElementById('link1').scrollIntoView();"></div>
</body>
</html>
```

3.17.11. Обращение к элементам документа. Выравнивание заголовков по центру

Обратиться к элементу документа можно следующими способами:

- по номеру элемента в коллекции (элементы нумеруются в порядке появления в документе, начиная с нуля). Номер элемента может быть указан как в круглых, так и в квадратных скобках:

```
Str = document.all(1).tagName;
Str = document.all[1].tagName;
```

- по имени или идентификатору элемента:

```
Str = document.all["<name или id>"].tagName;
Str = document.all("<name или id>").tagName;
Str = document.all.<name или id>.tagName;
```

- с помощью метода `item()`. Обратите внимание, можно использовать только круглые скобки:

```
Str = document.all.item(1).tagName;
Str = document.all.item("<name или id>").tagName;
```

  Обратите внимание: в Internet Explorer версии 8.0 второй способ не работает;

- если идентификатор элемента является уникальным, то к элементу можно обратиться не как к элементу коллекции, а как к отдельному объекту:

```
Str = <name или id>.tagName;
```

- если идентификатор элемента не является уникальным, то нужно будет указать второй индекс:

```
Str = document.all("<name или id>", 2).tagName;
```

  Обратите внимание: в Internet Explorer версии 8.0 способ не работает.

Вместо коллекции `all` может быть указана другая коллекция. Например, к изображению можно обратиться следующими способами:

```
Str = document.images.<name или id>.src;
Str = document.images[<Индекс>].src;
Str = document.images.item(<Индекс>).src;
```

А к ссылке можно обратиться через коллекцию `links`:

```
Str = document.links[<Индекс>].href;
```

Коллекции `all` и `links` имеют дополнительный метод `tags`, позволяющий фильтровать элементы по их тегу:

```
document.all.tags("H1")
```

Количество элементов в коллекции можно узнать с помощью свойства `length`:

```
document.all.tags("H1").length
```

В некоторых Web-браузерах нет коллекции `all`. Вместо нее для поиска элемента по идентификатору следует пользоваться методом `getElementById(<Идентификатор>)`. Метод возвращает ссылку только на один элемент (даже если элементов с одинаковым идентификатором несколько), т. к. по определению идентификатор должен быть уникальным. Пример:

```
Str = document.getElementById("check1").value;
```

Чтобы получить элементы по названию тега, следует воспользоваться методом `getElementsByTagName(<Тег>)`. Получим количество тегов `<h1>`:

```
len = document.getElementsByTagName("h1").length;
```

А теперь количество тегов `<input>` внутри формы с идентификатором `frm`:

```
var frms = document.getElementById("frm");
var len = frms.getElementsByTagName("input").length;
```

В качестве примера в листинге 3.59 производится перебор всех элементов коллекции с помощью цикла. Выровняем все заголовки первого уровня по центру.

Листинг 3.59. Выравнивание всех заголовков по центру

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Выравнивание всех заголовков по центру</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<h1>Заголовок1</h1>
<h1>Заголовок2</h1>
<h1>Заголовок3</h1>
<h1>Заголовок4</h1>
<script type="text/javascript">
<!--
var tagsH1 = document.getElementsByTagName("h1");
var len = tagsH1.length;
for (var i=0; i<len; i++) {
   tagsH1[i].style.textAlign = "center";
}
//-->
</script>
</body>
</html>
```

3.17.12. Работа с элементами документа. Изменение URL-адреса и текста ссылки. Преобразование ссылки в обычный текст

Как видно из листинга 3.59, мы можем обращаться к атрибутам стиля любого тега напрямую. В этом листинге мы присвоили атрибуту `textAlign` (в CSS `text-align`) значение `"center"`:

```
tagsH1[i].style.textAlign = "center";
```

Аналогичным способом можно изменить адрес гиперссылки:

```
<a href="doc1.html" id="link1">Текст ссылки</a>
<script type="text/javascript">
<!--
document.getElementById("link1").href = "doc2.html";
//-->
</script>
```

Или изменить адрес изображения:

```
<img src="image1.gif" id="image1">
<script type="text/javascript">
<!--
document.getElementById("image1").src = "image2.gif";
//-->
</script>
```

Или присвоить элементу другой стилевой класс:

```
<a href="doc1.html" class="i1" id="link1">Текст ссылки</a>
<script type="text/javascript">
<!--
document.getElementById("link1").className = "i2";
//-->
</script>
```

Какие параметры имеет тот или иной тег и какие значения он может принимать, мы рассматривали при изучении языка HTML в *главе 1*.

С параметрами тегов можно работать также с помощью встроенных методов. Получить значение параметра тега можно с помощью метода `getAttribute()`, присвоить новое значение параметру можно с помощью метода `setAttribute()`, удалить конкретный параметр можно с помощью метода `removeAttribute()`, а удалить все параметры можно с помощью метода `clearAttributes()`.

Если с помощью параметров тега `<a>` можно управлять внешним видом ссылки, то с помощью общих свойств можно изменить сам текст ссылки:

```
<a href="doc1.html" id="link1">Текст ссылки</a>
<script type="text/javascript">
```

```
<!--
document.getElementById("link1").innerHTML = "Новый текст ссылки";
//-->
</script>
```

Более того, мы можем преобразовать гиперссылку в обычный абзац:

```
<a href="doc1.html" id="link1">Текст ссылки</a>
<script type="text/javascript">
<!-- // В Firefox/3.5.1 не работает
document.getElementById("link1").outerHTML = "<p>Новый абзац</p>";
//-->
</script>
```

Или добавить что-либо перед, после или внутри тега:

```
<a href="doc1.html" id="link1">Текст ссылки</a>
<script type="text/javascript">
<!-- // В Firefox/3.5.1 не работает
var l1 = document.getElementById("link1");
l1.insertAdjacentHTML("BeforeBegin",
"<span style='color: red'>Перед ссылкой</span> ");
l1.insertAdjacentHTML("AfterBegin", "Перед началом текста ссылки ");
l1.insertAdjacentHTML("BeforeEnd", " После текста ссылки");
l1.insertAdjacentHTML("AfterEnd", " После ссылки");
//-->
</script>
```

В последнем случае есть небольшой нюанс. Если мы захотим заключить текст ссылки в какой-нибудь парный тег, например тег `<span>`, то получим не тот результат, что планировался. Рассмотрим такой код:

```
<a href="doc1.html" id="link1">Текст ссылки</a>
<script type="text/javascript">
<!-- // В Firefox/3.5.1 не работает
var l1 = document.getElementById("link1");
l1.insertAdjacentHTML("AfterBegin",
"<span style='color: red'>Перед началом текста ссылки ");
l1.insertAdjacentHTML("BeforeEnd", " После текста ссылки</span>");
//-->
</script>
```

Вместо

```
<a href="doc1.html" id="link1"><span style='color: red'>Перед началом текста
ссылки Текст ссылки После текста ссылки</span></a>
```

мы получим нечто подобное:

```
<a href="doc1.html" id="link1"><span style='color: red'>Перед началом текста
ссылки </span>Текст ссылки После текста ссылки</a>
```

Иными словами, все открывающие теги в дополняющем фрагменте будут автоматически закрыты, а непарные закрывающие — просто удалены. Поэтому в данном случае следует воспользоваться свойством innerHTML:

```
<a href="doc1.html" id="link1">Текст ссылки</a>
<script type="text/javascript">
<!--
var l1 = document.getElementById("link1");
l1.innerHTML = "<span style='color: red'>" + l1.innerHTML + "</span>";
//-->
</script>
```

Необходимо заметить, что все свойства, которые мы использовали в этом разделе для изменения документа, не входят в стандарт DOM и поддерживаются только Web-браузерами Internet Explorer и Opera. Исключением является свойство innerHTML. Оно не входит в стандарт, но поддерживается практически всеми современными Web-браузерами.

Стандарт DOM предоставляет следующие свойства для получения информации об узле и передвижения по дереву HTML-документа:

- nodeType — тип узла. Может принимать следующие значения:
  - 1 — ELEMENT_NODE — тег;
  - 3 — TEXT_NODE — простой текст;
  - 8 — COMMENT_NODE — комментарий;
  - 9 — DOCUMENT_NODE — HTML-документ.
- nodeName — имя узла. Например, название тега для узла ELEMENT_NODE;
- nodeValue — значение узла. Например, текст для узлов TEXT_NODE и COMMENT_NODE;
- childNodes — массив всех дочерних узлов;
- firstChild — первый дочерний узел или значение null, если такого узла нет;
- lastChild — последний дочерний узел или значение null, если узла нет;
- parentNode — родительский узел или значение null, если такого узла нет;
- previousSibling — узел, непосредственно следующий перед данным узлом, или значение null, если такого узла нет;
- nextSibling — узел, непосредственно следующий после данного узла, или значение null, если такого узла нет;
- attributes — возвращает массив всех параметров тега. Каждый элемент массива содержит два свойства:
  - name — название параметра;
  - value — значение параметра.

  Выведем все параметры и их значения для первой ссылки в HTML-документе:

```
var attr = document.getElementsByTagName("a")[0].attributes;
var msg = "";
```

```
for (var i=0, len=attr.length; i<len; i++) {
   msg += "Параметр: " + attr[i].name + "\n";
   msg += "Значение: " + attr[i].value + "\n";
}
window.alert(msg);
```

Создать новый узел и добавить его в HTML-документ позволяют следующие методы:

- `createElement(<Название тега>)` — создает новый узел `ELEMENT_NODE`;
- `createTextNode(<Текст>)` — создает новый узел `TEXT_NODE`;
- `appendChild(<Новый узел>)` — добавляет новый узел в конец данного узла. Если узел уже находится в документе, то удаляет его и вставляет в новое место. В качестве примера создадим абзац со ссылкой и добавим его в конец документа:

```
var p = document.createElement("p"); // Создаем абзац
var text = document.createTextNode("Это текст абзаца ");
p.appendChild(text); // Добавляем текст в абзац
var link = document.createElement("a"); // Создаем ссылку
link.href = "test.html"; // Задаем URL-адрес ссылки
var link_text = document.createTextNode("Это текст ссылки");
link.appendChild(link_text); // Добавляем текст в ссылку
p.appendChild(link); // Добавляет ссылку в конец абзаца
// Добавляем новый узел в конец документа
document.body.appendChild(p);
```

- `insertBefore(<Новый узел>, <Узел>)` — добавляет новый узел перед указанным узлом. Если узел уже находится в документе, то удаляет его и вставляет в новое место. Если в качестве второго параметра указать значение `null`, то узел будет добавлен в конец данного узла:

```
var p = document.getElementsByTagName("p")[0];
// Перемещаем первый абзац в конец документа
document.body.insertBefore(p, null);
```

Вставим новый абзац в начало документа:

```
var p = document.createElement("p"); // Создаем абзац
var text = document.createTextNode("Это текст абзаца ");
p.appendChild(text); // Добавляем текст в абзац
// Добавляем новый узел в начало документа
document.body.insertBefore(p, document.body.firstChild);
```

- `cloneNode(true | false)` — создает копию узла. Если в качестве параметра указано значение `true`, то будут скопированы все потомки данного узла. Обратите внимание на то, что обработчики событий не копируются. Сделаем полную копию абзаца и добавим новый узел в конец документа:

```
var p = document.getElementsByTagName("p")[0].cloneNode(true);
// Добавляем в конец документа
document.body.appendChild(p);
```

- `hasChildNodes()` — возвращает значение `true`, если узел имеет дочерние узлы, или `false` — в противном случае. Пример:

```
var p = document.getElementsByTagName("p")[0];
if (p.hasChildNodes()) window.alert("Есть");
else window.alert("Дочерних узлов нет");
```

- `removeChild(<Удаляемый узел>)` — удаляет узел. В качестве примера удалим первый абзац:

```
var p = document.getElementsByTagName("p")[0];
p.parentNode.removeChild(p);
```

- `replaceChild(<Новый узел>, <Старый узел>)` — заменяет узел другим узлом. Заменим первый тег `<DIV>` на новый абзац:

```
var p = document.createElement("p"); // Создаем абзац
var text = document.createTextNode("Это текст абзаца");
p.appendChild(text); // Добавляем текст в абзац
var div = document.getElementsByTagName("div")[0];
div.parentNode.replaceChild(p, div);
```

Для получения информации об элементах таблицы предназначены специальные свойства:

- `caption` — ссылка на элемент `CAPTION` или значение `null`, если он не существует;
- `tHead` — ссылка на элемент `THEAD` или значение `null`, если он не существует;
- `tFoot` — ссылка на элемент `TFOOT` или значение `null`, если он не существует;
- `tBodies` — массив всех элементов `TBODY` в таблице;
- `rows` — массив всех строк в таблице или секции `TBODY` при использовании свойства `tBodies`. Каждый элемент массива содержит следующие свойства:
  - `cells` — массив всех ячеек в строке таблицы. Каждый элемент массива содержит свойство `cellIndex`, через которое доступен индекс ячейки в строке;
  - `rowIndex` — индекс строки в таблице;
  - `sectionRowIndex` — индекс строки внутри раздела (`THEAD`, `TBODY` или `TFOOT`).

Для создания и удаления элементов таблицы предназначены следующие методы:

- `createCaption()` — создает новый элемент `CAPTION` или возвращает ссылку на существующий элемент. Добавим заголовок к таблице:

```
var d;
d = document.getElementsByTagName("table")[0].createCaption();
var text = document.createTextNode("Это заголовок таблицы");
d.appendChild(text); // Добавляем текст в элемент CAPTION
```

- `deleteCaption()` — удаляет элемент `CAPTION`;
- `createTHead()` — создает новый элемент `THEAD` или возвращает ссылку на существующий элемент;

- `deleteTHead()` — удаляет элемент `THEAD`;
- `createTFoot()` — создает новый элемент `TFOOT` или возвращает ссылку на существующий элемент;
- `deleteTFoot()` — удаляет элемент `TFOOT`;
- `deleteRow(<Индекс>)` — удаляет строку из таблицы по указанному индексу;
- `insertRow(<Индекс>)` — вставляет новый пустой элемент `TR` в указанную позицию;
- `insertCell(<Индекс>)` — вставляет пустой элемент `TD`. Добавим новый ряд в самое начало первой таблицы в HTML-документе:

```
var r;
r = document.getElementsByTagName("table")[0].insertRow(0);
var cell1 = r.insertCell(0);
var cell2 = r.insertCell(1);
var t1 = document.createTextNode("Первая ячейка");
var t2 = document.createTextNode("Вторая ячейка");
cell1.appendChild(t1);
cell2.appendChild(t2);
```

- `deleteCell(<Индекс>)` — удаляет указанную ячейку.

В качестве примера создадим таблицу с заголовком:

```
var table = document.createElement("table");
table.border = 1; // Отображаем рамку
table.width = 300; // Ширина таблицы
var caption = table.createCaption();
var txt = document.createTextNode("Это заголовок таблицы");
caption.appendChild(txt); // Добавляем текст в элемент CAPTION
var row1 = table.insertRow(0); // Вставляем первую строку
var row2 = table.insertRow(1); // Вставляем первую строку
var cell1_1 = row1.insertCell(0); // Вставляем ячейку
var cell1_2 = row1.insertCell(1); // Вставляем ячейку
var cell2_1 = row2.insertCell(0); // Вставляем ячейку
var cell2_2 = row2.insertCell(1); // Вставляем ячейку
txt = document.createTextNode("1");
cell1_1.appendChild(txt); // Вставляем текст в ячейку
txt = document.createTextNode("2");
cell1_2.appendChild(txt); // Вставляем текст в ячейку
txt = document.createTextNode("3");
cell2_1.appendChild(txt); // Вставляем текст в ячейку
txt = document.createTextNode("4");
cell2_2.appendChild(txt); // Вставляем текст в ячейку
// Вставляем таблицу в конец документа
document.body.appendChild(table);
```

3.17.13. Объект *style*. Работа с таблицами стилей при помощи JavaScript

Объект `style` позволяет получить доступ к каскадным таблицам стилей.

Свойства соответствуют атрибутам в каскадных таблицах стилей с небольшими отличиями в написании:

- символы "-" удаляются;
- первые буквы всех слов в названии атрибута, кроме первого, делаются прописными.

Приведем примеры преобразования атрибутов стиля в свойства объекта `style`:

```
color -> color
font-family -> fontFamily
font-size -> fontSize
border-left-style -> borderLeftStyle
```

Атрибуты стилей и их донустимые значения мы рассматривали при изучении CSS в *главе 2*. По аналогии с приведенными примерами можно преобразовать названия атрибутов стиля в свойства объекта `style`.

Кроме свойств, соответствующих атрибутам стиля, объект `style` имеет дополнительные свойства:

- `cssText` — стили, заданные внутри тега с помощью параметра `style`;
- `pixelHeight` и `pixelWidth` — высота и ширина элемента в пикселах;
- `pixelLeft` и `pixelTop` — горизонтальная и вертикальная координаты левого верхнего угла элемента в пикселах;
- `posHeight` и `posWidth` — высота и ширина элемеита в единицах измерения, заданных в определении стиля;
- `posLeft` и `posTop` — горизонтальная и вертикальная координаты левого верхнего угла элемента в единицах измерения, заданных в определении стиля.

Значения дополнительных свойств задаются и возвращаются в виде числа, что очень удобно для различных вычислений. Рассмотрим это на примере (листинг 3.60).

Листинг 3.60. Объект `style`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Объект style</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
 <style type="text/css">
```

```
  div.bl { position: absolute; top: 150px; width: 150px;
           height: 150px; overflow: auto; background-color: green }
 </style>
<script type="text/javascript">
<!--
function f_print() {
   var d1 = document.getElementById("div1");
   var d2 = document.getElementById("div2");
   var d3 = document.getElementById("div3");
   var curSt = 0;
   if (d1.currentStyle) curSt = d1.currentStyle.top;
   else if (window.getComputedStyle) // Для Firefox
      curSt = window.getComputedStyle(d1, null).top;
   d2.innerHTML = "style.left = " + d1.style.left +
                  " currentStyle.top = " + curSt;
   d3.innerHTML = "style.pixelLeft = " + d1.style.pixelLeft +
                  "  style.posLeft = " + d1.style.posLeft;
}
function f_click1() {
   var d1 = document.getElementById("div1");
   d1.style.left = "50px";
   f_print();
}
function f_click2() {
   var d1 = document.getElementById("div1"), left1;
   if (d1.style.pixelLeft)
      d1.style.pixelLeft += 10;
   else { // Для Firefox
      left1 = parseInt(d1.style.left) + 10;
      d1.style.left = left1 + "px";
   }
   f_print();
}
function f_load() {
   var d1 = document.getElementById("div1");
   var d2 = document.getElementById("div2");
   var d3 = document.getElementById("div3");
   d2.innerHTML = "style.left = " + d1.style.left +
                  " style.top = " + d1.style.top;
   d3.innerHTML = "style.pixelLeft = " + d1.style.pixelLeft +
                  "  style.posLeft = " + d1.style.posLeft;
}
//-->
</script>
</head>
<body onload="f_load();">
 <div id="div2"></div>
```

```
 <div id="div3"></div>
 <div class="bl" id="div1" style="left: 5px;">Абсолютно позиционированный
  блок</div>
 <div>
 <input type="button" value="Сдвинуть блок на позицию 50 px"
  onclick="f_click1();">
 <input type="button" value="Сдвинуть блок вправо на 10 px"
  onclick="f_click2();"><br>
 <input type="button" value="Выделить первую строку красным цветом"
  onclick="document.getElementById('div2').style.color = 'red';">
</div>
</body>
</html>
```

После загрузки в окне Web-браузера отобразится следующее сообщение:

```
style.left = 5px style.top =
style.pixelLeft = 5  style.posLeft = 5
```

Итак, в случае применения свойства `left` получено значение в виде строки (5px), а в случае со свойствами `pixelLeft` и `posLeft` в виде числа (5). Но почему, получив значения свойства `left`, мы не получили значения свойства `top`? Все дело в том, что `style` возвращает значение, только если атрибут задан внутри тега с помощью параметра `style`. Первый атрибут был задан нами внутри тега с помощью параметра `style`, а второй атрибут задан в таблице стилей внутри тега `<style>` в заголовке документа. Поэтому значение первого мы получили, а второго — нет. Для того чтобы получить значение атрибута стиля, заданное вне тега, нужно использовать не `style`, а свойство `currentStyle`:

```
d1.currentStyle.top
```

Эту строку кода мы использовали в функции `f_print()` и поэтому после вызова первой функции получили следующий результат:

```
style.left = 50px currentStyle.top = 150px
style.pixelLeft = 50 style.posLeft = 50
```

В некоторых Web-браузерах свойство `currentStyle` не поддерживается. Вместо него применяется метод `getComputedStyle()` объекта `window`:

```
if (d1.currentStyle) curSt = d1.currentStyle.top;
else if (window.getComputedStyle) // Для Firefox
   curSt = window.getComputedStyle(d1, null).top;
```

3.17.14. Объект *selection*. Проверка наличия выделенного фрагмента

Объект `selection` позволяет получить доступ к тексту, выделенному в окне Web-браузера.

Свойство `type` объекта `selection` возвращает `Text`, если на странице что-либо выделено, и `None`, если выделения нет.

Методы объекта `selection`:

- `clear()` — стирает выделенный текст;
- `createRange()` — возвращает объект `TextRange` *(см. разд. 3.17.15)*;
- `empty()` — убирает выделение с текста.

В Web-браузерах Firefox и Opera для получения выделенного текста в документе (но не в текстовых полях) применяется метод `getSelection()` объекта `window`. Объект `Selection`, возвращаемый методом `getSelection()`, содержит следующие свойства:

- `anchorNode` — возвращает ссылку на текстовый узел, в котором началось выделение. В Web-браузере Opera всегда возвращает текстовый узел, в котором находится левая граница фрагмента, независимо от направления выделения. Получим элемент, в котором началось выделение, и сделаем его фон красным:

```
var rng = window.getSelection();
rng.anchorNode.parentNode.style.backgroundColor = "red";
```

- `anchorOffset` — возвращает смещение от начала текстового узла (возвращаемого свойством `anchorNode`) до начальной границы выделения:

```
var rng = window.getSelection();
window.alert(rng.anchorOffset);
```

- `focusNode` — возвращает ссылку на текстовый узел, в котором закончилось выделение. В Web-браузере Opera всегда возвращает текстовый узел, в котором находится правая граница фрагмента, независимо от направления выделения. Получим элемент, в котором закончилось выделение, и сделаем его фон красным:

```
var rng = window.getSelection();
rng.focusNode.parentNode.style.backgroundColor = "red";
```

- `focusOffset` — возвращает смещение от начала текстового узла (возвращаемого свойством `focusNode`) до конечной границы выделения:

```
var rng = window.getSelection();
window.alert(rng.anchorOffset);
```

- `rangeCount` — возвращает количество объектов `Range`, которые входят в выделенный фрагмент;
- `isCollapsed` — возвращает `true`, если объект свернут в точку, и `false` — в противном случае:

```
var rng = window.getSelection();
if (rng.isCollapsed) window.alert("Свернут");
else window.alert("Нет");
```

Методы объекта `Selection`:

- `toString()` — возвращает текстовое содержимое выделенного фрагмента;
- `collapse(<Узел>, <Смещение>)` — сворачивает выделение в указанную точку;
- `collapseToStart()` — сворачивает выделение в начало фрагмента:

```
var rng = window.getSelection();
rng.collapseToStart(); // Сворачиваем в начало
```

- `collapseToEnd()` — сворачивает выделение в конец фрагмента:

```
var rng = window.getSelection();
rng.collapseToEnd(); // Сворачиваем в конец
```

- `deleteFromDocument()` — удаляет выделенный фрагмент из документа;
- `extend(<Узел>, <Смещение>)` — перемещает конечную границу выделенного фрагмента в указанную позицию. В качестве примера расширим (или уменьшим в зависимости от направления выделения) фрагмеит до конца узла, содержащего конечную границу:

```
var rng = window.getSelection();
if (!rng.isCollapsed) {
   var len = rng.focusNode.length;
   if (rng.focusOffset!=len) {
      rng.extend(rng.focusNode, len);
   }
}
else window.alert("Нет выделенного фрагмента");
```

- `getRangeAt(<Индекс>)` — возвращает объект `Range` по указанному индексу. Выражение `getRangeAt(0)` позволяет получить объект `Range`, полностью соответствующий текущему выделению;
- `addRange(<Объект Range>)` — добавляет указанный объект `Range` к текущему выделению;
- `removeRange(<Объект Range>)` — удаляет указанный объект `Range` из выделенного фрагмента;
- `removeAllRanges()` — удаляет все объекты `Range` из выделенного фрагмента;
- `selectAllChildren(<Узел>)` — выделяет текстовое содержимое указанного узла и всех его потомков. Работает только в Firefox. Выделим содержимое первого абзаца в документе:

```
var rng = window.getSelection();
var elem = document.getElementsByTagName("p")[0];
rng.selectAllChildren(elem);
```

В качестве примера проверим наличие выделенного фрагмента и при его наличии выведем фрагмент с помощью метода `alert()`, а затем уберем выделение (листинг 3.61).

Листинг 3.61. Проверка наличия выделенного фрагмента

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Проверка наличия выделенного фрагмента</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_click() {
   if (window.getSelection) { // FF, Opera
      window.alert(window.getSelection().toString());
   }
   else { // Для IE
      if (document.selection.type=="Text") {
         var range1 = document.selection.createRange();
         window.alert("Выделенный фрагмент\n\n \"" + range1.text + "\"");
         document.selection.empty(); // Убираем выделение
      }
      else {
         window.alert("Нет выделенного фрагмента");
      }
   }
}
//-->
</script>
</head>
<body>
<p>Проверка наличия выделенного фрагмента</p>
<p><input type="button" value="Проверить" onclick="f_click();"></p>
</body>
</html>
```

В Web-браузерах Firefox и Opera фрагмент, выделенный в текстовом поле, нельзя получить с помощью метода `getSelection()`. Вместо этого метода следует использовать свойства `selectionStart` и `selectionEnd`. Пример:

```
// Ссылка на текстовое поле
var elem = document.getElementById("txt1");
if (elem.selectionStart != undefined &&
    elem.selectionEnd != undefined) {
   var s = elem.selectionStart; // Начальная позиция
   var e = elem.selectionEnd; // Конечная позиция
   window.alert(elem.value.substring(s, e));
}
```

3.17.15. Объект *TextRange*. Поиск фрагмента в текстовом поле или документе. Расширение или сжатие выделенного фрагмента текста

Объект `TextRange` предоставляет доступ к фрагменту текста Web-страницы. Для работы с фрагментом необходимо создать объект с помощью метода `createTextRange()`:

```
var Text1 = document.body.createTextRange();
```

Свойства объекта `TextRange`:

- `text` — возвращает текстовый фрагмент;
- `htmlText` — возвращает HTML-код содержимого объекта;
- `boundingHeight` и `boundingWidth` — служат для определения высоты и ширины прямоугольника, содержащего текстовую область;
- `boundingLeft` и `boundingTop` — позволяют определить горизонтальную и вертикальную координаты левого верхнего угла прямоугольника, содержащего текстовую область, относнтельно элемента, содержащего объект `TextRange`;
- `offsetLeft` и `offsetTop` — возвращают горизонтальную и вертикальную координаты левого верхнего угла прямоугольника, содержащего текстовую область, относительно окна.

Методы объекта `TextRange`:

- `select()` — выделяет содержимое объекта `TextRange`;
- `pasteHTML(<Текст>)` — заменяет текущее содержимое объекта `TextRange` на HTML-фрагмент, указанный в качестве параметра;
- `findText(<Текст>)` — проверяет наличие заданного текста внутри объекта `TextRange`. Возвращает `true`, если текст был найден;
- `scrollIntoView()` — прокручивает содержимое окна так, чтобы объект был виден в окне;
- `expand(<Элемент>)` — расширяет объект `TextRange` на один `<Элемент>`. Возвращает `true`, если объект был расширен, и `false` — в противном случае;
- `move(<Элемент>, <Количество>)` — сжимает объект в точку и перемещает его на заданное `<Количество> <Элементов>`. `<Количество>` может принимать положительные или отрицательные значения. Если оно не задано, то объект сдвигается на один `<Элемент>`;
- `moveStart(<Элемент>, <Количество>)` — перемещает начальную границу объекта на заданное `<Количество> <Элементов>`;
- `moveEnd(<Элемент>, <Количество>)` — перемещает конечную границу объекта на заданное `<Количество> <Элементов>`.

В качестве <Элемента> могут быть заданы следующие строковые значения:

- character — символ;
- word — слово;
- sentence — предложение;
- textedit — область объекта;

- collapse(true | false) — сворачивает объект TextRange, т. е. помещает открывающие и закрывающие маркеры объекта TextRange вместе в начало или конец текущего объекта. Если параметр равен true, то маркеры помещаются в начало, если false — то в конец. Используется для установки маркера ввода в нужную позицию;
- moveToPoint(<x>, <y>) — передвигает границы объекта и сжимает его вокруг выбранной точки. Координаты отсчитываются относительно окна;
- moveToElementText(<Элемент страницы>) — перемещает объект так, чтобы он охватил текст в заданном элементе;
- duplicate() — возвращает новый объект TextRange, являющийся копией текущего;
- parentElement() — возвращает ссылку на родительский элемент, содержащий текущий объект TextRange;
- inRange(<Объект>) — равен true, если указанный объект содержится внутри текущего;
- isEqual(<Объект>) — равен true, если указанный объект равен текущему;
- getBookmark() — создает закладку;
- moveToBookmark(<Закладка>) — переходит к закладке. Возвращает true, если переход прошел успешно;
- compareEndPoints(<Диапазон>, <Объект>) — сравнивает два объекта по указанному <Диапазону>. Возвращает следующие значения:
  - -1 — если граница текущего находится левее или выше границы указанного объекта;
  - 0 — если они равны;
  - 1 — если граница текущего находится правее или ниже границы указанного объекта.

  В качестве <Диапазона> могут быть указаны следующие строковые значения:
  - StartToEnd — сравнение начальной границы текущего объекта с конечной границей указанного объекта;
  - StartToStart — сравнение начальной границы текущего объекта с начальной границей указанного объекта;
  - EndToStart — сравнение конечной границы текущего объекта с начальной границей указанного объекта;

- `EndToEnd` — сравнение конечной границы текущего объекта с конечной границей указанного объекта;

❒ `setEndPoint(<Диапазон>, <Объект>)` — переносит начальную или конечную границу текущего объекта в начало или конец заданного объекта.

В качестве `<Диапазона>` могут быть указаны следующие значения:

- `StartToEnd` совмещает начальную границу текущего объекта с конечной границей указанного объекта;
- `StartToStart` совмещает начальную границу текущего объекта с начальной границей указанного объекта;
- `EndToStart` совмещает конечную границу текущего объекта с начальной границей указанного объекта;
- `EndToEnd` совмещает конечную границу текущего объекта с конечной границей указанного объекта.

В качестве примера в листинге 3.62 реализована возможность поиска фрагмента в текстовой области.

Листинг 3.62. Поиск фрагмента в текстовой области

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Поиск фрагмента</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_click() {
   if (!document.getElementById("txt2").createTextRange) {
      window.alert("Объект не поддерживается Web-браузером");
      return;
   }
   var Range1, Text1, Select1;
   Text1 = document.getElementById("txt1").value;
   Select1 = document.getElementById("sel1")
   if (Text1 != "") {
      Range1 = document.getElementById("txt2").createTextRange();
      if (Range1.findText(Text1)) {
         Range1.scrollIntoView();
         switch (Select1.value) {
         case "1":
            Range1.expand("character"); break;
         case "2":
            Range1.expand("word"); break;
```

```
         case "3":
            Range1.collapse(true); break;
         case "4":
            Range1.collapse(false); break;
         case "5":
            Range1.expand("word");
            Range1.collapse(true);
            break;
         case "6":
            Range1.expand("word");
            Range1.collapse(false);
            break;
         }
         Range1.select(); // Выделяем найденный фрагмент
         var msg = "bounding X: " + Range1.boundingLeft;
         msg += " bounding Y: " + Range1.boundingTop + "<br>";
         msg += "offset X: " + Range1.offsetLeft;
         msg += " offset Y: " + Range1.offsetTop + "<br>";
         msg += "Тег: " + Range1.parentElement().tagName;
         msg += " id: " + Range1.parentElement().id + "<br>";
         document.getElementById("div1").innerHTML = msg;
      }
      else window.alert("Ничего не найдено");
   }
   else window.alert("Поле не заполнено");
}
//-->
</script>
</head>
<body>
<div>
<input type="text" id="txt1"><br>
<select id="sel1">
<option value="0">Выделить текст</option>
<option value="1">Выделить текст + 1 символ</option>
<option value="2">Выделить все слово</option>
<option value="3">Поместить курсор в начало найденного фрагмента</option>
<option value="4">Поместить курсор в конец найденного фрагмента</option>
<option value="5">Поместить курсор в начало слова с фрагментом</option>
<option value="6">Поместить курсор в конец слова с фрагментом</option>
</select><br>
<input type="button" value="Найти" onclick="f_click();"><br>
<script type="text/javascript">
<!--
document.write("<textarea id='txt2' cols='50' rows='10'>");
for (var i=1; i<51; i++) {
   document.write("Содержимое строки" + i + "\n");
}
```

```
document.write("<" + "/textarea>");
//-->
</script>
</div>
<div id="div1"></div>
</body>
</html>
```

А следующий пример демонстрирует применение методов `moveStart()` и `moveEnd()`. С помощью кнопок можно перемещать начальную или конечную границу объекта `TextRange`, таким образом расширяя или сжимая выделенный фрагмент (листинг 3.63).

Листинг 3.63. Расширение выделенного фрагмента

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Расширение выделенного фрагмента</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_click(Val1) {
   if (!document.selection) {
      window.alert("Объект не поддерживается Web-браузером");
      return;
   }
   if (document.selection.type=="Text") {
      var Range1 = document.selection.createRange();
      switch (Val1) {
        case 1:
           Range1.moveStart("character", -1);
           Range1.select();
           break;
        case 2:
           Range1.moveStart("character");
           Range1.select();
           break;
        case 3:
           Range1.moveEnd("character", -1);
           Range1.select();
           break;
        case 4:
           Range1.moveEnd("character");
           Range1.select();
```

```
            break;
         }
      }
      else {
         window.alert("Необходимо выделить фрагмент");
      }
}
//-->
</script>
</head>
<body>
<p>Расширение выделенного фрагмента</p>
<div>
Начальная граница
<input type="button" value=" &lt; " onclick="f_click(1);">
<input type="button" value=" &gt; " onclick="f_click(2);"><br>
Конечная граница
<input type="button" value=" &lt; " onclick="f_click(3);">
<input type="button" value=" &gt; " onclick="f_click(4);">
</div>
</body>
</html>
```

В Web-браузерах Firefox и Opera поддерживается абсолютно другой объект, называемый `Range`. Для работы с таким объектом необходимо создать область с помощью метода `createRange()` объекта `document`:

```
var rng = document.createRange();
```

Свойства объекта `Range`:

- `startContainer` — возвращает ссылку на узел, в котором содержится начальная точка области;
- `startOffset` — возвращает смещение от начала узла (возвращаемого свойством `startContainer`) до начальной точки области;
- `endContainer` — возвращает ссылку на узел, в котором содержится конечная точка области;
- `endOffset` — возвращает смещение от начала узла (возвращаемого свойством `endContainer`) до конечной точки области;
- `collapsed` — возвращает `true`, если объект свернут в точку, и `false` — в противном случае:

```
var rng = document.createRange();
if (rng.collapsed) window.alert("Свернут");
else window.alert("Нет");
```

- `commonAncestorContainer` — возвращает ссылку на узел, в котором содержатся как начальная, так и конечная точки области.

Методы объекта `Range`:

- `toString()` — возвращает текстовое содержимое области;
- `cloneRange()` — создает копию объекта `Range`;
- `cloneContents()` — создает копию внутреннего содержимого области. В качестве значения возвращает объект `DocumentFragment`;
- `detach()` — удаляет объект `Range`;
- `deleteContents()` — удаляет все внутреннее содержимое области из документа;
- `extractContents()` — удаляет все внутреннее содержимое области из документа и возвращает объект `DocumentFragment`, в котором будет находиться удаленное содержимое области;
- `collapse(<true | false>)` — сворачивает область в указанную точку. Если в качестве параметра указано значение `true`, то область сворачивается в начальную точку, а если `false` — то в конечную точку;
- `selectNode(<Узел>)` — ограничивает область указанным в качестве параметра узлом;
- `selectNodeContents(<Узел>)` — ограничивает область внутренним содержимым указанного узла;
- `insertNode(<Узел>)` — вставляет новый узел в начало области;
- `setStart(<Узел>, <Смещение>)` — устанавливает положение начальной точки области;
- `setStartBefore(<Узел>)` — устанавливает начальную точку области перед указанным узлом;
- `setStartAfter(<Узел>)` — устанавливает начальную точку области после указанного узла;
- `setEnd(<Узел>, <Смещение)` — устанавливает положение конечной точки области;
- `setEndBefore(<Узел>)` — устанавливает конечную точку области перед указанным узлом;
- `setEndAfter(<Узел>)` — устанавливает конечную точку области после указанного узла;
- `surroundContents(<Узел>)` — вкладывает содержимое области в указанный узел;
- `compareBoundaryPoints(<Точки сравнения>, <Область, с которой сравниваем>)` — сравнивает позиции двух областей. В качестве первого параметра могут быть указаны следующие значения:
  - `0` — `START_TO_START` — сравнение начальных точек;
  - `1` — `START_TO_END` — сравнение начальной точки области, указанной в качестве второго параметра, с конечной точкой данной области;
  - `2` — `END_TO_END` — сравнение конечных точек;

- 3 — END_TO_START — сравнение конечной точки области, указанной в качестве второго параметра, с начальной точкой данной области.

В качестве примера использования объекта Range найдем внутри абзаца текст "фрагмент" и вложим его в тег :

```
<p id="txt">Текст для выделения фрагмента</p>
<input type="button" id="btn1" onclick="f_click()" value="Выделить">
<script type="text/javascript">
function f_click() {
   if (document.createRange) {
      var p = document.getElementById("txt").firstChild;
      var text = p.nodeValue; // Получаем текст абзаца
      var ind = text.indexOf("фрагмент");
      if (ind != -1) { // Если текст найден
         // Создаем объект Range
         var rng = document.createRange();
         rng.setStart(p, ind); // Начальная точка
         // Конечная точка
         rng.setEnd(p, ind + 8);
         // Элемент, в который будем вкладывать текст
         var s = document.createElement("strong");
         // Вкладываем область в тег strong
         rng.surroundContents(s);
      }
   }
   else {
      window.alert("Web-браузер не поддерживает метод createRange");
   }
}
</script>
```

Теперь изменим цвет фона текстового фрагмента, выделенного пользователем:

```
<p id="txt">Текст для выделения фрагмента</p>
<input type="button" id="btn1" onclick="f_click()" value="Выделить">
<script type="text/javascript">
function f_click() {
   if (document.createRange && window.getSelection) {
      var sel = window.getSelection();
      if (!sel.isCollapsed) {
         var rng = sel.getRangeAt(0);
         sel.collapseToStart(); // Убираем выделение
         // Элемент, в который будем вкладывать выделенный текст
         var s = document.createElement("span");
         s.style.backgroundColor = "#FFE9B3";
         // Вкладываем область в тег span
         rng.surroundContents(s);
      }
```

```
      else window.alert("Нет выделенного фрагмента");
   }
   else {
      window.alert("Web-браузер не поддерживает методы");
   }
}
</script>
```

3.17.16. Работа с буфером обмена. Выделение фрагмента от позиции щелчка до конца документа и копирование его в буфер обмена

С помощью объекта `clipboardData` можно получить доступ к буферу обмена Windows. Получить доступ к объекту можно с помощью свойства `clipboardData` объекта `window`:

```
window.clipboardData
```

Методы объекта `clipboardData`:

- `clearData(<Формат данных>)` — удаляет данные из буфера обмена в указанном формате. Если формат не задан, то будут удалены все данные.

  Могут быть указаны следующие форматы данных:

  - `Text` — текстовый;
  - `URL` — интернет-адрес;
  - `File` — файл;
  - `HTML` — HTML-код;
  - `Image` — изображение.

- `getData(<Формат данных>)` — возвращает данные из буфера обмена в заданном формате. Могут быть указаны два формата:

  - `Text` — текстовый;
  - `URL` — интернет-адрес.

- `setData(<Формат данных>, <Данные>)` — помещает данные в буфер обмена в заданном формате. Возвращает `true`, если данные помещены в буфер обмена. Могут быть указаны два формата:

  - `Text` — текстовый;
  - `URL` — интернет-адрес.

В листинге 3.64 приведен пример выделения фрагмента от позиции курсора до конца документа, причем выделенный фрагмент будет скопирован в буфер обмена.

Листинг 3.64. Выделение фрагмента от позиции курсора до конца документа

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Выделение фрагмента от позиции курсора до конца документа</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_click() {
   var Range1, Range2;
   Range1 = document.body.createTextRange();
   Range1.moveToPoint(window.event.clientX, window.event.clientY);
   Range2 = document.body.createTextRange();
   Range2.setEndPoint("StartToStart", Range1);
   Range2.select();
   window.clipboardData.setData("Text", Range2.text);
}
//-->
</script>
</head>
<body onclick="f_click();">
<div>
<script type="text/javascript">
<!--
for (var i=1; i<21; i++) {
   document.write("Содержимое строки" + i + "<br>");
}
//-->
</script>
</div>
</body>
</html>
```

3.17.17. Реализация ссылок "Добавить сайт в Избранное" и "Сделать стартовой страницей"

Какой владелец сайта не мечтает, чтобы пользователь еще раз посетил сайт? Размещение ссылки с возможностью быстрого добавления сайта в Избранное позволит приблизить эту мечту. Метод `addFavorite()` объекта `external` позволяет вывести диалоговое окно для добавления адреса сайта в список Избранное Web-браузера. Вызов метода имеет следующий формат:

```
external.addFavorite(<URL-адрес>[, <Описание>]);
```

В листинге 3.65 приведен пример реализации ссылки для добавления в Избранное, а также ссылки, позволяющей сделать страницу стартовой, т. е. первой страницей, которую увидит пользователь, при запуске Web-браузера. Обратите внимание на ключевое слово `this`, которое возвращает ссылку на текущий элемент.

Листинг 3.65. Реализация ссылок "Добавить сайт в Избранное" и "Сделать стартовой страницей"

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Добавление сайта в Избранное</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!-- // Работает только в IE
function f_add() {
   external.addFavorite("http://www.mail.ru", "Национальная почта");
   return false;
}
function f_HomePage(obj) {
   obj.style.behavior="url(#default#homepage)";
   obj.setHomePage("http://www.mail.ru");
   return false;
}
//-->
</script>
</head>
<body>
<p><a href="http://www.mail.ru" onclick="return f_add(); ">
Добавить сайт в Избранное</a><br>
<a href="http://www.mail.ru" onclick="return f_HomePage(this);">
Сделать стартовой страницей</a></p>
</body>
</html>
```

3.17.18. Сохранение данных на компьютере клиента. Определение возможности использования cookies. Сохранение русского текста в cookies

Web-браузеры позволяют сохранять небольшой объем информации в специальном текстовом файле на компьютере пользователя. Такая информация называется cookies. Возможность использования cookies можно отключить в настройках Web-браузера. Для проверки возможности использования cookies следует использовать свойство `cookieEnabled` объекта `navigator`.

```
if (navigator.cookieEnabled) {
   window.alert("Использование cookies разрешено");
}
```

Запись cookies производится нутем присвоения значения свойству `cookie` объекта `document` в следующем формате:

```
document.cookie = "<Имя>=<Значение>; [expires=<Дата>;]
[domain=<Имя домена>;] [path=<Путь>;] [secure;]";
```

Здесь используются следующие параметры:

- `<Имя>=<Значение>` — задает имя сохраняемой переменной и ее значение. Это единственный обязательный параметр. Если не задан параметр `expires`, то по истечении текущего сеанса работы Web-браузера cookies будут автоматически удалены;
- `expires` — указывает дату удаления cookies в следующем формате:

  ```
  Thu, 01 Jan 1970 00:00:01 GMT
  ```

  Получить дату в этом формате можно с помощью методов `setTime()` и `toGMTString()` класса `Date`. Методу `setTime()` нужно передать текущее время в миллисекундах плюс время хранения cookies в миллисекундах. Текущее время можно получить с помощью метода `getTime()`. Рассчитать время хранения cookies можно исходя из следующих соотношений:

  - 1 секунда = 1000 миллисекунд;
  - 1 минута = 60 секунд = 60 000 миллисекунд;
  - 1 час = 60 минут = 3600 секунд = 3 600 000 миллисекунд;
  - 1 день = 24 часа = (24×3 600 000) миллисекунд = 86 400 000 миллисекунд.

  Например:

  ```
  var d = new Date();
  d.setTime(d.getTime()+3600000); // Задан 1 час
  var End_Date = d.toGMTString(); // Дата удаления cookies
  ```

- `domain=<Имя домена>` — задает доменную часть URL-адреса, для которой действует данный cookies;
- `path=<Путь>` — задает часть URL-адреса, определяющую путь к документам, для которых действует данный cookies.

Считывание cookies производится с помощью свойства `cookie` объекта `document`:

```
var cookies = document.cookie;
```

Переменная `cookies` будет содержать строку, в которой перечислены все установленные пары `имя=значение` через точку с запятой:

```
"имя1=значение1; имя2=значение2"
```

Для удаления cookies следует установить cookies с прошедшей датой.

В качестве примера сохраним имя и фамилию пользователя, и при следующем посещении будем приветствовать его, используя сохраненные данные (листинг 3.66). Добавим также возможность удаления cookies. Для совместимости закодируем введенные данные с помощью метода `escape()`, а при выводе раскодируем их с помощью метода `unescape()`. Это позволяет безопасно сохранять значения, введенные кириллицей.

Листинг 3.66. Установка и удаление cookies

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Установка и удаление cookies</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_cookies() {
   if (navigator.cookieEnabled) {
      var text1 = document.getElementById("txt1");
      var text2 = document.getElementById("txt2");
      if (text1.value != "" && text2.value != "") {
         var d = new Date();
         d.setTime(d.getTime()+3600000); // Задан 1 час
         var End_Date = d.toGMTString(); // Дата удаления cookies
         var Str = "name1=" + escape(text1.value);
         Str += "; expires=" + End_Date + ";";
         document.cookie = Str;
         Str = "name2=" + escape(text2.value);
         Str += "; expires=" + End_Date + ";";
         document.cookie = Str;
         text1.value = "";
         text2.value = "";
         f_load();
      }
      else {
         window.alert("Не заполнено обязательное поле");
      }
   }
}
function f_cookies_del() {
   if (navigator.cookieEnabled) {
      if (document.cookie != "") {
         var d = new Date();
         d.setTime(1000); // Дата в прошлом
         var End_Date = d.toGMTString();
```

```
            document.cookie = "name1=; expires=" + End_Date + ";";
            document.cookie = "name2=; expires=" + End_Date + ";";
            f_load();
         }
      }
   }
function f_load() {
   if (navigator.cookieEnabled) {
      var div1 = document.getElementById("div1");
      if (document.cookie != "") {
         var arr1, arr2;
         var obj = {};
         var Str = document.cookie;
         if (Str.indexOf("; ") != -1) {
            arr1 = Str.split("; ");
            for (var i=0, c=arr1.length; i<c; i++) {
               arr2 = arr1[i].split("=");
               obj[arr2[0]] = arr2[1];
            }
         }
         else {
            arr2 = Str.split("=");
            obj[arr2[0]] = arr2[1];
         }
         Str = "Привет, " + unescape(obj.name2).replace("<", "&lt;");
         Str += " " + unescape(obj.name1).replace("<", "&lt;");
         div1.innerHTML = Str;
      }
      else div1.innerHTML = "";
   }
}
//-->
</script>
</head>
<body onload="f_load();">
<div id="div1"></div>
<div>
Введите ваше имя:<br>
<input type="text" id="txt1"><br>
Введите вашу фамилию:<br>
<input type="text" id="txt2"><br>
<input type="button" value="Сохранить" onclick="f_cookies();"><br>
<input type="button" value="Удалить cookies" onclick="f_cookies_del();">
</div>
</body>
</html>
```

3.18. Работа с элементами формы

При изучении HTML мы рассмотрели создание элементов форм. В этом разделе мы научимся с помощью JavaScript обрабатывать данные, введенные пользователем в элементы формы. Обработка на стороне клиента позволит снизить нагрузку на Web-сервер за счет отмены отправки данных формы при неправильно введенных значениях.

3.18.1. Элементы управления

Для начала еще раз перечислим все элементы форм:

- `<input type="text">` — текстовое поле ввода;
- `<input type="password">` — текстовое поле для ввода пароля;
- `<input type="file">` — позволяет отправить файл на Web-сервер;
- `<input type="checkbox">` — поле для установки флажка;
- `<input type="radio">` — элемент-переключатель;
- `<input type="reset">` — киопка, при нажатии которой вся форма очищается;
- `<input type="submit">` — кнопка, при нажатии которой происходит отправка данных на Web-сервер;
- `<input type="button">` — обычная командная кнопка;
- `<input type="hidden">` — скрытый элемент формы;
- `<textarea>Текст</textarea>` — поле для ввода многострочного текста;
- `<select><option>Элемент</option></select>` — список с возможными значениями.

Все элементы должны быть расположены внутри тегов `<form>` и `</form>`. Именно форма определяет, что делать с данными дальше. Параметр `action` задает URL-адрес программы обработки формы, параметр `method` определяет, как будут пересылаться данные от формы до Web-сервера (методом `GET` или `POST`), а параметр `enctype` задает MIME-тип передаваемых данных. С помощью параметра `name` задается уникальное имя формы, благодаря которому можно управлять элементами формы из скриптов.

Параметр `name` необходимо указывать во всех элементах формы, за исключением кнопок. Именно имя элемента, заданное в параметре `name`, пересылается на Web-сервер вместе со значением элемента формы. Имя элемента в пределах формы должно быть уникальным, за исключением переключателей, объединенных в группу.

Для доступа к элементам формы из скриптов необходимо указать параметр `id`. Обычно для элементов форм значения параметров `name` и `id` содержат одно и то же имя:

```
<input type="text" name="text1" id="text1">
```

Если данные не нужно отправлять на Web-сервер, то можно вообще не использовать тег <form>. В этом случае вся обработка осуществляется с помощью скриптов.

3.18.2. Коллекция *Forms*. Доступ к элементу формы из скрипта

Все формы документа доступны через коллекцию forms. Например, чтобы получить значение текстового поля с именем text1 (входящего в состав формы form1), можно воспользоваться следующей строкой кода:

```
document.forms["form1"].text1.value
```

Обратиться к форме можно и как к любому элементу документа:

```
document.form1.text1.value
```

К отдельной форме можно также обратиться по индексу:

```
document.forms[0].text1.value
```

Если элемент управления находится внутри тега <form>, то ссылку на саму форму пужно обязательно указывать, иначе Web-браузер будет искать элемент в теле документа, игнорируя все формы, и в итоге вернет значение null.

Получить доступ к элементу, вне зависимости от того находится он внутри формы или нет, позволяет метод getElementById() объекта document:

```
document.getElementById("text1").value
```

Все элементы формы доступны через коллекцию elements:

```
document.forms["form1"].elements["text1"].value
document.forms["form1"].elements[0].value
document.forms[0].elements[0].value
document.form1.elements[0].value
```

3.18.3. Свойства объекта формы

Объект формы поддерживает следующие свойства:

- length — количество элементов формы;
- action — URL-адрес программы обработки формы;
- elements — ссылка на коллекцию elements;
- encoding — MIME-тип передаваемых данных;
- method — режим пересылки данных формы на Web-сервер;
- enctype — метод кодирования данных формы;
- name — имя формы;
- target — имя фрейма, в который будет загружен документ, являющийся результатом обработки данных формы Web-сервером.

3.18.4. Методы объекта формы

Объект формы поддерживает следующие методы:

- `submit()` — выполняет отправку данных формы серверной программе. Аналогично нажатию кнопки **Submit**;
- `reset()` — очищает форму, т. е. все элементы формы получают значения по умолчанию. Аналогично нажатию кнопки **Reset**.

3.18.5. События объекта формы

Объект формы поддерживает следующие события:

- `onsubmit` — наступает при отправке данных формы;
- `onreset` — возникает при очистке формы.

Элементы управления имеют свои свойства, методы и события. Рассмотрим каждый тип элементов формы по отдельности.

3.18.6. Текстовое поле и поле ввода пароля. Проверка правильности ввода E-mail и пароля. Получение данных из элемента формы

Текстовое поле и поле для ввода пароля имеют одинаковые свойства:

- `value` — значение элемента формы;
- `defaultValue` — начальное значение, заданное параметром `value`;
- `disabled` — запрет элемента формы: если задано значение `true`, то поле является неактивным (отображается серым цветом);
- `form` — ссылка на форму, в которой находится элемент;
- `maxLength` — максимальное количество символов, которое может быть введено в поле;
- `name` — имя элемента;
- `type` — тип элемента формы;
- `readOnly` — запрет редактирования: если задано значение `true`, текст в поле нельзя редактировать, если `false` — можно.

Методы тоже одинаковы:

- `blur()` — убирает фокус ввода с текущего элемента формы;
- `focus()` — помещает фокус на текущий элемент формы;
- `select()` — выделяет текст в поле.

Обоими элементами поддерживаются следующие события:

- `onblur` — происходит при потере фокуса элементом формы;

- onchange — наступает после изменения данных в поле, при переводе фокуса ввода на другой элемент либо при отправке данных формы. Наступает перед событием onblur;
- onfocus — возникает при получении фокуса ввода элементом формы.

Кроме перечисленных событий можно использовать стандартные события мыши и клавиатуры *(см. разд. 3.16.2 и 3.16.3)*.

В качестве примера рассмотрим форму ввода E-mail и пароля с проверкой правильности ввода (листинг 3.67). Если данные введены неправильно, то при отправке формы:

- поле выделяется розовым цветом;
- текст в поле выделяется;
- выводится сообщение об ошибке;
- отправка формы прерывается.

Поле **Повтор E-mail** запрещено для редактирования. При вводе адреса электронной почты данные автоматически копируются из поля **E-mail** в поле **Повтор E-mail**.

Листинг 3.67. Форма ввода E-mail и пароля с проверкой правильности ввода

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Пример использования текстовых полей</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_submit() {
   var pole1 = document.getElementById("pole1");
   var pole2 = document.getElementById("pole2");
   pole1.style.backgroundColor = "#FFFFFF";
   pole2.style.backgroundColor = "#FFFFFF";
   var p = /^[a-z0-9_\.\-]+@([a-z0-9\-]+\.)+[a-z]{2,6}$/i;
   var Str = pole1.value;
   if (!p.test(Str)) {
      window.alert("Неверный адрес E-mail");
      pole1.style.backgroundColor = "#FFE4E1";
      pole1.select();
      return false;
   }
   p = /^[a-z0-9_\.\-]{6,16}$/i;
   Str = pole2.value;
   if (!p.test(Str)) {
      window.alert("Неверный пароль");
      pole2.style.backgroundColor = "#FFE4E1";
```

```
      pole2.select();
      return false;
   }
   var msg = "Вы ввели следующие данные:\n\n E-mail: ";
   msg += pole1.value + "\n Пароль: " + pole2.value;
   window.alert(msg);
   return true;
}
function f_reset() {
   document.getElementById("pole1").style.backgroundColor = "#FFFFFF";
   document.getElementById("pole2").style.backgroundColor = "#FFFFFF";
}
function f_load() {
   document.getElementById("pole3").readOnly = true;
}
function f_keyup() {
   document.getElementById("pole3").value =
        document.getElementById("pole1").value;
}
//-->
</script>
</head>
<body onload="f_load();">
 <form action="test.php" method="GET" name="frm" id="frm"
  onsubmit="return f_submit();" onreset="f_reset();">
  <div>
  E-mail:<br>
  <input type="text" name="pole1" id="pole1"
  style="background-color: #FFFFFF" onkeyup="f_keyup();"><br>
  Повтор E-mail:<br>
  <input type="text" name="pole3" id="pole3"
  style="background-color: #FFFFFF"><br>
  Пароль:<br>
  <input type="password" name="pole2" id="pole2"
  style="background-color: #FFFFFF"><br>
  <input type="reset" value="Очистить">
  <input type="submit" value="Отправить">
  </div>
 </form>
</body>
</html>
```

3.18.7. Поле для ввода многострочного текста. Добавление слов из текстового поля в поле *<textarea>*

Поле для ввода многострочного текста, определяемое парным тегом `<textarea>`, поддерживает те же свойства, методы и события, что и простое поле ввода

(см. разд. 3.16.6), за исключением свойства `maxLength`. Кроме того, поддерживается еще одно свойство:

- `wrap` — режим переноса слов. Может принимать следующие значения:
  - `off` — не переносить слова;
  - `physical` — слова переносятся как на экране, так и при передаче данных серверу;
  - `virtual` — слова переносятся только на экране, но не при передаче данных серверу.

Для примера рассмотрим возможность добавления слов из текстового поля в поле для ввода многострочного текста (листинг 3.68). Добавить слово можно с помощью кнопки **Добавить слово** или с помощью клавиши <Enter>. Так как по умолчанию нажатие клавиши <Enter> приводит к отправке данных формы, то всплывание события прерывается с помощью присвоения значения `false` свойству `returnValue` объекта `event`. При нажатии кнопки **Значение поля** выводится текущее значение тега `<textarea>`.

Листинг 3.68. Добавление слов из текстового поля в поле `<textarea>`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Пример использования поля &lt;TEXTAREA&gt;</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_submit() {
   var value1 = document.getElementById("pole1").value;
   window.alert("Текущее значение: \n" + value1);
   return false;
}
function f_click() {
   var pole2 = document.getElementById("pole2");
   var text1 = pole2.value;
   if (text1 != "") {
      document.getElementById("pole1").value += text1 + "\n";
      pole2.value = "";
      pole2.focus();
   }
   else {
      window.alert("Поле не заполнено!");
      pole2.focus();
   }
}
```

```
function f_press(e) {
   e = e || window.event;
   if (e.keyCode==13) {
      f_click();
      if (e.preventDefault) e.preventDefault();
      else e.returnValue = false;
   }
}
//-->
</script>
</head>
<body>
 <form action="test.php" method="GET" name="frm" id="frm"
  onsubmit="return f_submit();">
 <div>
  Слово:<br>
  <input type="text" name="pole2" id="pole2"
   onkeypress="f_press(event);"><br>
  <textarea name="pole1" id="pole1" cols="15" rows="10"></textarea>
  <br><input type="button" value="Добавить слово"
   onclick="return f_click();"><br>
  <input type="submit" value=" Значение поля ">
 </div>
 </form>
</body>
</html>
```

3.18.8. Список с возможными значениями. Возможность добавления нового пункта. Применение списков вместо гиперссылок

Свойства объекта списка:

- `disabled` — запрет доступа: если задано значение `true`, то список является неактивным (отображается серым цветом);
- `form` — ссылка на форму, в которой находится элемент;
- `length` — количество пунктов в списке (доступно и для записи);
- `multiple` — разрешение множественного выделения: `true`, если из списка можно выбрать сразу несколько элементов одновременно;
- `name` — имя элемента;
- `options` — ссылка на коллекцию пунктов в списке;
- `selectedIndex` — номер выбранного пункта (нумерация начинается с нуля);
- `size` — число одновременно видимых элементов списка;

- `type` — тип элемента формы (`select-multiple` или `select-one`);
- `value` — значение пункта, выбранного в списке.

Свойства пункта списка:

- `defaultSelected` — пункт списка, выбранный изначально;
- `index` — номер пункта в списке;
- `selected` — признак выделения: `true`, если пункт выбран в списке;
- `disabled` — если задано значение `true`, то пункт списка является неактивным (отображается серым цветом). Свойство поддерживается Web-браузером Internet Explorer, начиная с версии 8.0;
- `text` — текст пункта списка;
- `value` — значение пункта, выбранного в списке.

Методы:

- `blur()` — убирает фокус ввода с текущего элемента формы;
- `focus()` — помещает фокус на текущий элемент формы.

События:

- `onblur` — наступает при потере фокуса элементом формы;
- `onchange` — происходит после выбора нового пункта списка;
- `onfocus` — наступает при получении фокуса ввода элементом формы.

Кроме перечисленных событий можно использовать стандартные события мыши и клавиатуры.

Рассмотрим пример работы со списками. Документ, приведенный в листинге 3.69, демонстрирует следующие возможности:

- добавление нового пункта списка. При заполнении первого поля и нажатии клавиши <Enter> фокус ввода перемещается во второе поле. При заполнении второго поля и нажатии клавиши <Enter> введенные значения добавляются в список. Вместо клавиши <Enter> можно воспользоваться кнопкой **Добавить**;
- получение всех выбранных значений из списка с возможностью множественного выбора;
- применение взаимосвязанных списков и получение значения выбранного пункта. При выборе элемента в первом списке загружаются соответствующие элементы во второй список. При выборе элемента во втором списке выводится сообщение со значением выбранного пункта;
- применение списков вместо гиперссылок. При выборе элемента списка загружается Web-страница, находящаяся по указанному в параметре `value` URL-адресу.

Листинг 3.69. Обработка списков

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

```
<html>
<head>
 <title>Пример обработки списков</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<form action="test.php" method="GET" name="frm" id="frm">
<!-- Добавление пункта в список  -->

<script type="text/javascript">
<!--
function f_click() {
   var pole1 = document.getElementById("pole1");
   var pole2 = document.getElementById("pole2");
   var select1 = document.getElementById("select1");
   if (pole1.value != "" && pole2.value != "") {
      var i = select1.length++;
      select1.options[i].text = pole1.value;
      select1.options[i].value = pole2.value;
      pole1.value = "";
      pole2.value = "";
      pole1.focus();
   }
   else {
      window.alert("Поле не заполнено!");
      pole1.focus();
   }
}
function f_press1(e) {
   e = e || window.event;
   if (e.keyCode==13) {
      document.getElementById("pole2").focus();
      if (e.preventDefault) e.preventDefault();
      else e.returnValue = false;
   }
}
function f_press2(e) {
   e = e || window.event;
   if (e.keyCode==13) {
      f_click();
      if (e.preventDefault) e.preventDefault();
      else e.returnValue = false;
   }
}
//-->
</script>
```

```
<div>
<b>Добавление пункта в список:</b><br><br>
Текст пункта:<br>
<input type="text" name="pole1" id="pole1" onkeypress="f_press1(event);">
<br>Значение пункта:<br>
<input type="text" name="pole2" id="pole2" onkeypress="f_press2(event);">
<br><select name="select1" id="select1">
</select><br>
<input type="button" value="Добавить" onclick="f_click();"><br><br>

<!-- Список со множественным выбором  -->

<script type="text/javascript">
<!--
function f_multi() {
   var msg = "";
   var select2 = document.getElementById("select2");
   var count = select2.length;
   for (var i=0; i<count; i++) {
      if (select2.options[i].selected) {
         msg += select2.options[i].value + " - ";
         msg += select2.options[i].text + "\n";
      }
   }
   window.alert(msg);
}
//-->
</script>
<b>Список со множественным выбором:</b><br><br>
<select name="select2" id="select2" size="5" multiple>
<option value="1" selected>Элемент1</option>
<option value="2">Элемент2</option>
<option value="3">Элемент3</option>
<option value="4">Элемент4</option>
<option value="5">Элемент5</option>
<option value="6">Элемент6</option>
</select><br>
<input type="button" value="Значения списка"
onclick="f_multi();"><br><br>

<!-- Взаимосвязанные списки  -->

<script type="text/javascript">
<!--
var Mass = [];
Mass[1] = [ "Тема1 Элемент1", "Тема1 Элемент2" ];
Mass[2] = [ "Тема2 Элемент1", "Тема2 Элемент2", "Тема2 Элемент3" ];
```

```
var value1 = [];
value1[1] = [ "1", "2" ];
value1[2] = [ "3", "4", "5" ];
function f_change() {
   var index = document.getElementById("select3").value;
   var select4 = document.getElementById("select4");
   var count = Mass[index].length;
   select4.length = count;
   for (i=0; i<count; i++) {
      select4.options[i].value = value1[index][i];
      select4.options[i].text = Mass[index][i];
   }
}
function f_change2() {
   var sel = document.getElementById("select4");
   var msg = "Значение: " + sel.options[sel.selectedIndex].value;
   msg += "\nТекст: " + sel.options[sel.selectedIndex].text;
   window.alert(msg);
}
//-->
</script>
<b>Взаимосвязанные списки:</b><br><br>
<select name="select3" id="select3" size="5" onchange="f_change();">
<option value="1">Тема1</option>
<option value="2">Тема2</option>
</select><br>
<select name="select4" id="select4" onchange="f_change2();">
<option value="1" selected>Тема1 Элемент1</option>
<option value="2">Тема1 Элемент2</option>
</select><br><br>

<!-- Переход на указанный сайт -->

<b>Переход на указанный сайт:</b><br><br>
<select
onchange="top.location.href=this.options[this.selectedIndex].value;">
<option value="http://www.mail.ru/" selected>Национальная почта Mail.ru
</option>
<option value="http://www.rambler.ru/">Рамблер</option>
</select>
</div>
</form>
</body>
</html>
```

3.18.9. Флажок и переключатели. Получение значения выбранного переключателя при помощи цикла и проверка установки флажка

Флажки и переключатели имеют следующие свойства:

- `value` — значение текущего элемента формы;
- `checked` — признак отметки: `true`, если флажок или переключатель находится во включенном состоянии;
- `defaultChecked` — флажок или переключатель установлен по умолчанию. Возвращает `true` или `false`;
- `disabled` — признак запрета: если задано значение `true`, то элемент является неактивным (отображается серым цветом);
- `indeterminate` — флажок находится в неопределенном состоянии (закрашивается серым). Возвращает `true` или `false`;
- `form` — ссылка на форму, в которой находится элемент;
- `name` — имя элемента;
- `type` — тип элемента формы.

Методы:

- `blur()` — убирает фокус ввода с текущего элемента формы;
- `focus()` — помещает фокус на текущий элемент формы.

События:

- `onblur` — наступает при потере фокуса элементом формы;
- `onclick` — возникает при выборе элемента;
- `onfocus` — происходит при получении фокуса ввода элементом формы.

Чтобы найти выбранный элемент-переключатель в группе, необходимо перебрать все переключатели в цикле. Получить значение выбранного переключателя можно через метод `item()`, указав индекс элемента в группе. Рассмотрим это на примере (листинг 3.70).

Листинг 3.70. Обработка флажков и переключателей

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Пример использования флажков и переключателей</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
```

```
<!--
function f_click() {
   var msg = "";
   if (document.getElementById("check1").checked) {
      msg = "Флажок установлен\n";
      msg += "Значение: " + document.getElementById("check1").value + "\n";
   }
   else {
      msg = "Флажок снят\n";
   }
   var value1 = "";
   var count = document.frm.radio1.length;
   for (i=0; i<count; i++) {
      if (document.frm.radio1.item(i).checked) {
         value1 = document.frm.radio1.item(i).value;
         break;
      }
   }
   if (value1 == "male") {
      msg += "Пол: Мужской\n";
   }
   else {
      msg += "Пол: Женский\n";
   }
   window.alert(msg);
}
//-->
</script>
</head>
<body>
<form action="test.php" method="GET" name="frm" id="frm">
<div>
<input type="checkbox" name="check1" id="check1" value="yes" checked>
Текст<br><br>
Укажите ваш пол:<br>
<input type="radio" name="radio1" id="radio1" value="male"
checked>Мужской
<input type="radio" name="radio1" id="radio2" value="female">Женский
<br><br>
<input type="button" value="Вывести значения" onclick="f_click();">
</div>
</form>
</body>
</html>
```

3.18.10. Кнопки. Обработка нажатия кнопки. Деактивация кнопки. Создание клавиши быстрого доступа и вывод текста на кнопке определенным цветом

Кнопки поддерживают следующие свойства:

- `value` — текст, отображаемый на кнопке;
- `disabled` — признак запрета: если задано значение `true`, то кнопка является неактивной (отображается серым цветом);
- `form` — ссылка на форму, в которой находится элемент;
- `name` — имя элемента;
- `type` — тип элемента формы.

Методы традиционны:

- `blur()` — убирает фокус ввода с текущего элемента формы;
- `focus()` — помещает фокус на текущий элемент формы.

События:

- `onblur` — наступает при потере фокуса элементом формы;
- `onclick` — возникает при нажатии кнопки;
- `onfocus` — происходит при получении фокуса ввода элементом формы.

В приведенном далее примере (листинг 3.71) кнопка изначально не активна. При вводе в текстовое поле кнопка активируется. При нажатии кнопки текст, введенный в текстовое поле, отображается на кнопке. Текстовое поле очищается, и кнопка деактивируется.

Листинг 3.71. Обработка нажатия кнопки

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Пример использования кнопок</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_up() {
   if (document.getElementById("text1").value == "") {
      document.getElementById("button1").disabled = true;
   }
   else {
      document.getElementById("button1").disabled = false;
   }
}
```

```
function f_click() {
   document.getElementById("button1").value =
        document.getElementById("text1").value;
   document.getElementById("text1").value = "";
   document.getElementById("button1").disabled = true;
}
//-->
</script>
</head>
<body>
 <form action="test.php" method="GET" onsubmit="return false;">
  <div>
  <input type="text" name="text1" id="text1" onkeyup="f_up();"><br>
  <input type="button" value="Изменить текст на кнопке"
   onclick="f_click();" id="button1" disabled>
  </div>
 </form>
</body>
</html>
```

Обычная командная кнопка может быть вставлена в Web-страницу не только с помощью тега `<input>`, но и с помощью парного тега `<button>`. При использовании этого тега текст на кнопке можно сделать цветным, а также можно задать клавишу быстрого доступа.

Переделаем пример из листинга 3.71. Вместо тега `<input>` используем тег `<button>` и добавим клавишу быстрого доступа (листинг 3.72). При одновременном нажатии клавиши, указанной в параметре `accesskey`, и клавиши <Alt> выполняется функция `f_click()`.

Листинг 3.72. Использование тега `<button>`

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Пример использования тега &lt;button&gt;</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_up() {
   if (document.getElementById("text1").value == "") {
       document.getElementById("button1").disabled = true;
   }
   else {
      document.getElementById("button1").disabled = false;
   }
}
```

```
function f_click() {
   document.getElementById("span1").innerText =
            document.getElementById("text1").value;
   document.getElementById("text1").value = "";
   document.getElementById("button1").disabled = true;
}
//-->
</script>
</head>
<body>
<form action="test.php" method="GET" onsubmit="return false;">
<div>
<input type="text" name="text1" id="text1" onkeyup="f_up();"><br>
<button accesskey="т" onclick="f_click();" id="button1" disabled>
<span id="span1" style="color: red">
<span style="text-decoration: underline">Т</span>екст красного цвета
</span></button>
</div>
</form>
</body>
</html>
```

3.18.11. Проверка корректности данных. Создание формы регистрации пользователя

Рассмотрим форму регистрации пользователя с проверкой корректности введенных данных (листинг 3.73).

Листинг 3.73. Проверка данных на стороне клиента

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Регистрация пользователя</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<script type="text/javascript">
<!--
function f_submit() {
   var name1 = document.getElementById("name1");
   if (name1.value=="") {
      window.alert("Введите имя");
      name1.focus();
      return false;
   }
```

```
   var fam1 = document.getElementById("fam1");
   if (fam1.value=="") {
      window.alert("Введите фамилию");
      fam1.focus();
      return false;
   }
   var age1 = document.getElementById("age1");
   var p = /^[0-9]{1,3}$/;
   if (!p.test(age1.value)) {
      window.alert("Неверный возраст");
      age1.focus();
      return false;
   }
   var mail1 = document.getElementById("mail1");
   p = /^[a-z0-9_\.\-]+@([a-z0-9\-]+\.)+[a-z]{2,6}$/i;
   if (!p.test(mail1.value)) {
      window.alert("Неверный адрес E-mail");
      mail1.focus();
      return false;
   }
   var pass1 = document.getElementById("pass1");
   var pass2 = document.getElementById("pass2");
   p = /^[a-z0-9_\.\-]{6,16}$/i;
   if (!p.test(pass1.value)) {
      window.alert("Неверный пароль");
      pass1.focus();
      return false;
   }
   else if (pass1.value != pass2.value) {
      window.alert("Пароли должны совпадать");
      pass1.focus();
      return false;
   }
   return true;
}
//-->
</script>
</head>
<body>
 <h2>Регистрация пользователя</h2>
 <form action="test.php" method="POST" name="form1"
  onsubmit="return f_submit();">
  <div>
   Имя:<br>
   <input type="text" name="name1" id="name1"><br>
   Фамилия:<br>
   <input type="text" name="fam1" id="fam1"><br>
```

```
    Возраст:<br>
    <input type="text" name="age1" id="age1"><br>
    E-mail:<br>
    <input type="text" name="mail1" id="mail1"><br>
    Пароль:<br>
    <input type="password" name="pass1" id="pass1"><br>
    Повторите пароль:<br>
    <input type="password" name="pass2" id="pass2">
    <br><br>
    <input type="reset" value="Очистить">
    <input type="submit" value="Отправить">
  </div>
 </form>
</body>
</html>
```

Итак, все данные проверены. Что же происходит после отправки данных формы? Давайте рассмотрим содержимое файла test.php (листинг 3.74).

Листинг 3.74. Проверка данных на стороне сервера

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Результаты регистрации</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
<div>
<?php
if (!isset($_POST['name1'])) echo "Форма не отправлена";
else {
   // Создаем короткие имена переменных
   $name = (isset($_POST['name1'])) ? $_POST['name1'] : '';
   $fam = (isset($_POST['fam1'])) ? $_POST['fam1'] : '';
   $age = (isset($_POST['age1'])) ? (int)$_POST['age1'] : 0;
   $mail = (isset($_POST['mail1'])) ? $_POST['mail1'] : '';
   $pass1 = (isset($_POST['pass1'])) ? $_POST['pass1'] : '';
   $pass2 = (isset($_POST['pass2'])) ? $_POST['pass2'] : '';
   // Если "магические" кавычки включены, то удаляем слэши
   if (get_magic_quotes_gpc()) {
      $user = stripslashes($user);
      $fam = stripslashes($fam);
      $email = stripslashes($email);
      $pass1 = stripslashes($pass1);
      $pass2 = stripslashes($pass2);
   }
```

```
    $err = "";
    if (strlen($name)>100 || strlen($name)<2) {
       $err .= "Недопустимая длина поля 'Имя'.<br>";
    }
    if (strlen($fam)>100 || strlen($fam)<2) {
       $err .= "Недопустимая длина поля 'Фамилия'.<br>";
    }
    if (!preg_match('/^[0-9]{1,3}$/s', $age) || $age==0) {
       $err .= "Неверный возраст.<br>";
    }
    if (!preg_match('/^[a-z0-9_\.\-]+@([a-z0-9\-]+\.)+[a-z]{2,6}$/is', $mail)
        || strlen($mail)>70) {
       $err .= "Неверный адрес E-mail.<br>";
    }
    if (!preg_match('/^[a-z0-9_\.\-]{6,16}$/is', $pass1)) {
       $err .= "Неверный пароль.<br>";
    }
    else {
       if ($pass1 != $pass2) {
          $err .= "Пароли должны совпадать.<br>";
       }
    }
    if ($err=="") { // Если ошибок нет
    // Добавляем данные в базу данных и отправляем подтверждение на E-mail
       echo "<b>Регистрация прошла успешно</b>";
    }
    else {
       echo "<span style='color: red'>При заполнении формы были допущены ";
       echo "следующие ошибки:</span><br><br>";
       echo $err;
    }
}
?>
</div>
</body>
</html>
```

Эта программа, написанная на PHP, очень напоминает программу на JavaScript. Как и в JavaScript, код программы может внедряться в HTML-документ. Только вместо открывающего тега `<script>` используется дескриптор `<?php`, а вместо закрывающего тега `</script>` — дескриптор `?>`. Но главным отличием является то, что программа на PHP выполняется не на компьютере пользователя, а на Web-сервере.

Как видно из примера, все имена полей, заданные с помощью параметра `name`, доступны через переменную окружения `$_POST`. Более того, если в файле конфигурации включена поддержка глобальных переменных, то все имена доступны как обычные переменные. Что же происходит дальше? Мы опять проверяем введенные

данные... но зачем? Ведь мы уже проверили их с помощью JavaScript... Как уже говорилось ранее, любой пользователь может отключить JavaScript в настройках Web-браузера. Поэтому проверять данные нужно обязательно. Так как файл test.php выполняется не на компьютере пользователя, а на сервере, то проверка будет выполнена независимо от программного обеспечения компьютера пользователя, и отключить ее пользователь не сможет.

При успешной проверке данные обычно добавляются в базу данных, и отправляется письмо с подтверждением регистрации.

Что будет, если сохранить файл test.php на локальном компьютере, а затем отправить данные формы этому файлу? Вместо надписи "Регистрация прошла успешно" мы получим нечто подобное:

```
100 || strlen($name)<2) { $err .= "Недопустимая длина поля 'Имя'.
"; } if (strlen($fam)>100 || strlen($fam)<2) { $err .= "Недопустимая длина поля
'Фамилия'.
"; } if (!preg_match('/^[0-9]{1,3}$/s', $age) || $age==0) { $err .= "Неверный
 возраст.
"; } if (!preg_match('/^[a-z0-9_\.\-]+@([a-z0-9\-]+\.)+[a-z]{2,6}$/is', $mail)
|| strlen($mail)>70) { $err .= "Неверный адрес E-mail.
"; } if (!preg_match('/^[a-z0-9_\.\-]{6,16}$/is', $pass1)) { $err .= "Неверный
 пароль.
"; } else { if ($pass1 != $pass2) { $err .= "Пароли должны совпадать.
"; } } if ($err=="") { // Если ошибок нет // Добавляем данные в базу данных и
отправляем подтверждение на E-mail echo "Регистрация прошла успешно"; } else {
 echo "При заполнении формы были допущены "; echo "следующие ошибки:
"; echo $err; } } ?>
```

Иными словами, для выполнения программы, написанной на языке PHP, необходимо специальное программное обеспечение. Какое программное обеспечение необходимо, где его найти и как установить, мы рассмотрим в следующей главе.

3.19. Пользовательские объекты

В предыдущих разделах мы рассмотрели возможности встроенных объектов. Язык JavaScript предоставляет также возможность создания пользовательских объектов. Тем не менее следует заметить, что в JavaScript нет полноценной поддержки объектно-ориентированного программирования, такой как в языках C++ или Java.

3.19.1. Создание объектов

Создать новый объект можно с помощью встроенного класса `Object`:

```
var car = new Object();
car.model = "ВАЗ-2109"; // Сохранили строку
car.year = 2007; // Сохранили число
car.getModel = function() { // Сохранили ссылку на функцию
```

```
   return this.model;
};
// Вывод значений
window.alert(car.model); // "ВАЗ-2109"
window.alert(car.year); // 2007
window.alert(car.getModel()); // "ВАЗ-2109"
```

После создания объекта в переменной `car` сохраняется ссылка на него. Используя точечную нотацию, можно добавить свойство (переменную внутри объекта). В качестве значения свойства может быть указан любой тип данных: число, строка, массив или другой объект. Если в качестве значения указать ссылку на функцию, то такое свойство становится методом объекта, внутри которого доступен указатель (`this`) на текущий объект.

Создать объект можно также с помощью фигурных скобок:

```
var car = {
   model: "ВАЗ-2109", // Сохранили строку
   year: 2007, // Сохранили число
   getModel: function() { // Сохранили ссылку на функцию
      return this.model;
   }
};
// Вывод значений
window.alert(car.model); // "ВАЗ-2109"
window.alert(car.year); // 2007
window.alert(car.getModel()); // "ВАЗ-2109"
```

В этом случае значение свойства указывается после двоеточия, а пары "свойство/значение" перечисляются через запятую. Если между фигурными скобками нет никаких выражений, то создается пустой объект:

```
var obj = {}; // Пустой объект
```

При создании объектов следует учитывать один очень важный момент. Например, нам необходимо определить два нустых объекта, которые в дальнейшем будут использоваться раздельно. Очень силен соблазн написать следующим образом:

```
var obj1 = obj2 = {}; // Якобы определили два объекта
```

Проблема заключается в том, что в данном примере создается только один объект, а ссылка на него сохраняется в двух переменных. Таким образом, все изменения `obj1` будут отражаться и на переменной `obj2`:

```
var obj1 = obj2 = {}; // Якобы определили два объекта
obj1.test = "Это значение свойства test объекта obj1";
window.alert(obj2.test);
// Выведет: Это значение свойства test объекта obj1
```

Помните, что присваивание и сравнение объектов производится по ссылке, а не по значению. Поэтому создавать объекты необходимо раздельно:

```
var obj1 = {};
var obj2 = {};
obj1.test = "Это значение свойства test объекта obj1";
window.alert(obj2.test); // Выведет: undefined
```

Если после ключевого слова `new` указана функция, то она становится конструктором объекта, которому можно передать начальные данные при инициализации. Внутри конструктора доступен указатель (`this`) на текущий объект:

```
function Cars(m, y) { // Конструктор объекта
   this.model = m;
   this.year = y;
   this.getModel = function() {
      return this.model;
   }
}
// Создание экземпляра
var car = new Cars("ВАЗ-2109", 2007);
// Вывод значений
window.alert(car.model); // "ВАЗ-2109"
window.alert(car.year); // 2007
window.alert(car.getModel()); // "ВАЗ-2109"
```

Все рассмотренные варианты позволяли создавать свойства и методы экземпляра объекта. Тем не менее можно также создать свойства и методы, связанные с самим объектом, а не с его экземпляром:

```
function Cars() { }
Cars.model = "ВАЗ-2109";
Cars.year = 2007;
Cars.getModel = function() {
   return Cars.model;
};
```

Получить значения свойств и вызвать метод можно без создания экземпляра:

```
window.alert(Cars.model); // "ВАЗ-2109"
window.alert(Cars.year); // 2007
window.alert(Cars.getModel()); // "ВАЗ-2109"
```

Как видно из примеров, чтобы обратиться к свойству, следует указать его название после точки. Доступ к методам осуществляется таким же образом, но после имени метода необходимо указать круглые скобки. Кроме точечной нотации к свойствам и методам можно обратиться как к элементам ассоциативного массива. В этом случае название задается внутри квадратных скобок:

```
window.alert(car["model"]); // "ВАЗ-2109"
window.alert(car["year"]); // 2007
window.alert(car["getModel"]()); // "ВАЗ-2109"
```

Обратите внимание на то, что название указывается в виде строки, которую можно изменить внутри программы динамически. Это обстоятельство позволяет обратиться к свойствам, названия которых заранее неизвестны. Выведем названия всех свойств и их значения с помощью цикла `for...in`:

```
function Cars(m, y) { // Конструктор объекта
   this.model = m;
   this.year = y;
}
// Создание экземпляра
var car = new Cars("ВАЗ-2109", 2007);
// Вывод всех значений
for (var P in car) {
   // Переменной P на каждой итерации присваивается
   // название свойства объекта
   window.alert(P + " = " + car[P]);
}
```

Оператор `in` позволяет также проверить существование свойства (включая унаследованные) у объекта. Если свойство существует, то возвращается значение `true`:

```
if ("model" in car) window.alert("Свойство определено");
else window.alert("Нет");
```

Проверить наличие не унаследованного свойства позволяет метод `hasOwnProperty()`. В качестве значения указывается название свойства:

```
if ("toString" in car) window.alert("Свойство определено");
else window.alert("Нет");
// Выведет: "Свойство определено"
if (car.hasOwnProperty("toString"))
   window.alert("Свойство определено");
else window.alert("Нет");
// Выведет: "Нет", т. к. toString является унаследованным свойством
if (car.hasOwnProperty("getModel"))
   window.alert("Свойство определено");
else window.alert("Нет");
// Выведет: "Свойство определено"
```

Если название метода указать в условии без круглых скобок, то это позволит проверить наличие метода:

```
if (car.getModel) window.alert("Метод определен");
else window.alert("Нет");
```

Обратите внимание на то, что проверять таким образом наличие свойства нельзя, т. к. значение 0 будет интерпретировано как `false`.

С помощью оператора `instanceof` можно проверить принадлежность экземпляра какому-либо объекту:

```
if ((typeof car == "object") && (car instanceof Cars))
   window.alert("Экземпляр car принадлежит объекту Cars");
else window.alert("Нет");
```

Удалить свойство позволяет оператор `delete`:

```
delete car.model;
```

3.19.2. Прототипы

В предыдущем разделе мы определяли метод `getModel()` внутри конструктора:

```
function Cars(m, y) { // Конструктор объекта
   this.model = m;
   this.year = y;
   this.getModel = function() { // Метод
      return this.model;
   }
}
```

Подобное решение не является эффективным. Предположим, необходимо составить массив, описывающий тысячу автомобилей. Свойства `model` и `year` в этом случае будут содержать разные значения, а вот метод `getModel()` во всех этих объектах один и тот же.

Использование прототипов позволяет определить метод вне конструктора. При создании объекта наследуются все свойства, которые имеются в прототипе. Таким образом, метод `getModel()` будет определен один раз, но будет наследоваться всеми экземплярами объекта.

Для добавления метода в прототип используется свойство `prototype`:

```
function Cars(m, y) { // Конструктор объекта
   this.model = m;
   this.year = y;
}
Cars.prototype.getModel = function() {
   return this.model;
}
var car1 = new Cars("Москвич-412", 1978);
window.alert(car1.getModel()); // "Москвич-412"
var car2 = new Cars("ВАЗ-2109", 2002);
window.alert(car2.getModel()); // "ВАЗ-2109"
```

Как уже говорилось, свойства, определенные в прототипе, наследуются всеми экземплярами. Таким образом, метод `getModel()` доступен для перебора в цикле `for...in`, а также успешно проверяется на наличие с помощью оператора `in`. Тем не менее метод `hasOwnProperty()` позволяет определить, что метод является унаследованным:

```
if ("getModel" in car1) window.alert("Метод определен");
else window.alert("Нет");
// Выведет: "Метод определен"
if (car1.hasOwnProperty("getModel"))
   window.alert("Метод определен");
else window.alert("Нет");
// Выведет: "Нет", т. к. метод унаследован
```

Любой созданный объект автоматически наследует свойства класса `Object`. Например, при попытке вывести значение экземпляра объекта в диалоговом окне вызывается метод `toString()`, который должен возвращать значение в виде строки. Для примера выведем текущее значение:

```
var car1 = new Cars("Москвич-412", 1978);
window.alert(car1);
```

В результате в диалоговом окне получим следующий результат:

```
[object Object]
```

С помощью прототипов можно переопределить этот метод таким образом, чтобы выводилось нужное нам значение:

```
Cars.prototype.toString = function() {
   return "Модель: " + this.model + " Год выпуска: " + this.year;
}
var car1 = new Cars("Москвич-412", 1978);
window.alert(car1);
```

В результате в диалоговом окне получим следующий результат:

```
Модель: Москвич-412 Год выпуска: 1978
```

При попытке выполнить арифметическую операцию вызывается метод `valueOf()`, который должен возвращать значение в виде числа. Для примера переопределим метод таким образом, чтобы он возвращал сколько лет автомобилю:

```
Cars.prototype.valueOf = function() {
   return 2009 - this.year;
}
var car1 = new Cars("Москвич-412", 1978);
window.alert(car1 * 1); // 31
```

Практически все встроенные объекты JavaScript (например, `String`, `Array`) имеют свойство `prototype`. С его помощью можно расширить возможности встроенных классов, например, добавить новый метод. В качестве примера добавим метод `inArray()` в класс `Array`. Этот метод будет производить поиск значения в массиве и возвращать индекс первого вхождения. Если вхождение не найдено, то метод вернет значение `-1`:

```
Array.prototype.inArray = function(elem) {
   for (var i=0, len=this.length; i<len; i++) {
```

```
      if (this[i]===elem) return i;
   }
   return -1;
}
var arr = [ 1, 2, 3, 4, 5, 1 ];
var pos = arr.inArray(5);
if (pos != -1) window.alert("Индекс элемента " + pos);
else window.alert("Не найдено");
// Выведет: "Индекс элемента 4"
```

Примечание

Не рекомендуется расширять возможности встроенных классов, т. к. другие программисты могут прийти в недоумение, увидев новый метод.

3.19.3. Пространства имен

Предположим, программист написал функцию с названием `inArray()`. Через некоторое время потребовалось подключить модуль стороннего разработчика, в котором все функции объявлены в глобальной области видимости. Если в этом модуле объявлена функция с названием `inArray()`, то возникнет конфликт имен. Следует заметить, что никакого сообщения об ошибке в данном случае выведено не будет. Функция, которая объявлена последней, просто переопределит уже существующую функцию. Далее все зависит от частоты использования функции. Все выражения, которые зависят от этой функции, станут работать некорректно. В итоге будет получен результат, который не планировался, или программа завершится с критической ошибкой.

Чтобы избежать подобной ситуации, следует строго придерживаться концепции пространств имен. Согласно этой концепции модуль может импортировать в глобальную область видимости только один идентификатор. Следует заметить, что это требование касается не только модулей сторонних разработчиков, но и относится к вашим собственным программам.

В языке JavaScript в качестве пространства имен используются объекты. Созданный экземпляр помещается в глобальную область видимости, а остальные идентификаторы доступны через свойства объекта:

```
var myModule = {}; // Объявление пространства имен
myModule.test = function() {
   window.alert("Это функция test");
}
myModule.inArray = function() {
   window.alert("Это функция inArray");
}
myModule.test();
myModule.inArray();
```

В этом примере функция `inArray()` расположена внутри пространства `myModule`. Поэтому конфликт имен сводится к минимуму. Однако может возникнуть ситуация, когда пространства имен называются одинаково. В этом случае решением является создание вложенных объектов. Очень часто название пространства имен совпадает с названием сайта разработчика. В качестве основного объекта используется название зоны, а вложенный объект носит название домена. Например, для сайта **http://wwwadmin.ru/** создание пространства имен будет выглядеть так:

```
var ru; // Объявляем, иначе будет ошибка при проверке
if (!ru) ru = {}; // Объявление пространства имен
else if (typeof ru != "object")
   throw new Error("Идентификатор ru не является объектом");
if (ru.wwwadmin)
   throw new Error("Пространство имен уже занято");
ru.wwwadmin = { // Объявление вложенного пространства имен
   test: function() {
      window.alert("Это функция test");
   },
   inArray: function() {
      window.alert("Это функция inArray");
   }
};
ru.wwwadmin.test();
ru.wwwadmin.inArray();
```

Таким образом, если домен принадлежит вам, никакого конфликта имен не будет, но пользоваться таким длинным названием не очень удобно. Учитывая, что присваивание объектов производится по ссылке, а не по значению, то данная проблема решается просто. В программе определяется короткий идентификатор и в нем сохраняется ссылка на объект:

```
var $ = ru.wwwadmin;
$.test();
$.inArray();
```

Кроме того, можно использовать анонимную функцию, в параметре которой указывается короткий идентификатор, а при вызове функции передается ссылка на объект, являющийся пространством имен:

```
(function($) {
   $.test();
   $.inArray();
})(ru.wwwadmin);
```

В этом примере идентификатор `$` будет доступен только внутри анонимной функции, а так как функция не имеет названия, в глобальной области видимости никакой идентификатор не сохраняется.

3.20. AJAX

AJAX (Asynchronous JavaScript and XML, асинхронный JavaScript и XML) — это технология программной подгрузки произвольных данных для их вывода на страницу (возможно, после обработки) или использования в вычислениях. Программа инициирует загрузку файла с данными, а потом считывает его содержимое и пускает в обработку — и все это без перезагрузки самой страницы.

Так можно загружать данные трех различных типов:

- Фрагменты HTML-кода или обычного текста, которые мы можем просто вывести на экран, вставив их в любой контейнер.
- Данные, закодированные с помощью языка XML. Понятно, что просто вывести их на экран нельзя, и нам потребуется раскодировать их и преобразовать в подходящий для вывода в составе страницы формат — HTML-код.
- Данные, закодированные в формате JSON (о нем речь пойдет чуть позже). Их также потребуется раскодировать и преобразовать в HTML-код.

> ***Внимание!***
>
> Технология AJAX позволяет загружать данные исключительно с Web-сервера. Загрузка из локальных файлов во всех Web-обозревателях заблокирована в целях безопасности.

3.20.1. Подготовка к загрузке данных

Прежде чем загрузить какой-либо файл с применением технологии AJAX, нам следует получить объект, который, собственно, и выполнит его загрузку. Процесс его получения различается в зависимости от используемого Web-обозревателя.

Стандартный способ

В Firefox, Chrome, Opera, Safari и Internet Explorer, начиная с версии 7, загрузкой данных "заведует" класс `XMLHttpRequest`. Этот класс поддерживается самим Web-обозревателем и определен в стандарте DOM, поэтому способ загрузки данных с его помощью носит название стандартного.

Нам нужно лишь создать объект упомянутого класса оператором `new`. Никакие параметры при этом не указываются:

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

Класс `XMLHttpRequest` также доступен через одноименное свойство объекта `window`.

Способ, применяемый в Internet Explorer 5 и 6

Однако в Internet Explorer версий 5 и 6 класс `XMLHttpRequest` не поддерживается. Вместо этого следует использовать полностью аналогичный класс `Microsoft.XMLHTTP`, поддерживаемый отдельной программой, которая устанавливается совместно с Web-обозревателем.

Для получения класса, поддерживаемого сторонней программой, в Internet Explorer предусмотрена функция `ActiveXObject`. Имя класса передается ей единственным параметром в виде строки. А, получив класс, можно создать на его основе объект с помощью все того же оператора `new`:

```
var oAJAX = new ActiveXObject("Microsoft.XMLHTTP");
```

> ***Внимание!***
>
> В дальнейшем, описывая реализацию технологии AJAX и говоря о классе `XMLHttpRequest`, мы будем иметь в виду также и класс `Microsoft.XMLHTTP`, поскольку они полностью идентичны по своим возможностям.

Универсальный способ

На практике нам будет полезнее универсальный, кроссплатформенный способ, одинаково работающий во всех Web-обозревателях. Код, реализующий этот способ, приведен в листинге 3.75.

Листинг 3.75. Кроссплатформенная загрузка данных

```
if (window.XMLHttpRequest)
  var oAJAX = new XMLHttpRequest();
} else {
  var oAJAX = new ActiveXObject("Microsoft.XMLHTTP");
}
```

3.20.2. Отправка запроса

Получив объект, реализующий технологию AJAX, мы можем отправить Web-серверу запрос на загрузку файла с данными.

Синхронный или асинхронный запрос?

Существуют две разновидности запросов на получение данных AJAX, которые мы можем отправить:

- *синхронный запрос*, при котором Web-обозреватель приостанавливает выполнение программы и ждет, пока файл с данными не будет получен;
- *асинхронный запрос*, при котором Web-обозреватель продолжает выполнять программу, не дожидаясь получения запрошенного файла, а когда он, наконец, будет получен, генерирует особое событие (о нем мы поговорим далее).

На практике чаще встречаются асинхронные запросы, т. к. они позволяют получить данные по ходу выполнения прочего JavaScript-кода и, соответственно, не приводят к "зависанию" всей страницы. Но, к сожалению, Web-обозреватель может одновременно выполнять лишь один асинхронный запрос, поэтому, если нам понадобится подгрузить сразу несколько файлов, придется прибегнуть к синхронным запросам.

Задание параметров запроса

Сначала нам следует указать параметры отправляемого запроса: метод отсылки данных (GET или POST), интернет-адрес файла с данными или программы, которая сгенерирует эти данные, и вид запроса (синхронный или асинхронный). Все это выполняет метод `open` класса `XMLHttpRequest`:

```
<объект класса XMLHttpRequest>.open(<метод отправки данных>,
<интернет-адрес>, true|false)
```

Метод отправки данных указывается в виде строки `"GET"` или `"POST"`. *Интернет-адрес*, с которого запрашиваются данные, также указывается как строка. Если третьим параметром передано значение `true`, будет выполнен асинхронный запрос, если `false` — синхронный. Метод `open` не возвращает результат.

Примеры:

```
oAJAX.open("GET", "/fragments/fragment3.html", false);
```

Задаем параметры синхронного запроса на получение файла /fragments/fragment3.html, в котором хранится фрагмент HTML-кода для вывода на страницу.

```
oAJAX.open("POST", "search.php", true);
```

А здесь мы задаем параметры для асинхронного запроса на получение данных от программы search.php, причем входные данные для этой программы (они могут быть введены посетителем в специально предусмотренной для этого форме или сгенерированы скриптом) мы отошлем по методу `POST`.

Задание MIME-типа отправляемых данных

Если данные, которые мы собираемся подгрузить, генерируются выполняющейся на стороне сервера программой, эта программа для работы может требовать какую-либо входную информацию. Эта информация может как вводиться посетителем в специальной форме, так и генерироваться скриптом. И отправить мы ее можем методом `GET` или `POST`.

Если мы отправляем входные данные методом `POST`, то обязаны указать соответствующий MIME-тип этих данных. Для чего вызовем метод `setRequestHeader` класса `XMLHttpRequest`:

```
<экземпляр объекта XMLHttpRequest>.setRequestHeader(<имя параметра>,
<значение параметра>)
```

Оба параметра этого метода указываются в виде строк. В нашем случае именем параметра станет строка `"Content-type"` (именно этот параметр указывает MIME-тип данных), а его значением — наименование нужного метода отправки данных. Метод `setRequestHeader` не возвращает результат.

Пример:

```
oAJAX.setRequestHeader("Content-type",
"application/x-www-form-urlencoded");
```

Собственно отправка запроса

Вот теперь мы можем, наконец, отправить запрос. Выполняется это вызовом не возвращающего результат метода `send` класса `XMLHttpRequest`.

Пример:

```
oAJAX.send();
```

Отправка данных с запросом

Но как же отправить входную информацию для серверной программы, которая сгенерирует нам данные? Это зависит от метода отсылки входной информации, что мы указали первым параметром метода `open`.

- Если входные данные отправляются по методу `GET`, они просто добавляются к интернет-адресу, указанному вторым параметром метода `open`. Тогда метод `send` вызывается без указания параметров.
- Если входные данные отправляются по методу `POST`, эти данные указываются в качестве единственного параметра метода `send`. К запрашиваемому интернет-адресу они в этом случае не добавляются.

В качестве примера рассмотрим форму входа на сайт:

```
<form id="frmLogin" action="login.php"
enctype="application/x-www-form-urlencoded">
  <p>Имя: <input type="text" name="login"></p>
  <p>Пароль: <input type="password" name="password"></p>
  <p><input type="submit" name="submit" value="Войти"></p>
</form>
```

При нажатии кнопки **Войти** будет выполнен следующий скрипт, который создаст строку `sData` с введенными в форму данными, закодированными соответствующим образом:

```
var oForm = document.getElementById("frmLogin");
var cEls = oForm.elements;
var sData = "";
for (i in cEls) {
  if (s != "") s += "&";
  s += cEls[i].name + "=" + encodeURIComponent(cEls[i].value);
}
```

Тогда, чтобы отправить эти данные серверной программе с применением метода `GET`, мы напишем такой скрипт:

```
oAJAX("GET", oForm.action + "?" + sData, true);
oAJAX.send();
```

А вот скрипт для отправки данных по методу `POST`:

```
oAJAX("POST", oForm.action, true);
oAJAX.setRequestHeader("Content-type", oForm.enctype);
oAJAX.send(s);
```

3.20.3. Получение данных

Получив от нас AJAX-запрос на подгрузку данных, Web-сервер либо отправит нам запрошенный файл, либо вызовет серверную программу, которая получит отправленную нами входную информацию и сгенерирует результирующие данные, которые, опять же, будут отправлены нам. Осталась мелочь — получить и обработать их.

Оформление обработчика данных

Сначала нам следует написать код, который будет обрабатывать полученные данные, иначе говоря, обработчик данных. Этот код можно оформить двумя способами, первый из которых является универсальным, а второй применим лишь в случае синхронного запроса.

- Обработчик можно оформить в виде функции (обычной или анонимной), которую присвоить свойству `onreadystatechange` класса `XMLHttpRequest`. Такой обработчик будет универсальным, подходящим и для синхронного, и для асинхронного запроса.

  > ***Внимание!***
  >
  > **Присвоение функции-обработчика свойству `onreadystatechange` следует выполнять перед вызовом метода `send`. Существует вероятность того, что запрошенные данные загрузятся ранее, чем для них будет указан обработчик, и такую ситуацию лучше исключить.**

  Говорят, что при получении данных в случае асинхронного запроса Web-обозреватель генерирует событие `onreadystatechange`. Это событие и обрабатывает функция, присвоенная упомянутому свойству.

  Пример:

```
function getData() {
  // Здесь получаем и обрабатываем данные
}
oAJAX.open("GET", "search.php", true);
oAJAX.onreadystatechange = getData;
oAJAX.send();
```

- Если был выполнен синхронный запрос, код обработчика можно просто поместить после вызова метода `send`.

  Пример:

```
oAJAX.open("GET", "search.php", true);
oAJAX.send();
// Здесь получаем и обрабатываем данные
```

  Получаем и обрабатываем данные, полученные в результате синхронного запроса.

Определение успешного получения данных

Если мы используем первый способ оформления кода, который будет обрабатывать полученные данные, то должны приготовиться к тому, что событие `onreadystatechange` будет возникать всякий раз, когда изменяется состояние ожидания этих данных (когда запрос собственно отправляется, когда данные получены, но еще не обработаны, и др.). И еще нам следует иметь в виду, что Web-сервер или серверная программа может вернуть не только запрашиваемые нами данные, но и сообщение об ошибке.

В рассматриваемом случае нам понадобятся следующие свойства класса `XMLHttpRequest`:

- `readyState` — возвращает число, обозначающее состояние ожидания данных:
  - 0 — соединение с Web-сервером еще не установлено;
  - 1 — соединение с Web-сервером установлено;
  - 2 — данные загружены, но еще не обработаны;
  - 3 — идет обработка загруженных данных;
  - 4 — данные обработаны и могут быть извлечены;
- `status` — возвращает код ответа Web-сервера в виде числа: 200 (данные успешно получены) или 404 (возникла ошибка).

Пример:

```
function getData() {
  if ((oAJAX.readyState == 4) && (oAJAX.status == 200)) {
    // Данные получены. Выполняем их обработку
  }
}
oAJAX.open("GET", "search.php", true);
oAJAX.onreadystatechange = getData;
oAJAX.send();
```

Собственно получение данных

Загруженные данные мы можем извлечь из свойства `responseText` класса `XMLHttpRequest`. Эти данные представляют собой строку, хранящую фрагмент HTML-кода или данные, закодированные в форматах XML или JSON.

В листинге 3.76 приведен полный код, подгружающий фрагмент кода HTML, который хранится в файле /fragments/fragment3.html на сервере, и выводящий его на страницу в контейнере `output`.

Листинг 3.76. Пример загрузки с сервера фрагмента HTML-кода

```
<div id="output"></div>
. . .
function getData() {
```

```
  if ((oAJAX.readyState == 4) && (oAJAX.status == 200)) {
    var oOutput = document.getElementById("output");
    oOutput.innerHTML = oAJAX.responseText;
  }
}

if (window.XMLHttpRequest)
  var oAJAX = new XMLHttpRequest();
} else {
  var oAJAX = new ActiveXObject("Microsoft.XMLHTTP");
}
oAJAX.open("GET", "/fragments/fragment3.html", true);
oAJAX.onreadystatechange = getData;
oAJAX.send();
```

3.20.4. Формат JSON

Ранее мы говорили, что данные, предназначенные для дальнейшей обработки, можно кодировать в формате *JSON* (JavaScript Object Notation, объектная нотация JavaScript). Во многих случаях этот формат намного удобнее, чем XML, вследствие его компактности и простоты обработки.

Описание формата JSON

Данные, закодированные в формате JSON, представляют собой строку с кодом, объявляющим объект класса `Object` *(см. разд. 3.19.1)*. В свойствах этого объекта, собственно, и хранятся отдельные значения, составляющие массив закодированных данных.

Здесь нужно помнить три момента:

1. В данных JSON имена свойств также берутся в кавычки.
2. Значениями свойств не могут быть функции, т. е. в объектах, закодированных в JSON, могут присутствовать лишь свойства, но никак не методы.
3. Данные JSON всегда сохраняются в кодировке UTF-8.

Пример данных JSON:

```
{ "id": "3.20.4", "title": "Формат JSON" }
```

В листинге 3.77 приведен другой пример кодирования в формате JSON — на этот раз целого массива данных.

Листинг 3.77. Пример кодирования данных в формате JSON

```
{
  "status": 1,
  "data": [
    { "id": "3.20.1", "title": "Подготовка к загрузке данных" },
```

```
    { "id": "3.20.2", "title": "Отправка запроса" },
    { "id": "3.20.3", "title": "Получение результата" }
  ]
}
```

Здесь свойство `status` хранит числовой код состояния (`1` обозначает отсутствие ошибок при генерировании данных серверной программой), а свойство `data` — массив, описывающих параграфы какой-либо книги.

Декодирование данных JSON: стандартный способ

В новых Web-обозревателях язык JavaScript поддерживает объект `JSON`. Метод `parse` этого объекта выполняет декодирование данных JSON:

```
JSON.parse(<данные JSON>)
```

Данные JSON передаются в виде строки (собственно, в виде строки мы и получим их из свойства `responseText` класса `XMLHttpRequest`). Метод возвращает сгенерированный на основе этих данных объект класса `Object`, который мы можем без проблем обработать в программе.

Пример:

```
var oData = JSON.parse(oAJAX.responseText);
```

Декодирование данных JSON: способ, применяемый в устаревших Web-обозревателях

Старые программы Web-обозревателей, в частности, Internet Explorer 7 и более ранние его версии, не поддерживают объект `JSON`. Поэтому в них для декодирования данных JSON нам придется применять функцию `eval`.

Пример:

```
var oData = eval(oAJAX.responseText);
```

Декодирование данных JSON: универсальный способ

Чтобы скрипты, загружающие и обрабатывающие данные JSON, работали во всех Web-обозревателях, мы применим универсальный способ, написав следующий код:

```
if (JSON) {
  var oData = JSON.parse(oAJAX.responseText);
} else {
  var oData = eval(oAJAX.responseText);
}
```

В листинге 3.78 приведен код, который загружает данные, взятые из листинга 3.77 и сохраненные в файле data.json, и выводит содержащийся в них список параграфов в контейнер `output` в виде набора абзацев. Каждый абзац содержит порядковый номер и заголовок параграфа.

Листинг 3.78. Пример загрузки и вывода данных

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>AJAX и JSON</title>
    <script>
      window.addEventListener("load", function() {
        var oAJAX = new XMLHttpRequest();
        oAJAX.open("GET", "data.json", true);
        oAJAX.onreadystatechange = function() {
          var oOutput = document.getElementById("output");
          if ((oAJAX.readyState == 4) && (oAJAX.status == 200)) {
            if (JSON) {
              var oData = JSON.parse(oAJAX.responseText);
            } else {
              var oData = eval(oAJAX.responseText);
            }
            if (oData.status == 1) {
              var oP;
              for (i in oData.data) {
                oP = document.createElement("p");
                oP.textContent = oData.data[i].id + ". " +
                oData.data[i].title;
                oOutput.appendChild(oP);
              }
            } else {
              oOutput.textContent = "Получены ошибочные данные.";
            }
          }
        }
        oAJAX.send();
      }, false);
    </script>
  </head>
  <body>
    <div id="output"></div>
  </body>
</html>
```

Отметим, что весь код мы оформили в виде обработчика события `onload` окна Web-обозревателя. Тем самым мы гарантировали, что он будет выполнен лишь после окончания загрузки страницы.

3.21. DOM 3

Объектная модель документа DOM совершенствуется вместе с языком HTML. В настоящее время для HTML 5 разрабатывается очередная ее версия, носящая номер 3. Как и в случае с HTML 5 и CSS 3, DOM 3 существует лишь в виде рабочего стандарта, но уже в той или иной степени поддерживается всеми Web-обозревателями, обладающими поддержкой HTML 5.

3.21.1. Обращение к элементам страницы

DOM 3 предлагает расширенные средства для получения доступа к элементам страницы.

Обращение по имени класса

Мы можем получить доступ ко всем элементам, к которым был привязан стиль с указанным классом. Для этого предназначен метод `getElementsByClassName`, поддерживаемый объектом `document` и всеми элементами страницы:

```
<страница или элемент>.getElementsByClassName(<имя класса>)
```

Имя класса указывается в виде строки без символа точки. В качестве результата возвращается ссылка на коллекцию найденных элементов.

Примеры:

```
<p>Первый абзац.</p>
<p class="paragraph">Второй абзац.</p>
. . .
var cPars = document.getElementsByClassName("paragraph");
```

Коллекция `cPars` будет содержать лишь второй абзац, поскольку только к нему привязан стилевой класс `paragraph`.

```
<div id="cont">
  <p>Первый абзац.</p>
  <p class="paragraph">Второй абзац.</p>
</div>
. . .
var oCont = document.getElementById("cont");
var cPars = oCont.getElementsByClassName("paragraph");
```

А здесь мы сначала обращаемся к контейнеру `cont`, а уже в нем ищем все элементы с привязанным классом `paragraph`.

Обращение по селектору

Исключительно полезная возможность — обращение к элементам страницы по заданному селектору. Для этого применяются два метода, поддерживаемые объектом `document` и всеми элементами страницы.

Метод querySelector возвращает первый из подходящих элементов или null в случае неудачного поиска:

```
<страница или элемент>.querySelector(<селектор>)
```

Селектор задается в виде строки.

Рассмотрим пару примеров:

```
var oEl = document.querySelector("div#nav ul");
```

Получаем список, находящийся в контейнере с идентификатором nav.

```
var oEl = document.querySelector("#main h1+p");
```

Получаем первый абзац, следующий за первым заголовком первого уровня, который находится в элементе с идентификатором main.

Метод querySelectorAll возвращает коллекцию со всеми элементами, совпадающими с заданным селектором:

```
<страница или элемент>.querySelectorAll(<селектор>)
```

Пример:

```
var cEls = document.querySelectorAll("#main h1+p");
```

Получаем все абзацы, непосредственно следующие за заголовками первого уровня и находящиеся в элементе с идентификатором main.

3.21.2. Управление содержимым страницы

Изменения, предлагаемые DOM 3 для управления содержимым страницы, направлены на стандартизацию соответствующих инструментов и упрощение их использования.

Создание, изменение и удаление элементов страницы

Отныне создавать, изменять и удалять любые элементы страницы следует исключительно с помощью методов createElement, createTextNode, appendChild, insertBefore, removeChild, replaceChild, cloneNode, write и writeln. Специфические свойства и методы, предназначенные для работы с содержимым таблиц, больше не поддерживаются.

Работа с содержимым элементов

Для указания содержимого элемента следует использовать свойство innerHTML, Свойство outerHTML и метод insertAdjacentHTML не поддерживаются.

Для задания текстового содержимого элемента нужно применять свойство textContent. Свойства innerText, outerText и методы getAdjacentText, insertAdjacentText и replaceAdjacentText не поддерживаются.

Работа со стилями элементов

Для работы со стилями, привязанными к элементам страниц, следует использовать исключительно объект `style`, доступный через одноименное свойство. Свойство `currentStyle` не поддерживается, и вместо него рекомендуется применять метод `getComputedStyle`.

В самом объекте `style` более не поддерживаются свойства `cssText`, `pixelBottom`, `pixelHeight`, `pixelTop`, `pixelWidth`, `posBottom`, `posHeight`, `posTop` и `posWidth`. Их следует заменять свойствами `bottom`, `height`, `top` и `width`, преобразуя их значения в числовой вид с помощью функции `parseInt`.

3.21.3. Обработка событий

Для обработки событий DOM 3 также предлагает бóльшую кроссплатформенность и более простые инструменты.

Указание обработчиков событий

Для указания обработчиков событий теперь следует использовать:

- либо метод `addEventListener`;
- либо параметры тегов (`onclick`, `onmousedown` и др.).

Все прочие способы указания обработчиков, описанные в этой главе ранее, не поддерживаются.

Новый набор событий

Набор событий, поддерживаемый элементами страницы и самой страницей, несколько изменился: добавилась поддержка события `onmousewheel`, периодически возникающего при вращении колесика мыши.

Новый объект *event*

DOM 3 предлагает новый объект `event`, хранящий сведения о возникшем событии. Набор поддерживаемых им свойств несколько изменен по сравнению с описанным ранее в этой книге.

Теперь объект `event` поддерживает следующие новые свойства:

- `bubbles` — `true`, если текущее событие может всплывать, и `false` — в противном случае.
- `cancellable` — `true`, если действие по умолчанию для текущего события может быть отменено, и `false` — в противном случае.
- `defaultPrevented` — `true`, если действие по умолчанию для текущего события было отменено в этом или предыдущем обработчике, и `false` — в противном случае.
- `detail` — для событий `onclick`, `onmousedown` и `onmouseup` возвращает количество выполненных щелчков мышью. Для события `onmousewheel` возвращает величину,

на которую было повернуто колесико мыши, умноженную на 3; положительное значение обозначает прокрутку вниз, отрицательное — вверх.

- `eventPhase` — обозначение фазы события:
  - 1 — событие было перехвачено у вложенного элемента;
  - 2 — событие в данный момент обрабатывается в том же элементе, в котором и возникло;
  - 3 — событие "всплыло" из вложенного элемента.
- `metaKey` — `true`, если была нажата клавиша <Meta>, и `false`, если эта клавиша не была нажата.
- `pageX` — горизонтальная координата курсора мыши относительно верхнего левого угла страницы в пикселах.
- `pageY` — вертикальная координата курсора мыши относительно верхнего левого угла страницы в пикселах.
- `relatedTarget` — для события `onmouseover` возвращает ссылку на элемент, с которого был уведен курсор мыши. Для события `onmouseout` возвращает ссылку на элемент, на который был наведен курсор мыши.
- `timeStamp` — время возникновения события в миллисекундах.

Также новый объект `event` поддерживает методы `preventDefault` и `stopPropagation`.

Свойства `altLeft`, `cancelBubble`, `ctrlLeft`, `fromElement`, `offsetX`, `offsetY`, `propertyName`, `repeat`, `returnValue`, `shiftLeft`, `srcElement`, `toElement`, `x` и `y` не поддерживаются.

3.21.4. Работа с формами и элементами управления

Для работы с формами и элементами управления DOM 3 предлагает весьма обширный набор новых возможностей.

Работа с формами

Новых инструментов для работы с самими формами не очень много. Прежде всего, это два новых свойства:

- `autocomplete` — признак, указывающий, работает ли в форме автодополнение. Если в качестве признака задана строка `"on"`, автодополнение работает, если `"off"` — не работает.
- `noValidate` — признак того, будет ли форма проверять введенные в нее данные на корректность. Значение `true` активизирует проверку данных, значение `false` отключает ее.

Также нас может заинтересовать метод `checkValidity`. Он возвращает `true`, если во все элементы управления формы занесены корректные данные, и `false` — в противном случае.

Пример:

```
<form id="frm">
  . . .
</form>
. . .
var frm = document.getElementById("frm");
if (!frm.checkValidity()) {
  window.alert("Введите корректные данные!");
}
```

Работа с элементами управления

Элементы управления поддерживают следующие вновь введенные свойства:

- `autocomplete` — аналогично одноименному свойству формы.
- `autofocus` — признак того, должен ли этот элемент управления получить фокус ввода сразу после загрузки страницы. Значение `true` активизирует автоматическое получение фокуса ввода, значение `false` деактивирует его.
- `max` — максимальное значение, которое можно указать в поле ввода числа или регуляторе, в виде числа.
- `min` — минимальное значение, которое можно указать в поле ввода числа или регуляторе, в виде числа.
- `multiple` — признак, позволяет ли поле ввода файла или список выбирать сразу несколько позиций: `true` — позволяет, `false` — не позволяет.
- `pattern` — регулярное выражение, определяющее формат заносимого в поле ввода значения и записанное в виде строки.
- `placeholder` — текст подсказки, выводимой прямо в поле ввода, в виде строки.
- `required` — признак того, является ли элемент управления обязательным для ввода: `true` — является, `false` — не является.
- `selectedOptions` — возвращает коллекцию выбранных в списке пунктов.
- `selectionStart` — номер первого символа выделенного в поле ввода или области редактирования фрагмента текста (в виде числа). Если текст не выделен, возвращает номер символа, на котором стоит текстовый курсор.
- `selectionEnd` — номер последнего символа выделенного фрагмента текста (в виде числа). Если текст не выделен, возвращает номер символа, на котором стоит текстовый курсор.
- `step` — интервал между значениями, допустимыми для занесения в поле ввода числа или выбора с помощью регулятора (в виде числа).
- `type` — тип элемента управления, фактически — значение параметра `type` тега `<input>`, в виде строки. Для области редактирования возвращает строку `"textarea"`, для списков с возможностью выбора одного пункта — `"select-one"`, для списков с возможностью выбора нескольких пунктов — `"select-multiple"`.

- `validationMessage` — возвращает текст сообщения (в виде строки) о некорректно занесенном в поле ввода значении.
- `validity` — возвращает сведения об ошибках, допущенных посетителем при занесении значения в поле ввода. Представляет собой объект класса `ValidityState`, поддерживающего следующие доступные только для чтения свойства:
  - `badInput` — `true`, если введенное посетителем значение неполное;
  - `patternMismatch` — `true`, если введенное значение не совпадает с заданным шаблоном (регулярным выражением, занесенным в свойство `pattern`);
  - `rangeOverflow` — `true`, если введенное число больше указанного максимального значения;
  - `rangeUnderflow` — `true`, если введенное число меньше указанного минимального значения;
  - `stepMismatch` — `true`, если введенное число не укладывается в заданный интервал;
  - `tooLong` — `true`, если введенная строка слишком длинная;
  - `typeMismatch` — `true`, если введенное значение не соответствует требуемому типу (т. е. не является числом, интернет-адресом, адресом электронной почты и т. п.);
  - `valueMissing` — `true`, если не указано обязательное для ввода значение;
  - `customError` — `true`, если было задано иное сообщение об ошибке ввода данных (как это сделать, будет описано далее);
  - `valid` — `true`, если введенное значение полностью корректно (и все перечисленные ранее свойства хранят значение `false`).
- `willValidate` — возвращает `true`, если значение, присутствующее в этом элементе управления, будет проверяться на корректность. Для элементов, недоступных для посетителя и доступных лишь для чтения, возвращается `false`.

Еще элементы управления поддерживают два новых события:

- `oninput` — периодически возникает в процессе ввода данных в поле ввода или в область редактирования;
- `oninvalid` — возникает, если значение, занесенное в элемент управления, некорректно.

Обработка списков с возможностью выбора нескольких пунктов

Потребность в списках с возможностью одновременного выбора нескольких нунктов возникает нечасто. Однако нам нужно знать, как получить все выбранные в таком списке нункты в скрипте.

Листинг 3.79 содержит код, выполняющий эту задачу. В нем мы использовали свойство `selectedOptions` списка, введенное стандартом DOM 3.

Листинг 3.79. Пример выбора нескольких пунктов из списка

```
<select id="selTechs" multiple>
  <option value="html">HTML 5</option>
  <option value="css">CSS 3</option>
  <option value="javascript">JavaScript</option>
  <option value="dom">DOM 3</option>
  <option value="php">PHP</option>
  <option value="mysql">MySQL</option>
</select>
. . .
var selTechs = document.getElementById("selTechs");
var cSIs = selTechs.selectedOptions;
var aSIs = [];
for (i in cSIs) {
  aSIs.push(cSIs[i].value);
}
```

В результате выполнения этого кода в массиве `aSIs` окажется набор значений выбранных в списке нунктов.

Расширенная проверка значения, занесенного в поле ввода

Мы уже знаем, что DOM 3 оснастила поля ввода богатым набором инструментов для проверки занесенных в них данных на корректность. Теперь, если мы хотим удостовериться, что посетитель ввел в поле, скажем, число, а не что-то другое, мы создадим поле ввода числа и укажем для него параметры, которым должно удовлетворять это число, — об остальном позаботится сам Web-обозреватель.

Web-обозреватель сам выводит на экран сообщения об ошибках ввода данных. Однако мы можем предусмотреть свои собственные сообщения. Пример выводящего их кода иллюстрирует листинг 3.80.

Листинг 3.80. Пример вывода сообщений об ошибке

```
<input type="number" id="txtYear" min="2000" max="3000" step="10">
<div id="output"></div>
. . .
var txtYear = document.getElementById("txtYear");
var oOutput = document.getElementById("output");
txtYear.addEventListener("invalid", function() {
  if (txtYear.validity.rangeUnderflow)
    oOutput.textContent = "Значение года не должно быть меньше 2000."
  if (txtYear.validity.rangeOverflow)
    oOutput.textContent = "Значение года не должно быть меньше 3000."
  if (txtYear.validity.stepMismatch)
    oOutput.textContent = "Значение года должно быть кратно 10."
```

```
  if (txtYear.validity.valid)
    oOutput.textContent = ""
}, false);
```

Здесь мы обрабатываем предусмотренное стандартом DOM 3 событие `oninvalid`, возникающее в полях ввода при занесении в них некорректных данных. В обработчике этого события мы получаем объект класса `ValidityState`, хранящий сведения об ошибке ввода, и выводим в контейнере `output` соответствующее сообщение. Если посетитель ввел корректное значение, мы очищаем этот контейнер.

Если мы хотим вывести свое собственное сообщение об ошибке прямо под элементом управления, в который было введено некорректное значение, как это делает сам Web-обозреватель, то зададим это сообщение с помощью метода `setCustomValidity` нужного элемента управления:

<элемент управления>.setCustomValidity(*<сообщение об ошибке ввода>*)

Сообщение об ошибке ввода указывается в виде строки. Результат этот метод не возвращает.

Как только мы зададим для элемента управления свое сообщение об ошибке, вызвав метод `setCustomValidity`, занесенное в этот элемент значение будет считаться некорректным. Web-обозреватель выведет заданное нами сообщение на экран, а данные отправлены не будут.

Чтобы пометить введенное в элемент управления значение как корректное, достаточно вызвать этот метод, передав ему в качестве параметра пустую строку.

Пример кода, выводящего сообщение об ошибке ввода подобным образом, показан в листинге 3.81.

Листинг 3.81. Пример вывода сообщения об ошибке методом `setCustomValidity`

```
<input type="number" id="txtYear">
. . .
var txtYear = document.getElementById("txtYear");
txtYear.addEventListener("input", function() {
  var n = parseInt(txtYear.value);
  if (!isNaN(n)) {
    if (n < 2000) {
      txtYear.setCustomValidity("Значение года не должно быть ↳
      меньше 2000.");
    }
  }
}, false);
```

3.21.5. Работа с графическими изображениями

Для управления графическими изображениями DOM 3 предоставляет нам три дополнительных свойства, поддерживаемые объектом-изображением:

- `complete` — возвращает `true`, если изображение было полностью загружено, и `false` — в противном случае;
- `naturalHeight` — возвращает высоту изображения, загруженного из указанного файла (в пикселах в виде числа);
- `naturalWidth` — возвращает ширину изображения, загруженного из указанного файла (в пикселах в виде числа).

Пример:

```
<img id="image" src="/images/picture.jpg">
. . .
var oI = document.getElementById("image");
if (oI.parentNode.clientWidth < oI.naturalWidth)
  oI.style.width = oI.parentNode.clientWidth + "px";
```

Если ширина изображения превышает ширину контейнера, устанавливаем первую равной последней.

3.21.6. Работа с мультимедиа

Элементы аудио- и видеороликов, помещенных на страницу средствами HTML 5, также поддерживают расширенный набор программных инструментов, который довольно велик.

Свойства, методы и события аудио- и видеороликов

Начнем, как обычно, с перечисления дополнительных свойств.

- `autoplay` — признак, запустится ли воспроизведение ролика сразу после его загрузки: `true` — запустится, `false` — не запустится.
- `controls` — признак, будут ли на экране присутствовать элементы управления воспроизведением ролика: `true` — будут присутствовать, `false` — не будут присутствовать.
- `currentSrc` — возвращает интернет-адрес воспроизводящегося в данный момент ролика в виде строки. Может отличаться от интернет-адреса, заданного в самом теге, если ролик предоставляется серверным приложением, автоматически выбирающим мультимедийный файл в зависимости от параметров клиентского компьютера.
- `currentTime` — текущая позиция воспроизведения ролика в виде числа в секундах.
- `duration` — возвращает продолжительность ролика в виде числа в секундах.
- `ended` — возвращает `true`, если воспроизведение ролика закончилось, и `false` — в противном случае.
- `height` — высота воспроизводящегося на странице ролика (в виде числа в пикселах).

- `loop` — признак, будет ли ролик воспроизводиться бесконечно: `true` — ролик воспроизводится бесконечно, `false` — ролик будет воспроизведен всего один раз.
- `muted` — признак, приглушен ли звук в данный момент: `true` — приглушен, `false` — не приглушен.
- `networkState` — возвращает текущее состояние загрузки ролика (в виде числа):
  - 0 — загрузка еще не началась;
  - 1 — ролик уже загружен;
  - 2 — идет загрузка ролика;
  - 3 — файл с роликом отсутствует на сервере.
- `paused` — признак, приостановлено ли воспроизведение ролика в данный момент: `true` — приостановлено, `false` — не приостановлено.
- `playbackRate` — текущая скорость воспроизведения ролика. Значение должно представлять собой число с плавающей точкой, которое будет умножено на значение скорости воспроизведения, полученной из файла с роликом; положительные значения задают воспроизведение в прямом порядке, отрицательные — в обратном.

  Пример:

```
<video id="vid" src="film.mp4"></video>
. . .
var oVid = document.getElementById("vid");
oVid.playbackRate = 2;
```

  Увеличиваем скорость воспроизведения ролика вдвое.
- `poster` — интернет-адрес файла заставки в виде строки.
- `preload` — признак, следует ли выполнять предварительную загрузку ролика. Значением этого свойства должна быть одна из строк:
  - `"none"` — не выполнять предзагрузку;
  - `"metadata"` — загрузить только самое начало файла, где хранятся сведения о ролике;
  - `"auto"` — загрузить весь файл; значение по умолчанию.
- `readyState` — возвращает состояние загрузки ролика (в виде числа):
  - 0 — загрузка еще не началась;
  - 1 — загружен лишь заголовок файла, хранящий сведения о самом ролике;
  - 2 — загружены данные, достаточные для воспроизведения текущего кадра, но для воспроизведения следующих кадров данных недостаточно;
  - 3 — загружены данные, достаточные для воспроизведения текущего и следующих кадров;

- 4 — загружены данные, достаточные для бесперебойного воспроизведения всего ролика.

- `seeking` — возвращает `true`, если в данный момент посетитель меняет позицию воспроизведения ролика, и `false` — в противном случае.
- `src` — интернет-адрес воспроизводящегося в данный момент ролика (в виде строки).
- `videoHeight` — возвращает высоту видеоролика, загруженную из файла (в виде числа в пикселах).
- `videoWidth` — возвращает ширину видеоролика, загруженную из файла (в виде числа в пикселах).
- `volume` — громкость звука. Значение должно представлять собой число с плавающей точкой от 0 (звук отсутствует) до 1 (максимальная громкость).
- `width` — ширина воспроизводящегося на странице ролика (в виде числа в пикселах).

Примеры:

```
<video id="vid" src="film.mp4"></video>
. . .
var oVid = document.getElementById("vid");
oVid.loop = true;
oVid.volume = 0.5;
```

Делаем ролик воспроизводящимся бесконечно и задаем для него половинную громкость.

```
oVid.width = oVid.naturalWidth / 2;
oVid.height = oVid.naturalHeight / 2;
```

Уменьшаем размеры воспроизводящегося на экране видеоролика вдвое.

Закончив с многочисленными свойствами, займемся дополнительными методами, поддерживаемыми аудио- и видеороликами.

Метод `canPlayType` позволяет узнать, поддерживает ли Web-обозреватель воспроизведение роликов указанного формата:

<ролик>`.canPlayType(`*<MIME-тип>*`)`

MIME-тип формата ролика (его можно найти в табл. 1.1) указывается в виде строки. Метод возвращает одно из следующих строковых значений:

- `"probably"` — скорее всего, поддерживает;
- `"maybe"` — возможно, поддерживает;
- `""` (пустая строка) — гарантированно не поддерживает.

Пример:

```
if (oVid.canPlayType("video/webm") != "") {
  oVid.src = "film.webm";
```

```
} else {
  oVid.src = "film.mp4";
}
```

Если есть возможность воспроизвести ролик в формате WebM, загружаем ролик именно в таком формате, иначе выполняем загрузку ролика в формате MP4.

Вот еще три метода, не принимающие параметров и не возвращающие результат:

- `load` — выполняет повторную загрузку ролика;
- `pause` — приостанавливает воспроизведение ролика;
- `play` — запускает или возобновляет воспроизведение ролика.

Что касается специфических для аудио- и видеороликов событий, то их список также весьма велик.

- `onabort` — возникает, когда посетитель прерывает загрузку файла с роликом.
- `oncanplay` — возникает, если Web-обозреватель получил достаточно данных, чтобы, по крайней мере, начать воспроизведение ролика, однако в будущем возможны приостановки воспроизведения для подгрузки данных.
- `oncanplaythrough` — возникает, если Web-обозреватель получил достаточно данных, чтобы начать воспроизведение ролика без приостановок для подгрузки данных.
- `ondurationchange` — возникает, когда Web-обозреватель получает значение продолжительности ролика, или когда это значение почему-то изменяется.
- `onemptied` — возникает при выгрузке файла ролика, что может произойти, например, при вызове метода `load`.
- `onended` — возникает, если воспроизведение ролика закончилось.
- `onerror` — возникает при прерывании загрузки файла с роликом в результате ошибки скрипта или сетевого сбоя.
- `onloadeddata` — возникает, если Web-обозреватель загрузил достаточно данных для вывода на экран первого кадра, но не для начала воспроизведения ролика.
- `onloadedmetadata` — возникает, когда Web-обозреватель получает величины размеров "картинки" ролика.
- `onloadstart` — возникает, когда Web-обозреватель только начинает загрузку файла с роликом.
- `onpause` — возникает, когда воспроизведение ролика приостанавливается.
- `onplay` — возникает после начала или возобновления воспроизведения ролика.
- `onplaying` — возникает, когда Web-обозреватель получает достаточно данных, чтобы возобновить воспроизведение приостановленного ролика.
- `onprogress` — периодически возникает в процессе загрузки Web-обозревателем мультимедийного файла.
- `onratechange` — возникает при изменении значения скорости воспроизведения ролика.

- `onseeked` — возникает по завершении изменения текущей позиции воспроизведения ролика.
- `onseeking` — возникает после начала изменения текущей позиции воспроизведения ролика.
- `onstalled` — возникает через три секунды после остановки процесса подгрузки очередной порции данных из мультимедийного файла. Обычно означает, что дальнейшая подгрузка файла с роликом невозможна.
- `ontimeupdate` — возникает при изменении текущей позиции воспроизведения.
- `onvolumechange` — возникает при изменении значения громкости, приглушении звука или восстановления его после приглушения.
- `onwaiting` — возникает, когда воспроизведение ролика приостанавливается для подгрузки очередной порции данных.

Для события `onprogress` объект `event`, хранящий сведения о событии, предоставляет три дополнительных свойства:

- `lengthComputable` — возвращает `true`, если Web-обозреватель может определить размер загружаемого мультимедийного файла, и `false` — в противном случае;
- `loaded` — возвращает размер уже загруженной части файла в виде числа в байтах;
- `total` — возвращает полный размер загружаемого файла в виде числа в байтах.

Создание элементов для управления воспроизведением ролика

Разумеется, сам Web-обозреватель предоставляет элементы для управления воспроизведением ролика, и их очень просто вывести на экран. Но иногда приходится предусматривать собственные элементы подобного рода. Код из листинга 3.82 показывает, как это можно сделать.

Листинг 3.82. Пример создания элементов управления воспроизведением видеоролика

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Видео</title>
    <script>
      window.addEventListener("load", function() {
        var oVid = document.getElementById("vid");
        var btnStart = document.getElementById("btnStart");
        var btnPause = document.getElementById("btnPause");
        var btnStop = document.getElementById("btnStop");
        var ranPosition = document.getElementById("ranPosition");
```

```
  var ranVolume = document.getElementById("ranVolume");
  var btnMute = document.getElementById("btnMute");
  var btnUnmute = document.getElementById("btnUnmute");
  oVid.addEventListener("canplaythrough", function() {
    btnStart.disabled = false;
    btnStop.disabled = false;
    ranPosition.disabled = false;
    ranPosition.value = 0;
    ranPosition.max = oVid.duration;
  }, false);
  oVid.addEventListener("timeupdate", function() {
    ranPosition.value = oVid.currentTime;
  }, false);
  btnStart.addEventListener("click", function() {
    oVid.play();
    btnStart.disabled = true;
    btnPause.disabled = false;
  }, false);
  btnPause.addEventListener("click", function() {
    oVid.pause();
    btnStart.disabled = false;
    btnPause.disabled = true;
  }, false);
  btnStop.addEventListener("click", function() {
    oVid.pause();
    ranPosition.value = 0;
    oVid.currentTime = 0;
    btnStart.disabled = false;
    btnPause.disabled = true;
  }, false);
  ranPosition.addEventListener("change", function() {
    oVid.currentTime = ranPosition.value;
  }, false);
  ranVolume.addEventListener("change", function() {
    oVid.volume = ranVolume.value / 10;
  }, false);
  btnMute.addEventListener("click", function() {
    oVid.muted = true;
    btnMute.disabled = true;
    btnUnmute.disabled = false;
  }, false);
  btnUnmute.addEventListener("click", function() {
    oVid.muted = false;
    btnMute.disabled = false;
    btnUnmute.disabled = true;
  }, false);
}, false);
```

```
    </script>
  </head>
  <body>
    <video id="vid" src="film.mp4"></video>
    <div>
      <input type="button" id="btnStart" value="Старт" disabled>
      <input type="button" id="btnPause" value="Пауза" disabled>
      <input type="button" id="btnStop" value="Стоп" disabled>
    </div>
    <div>
      <input type="range" id="ranPosition" min="0" disabled>
    </div>
    <div>
      <input type="range" id="ranVolume" min="0" max="10">
      <input type="button" id="btnMute" value="Тихо">
      <input type="button" id="btnUnmute" value="Громко" disabled>
    </div>
  </body>
</html>
```

3.21.7. Канва HTML 5. Программируемая графика

Еще в *главе 1* упоминалось, что язык HTML 5 предлагает довольно богатые возможности по выводу на страницу произвольной графики, рисуемой программно, в скрипте. Настала пора узнать, как это делается.

Канва

Для вывода программируемой графики применяется особый элемент страницы, называемой *канвой*. В элементах прочих типов программное рисование не поддерживается.

Канва создается с помощью парного тега `<canvas>`. Необязательные параметры `width` и `height` задают, соответственно, ширину и высоту канвы в пикселах; если они не указаны, размеры канвы составят 300×150 пикселов.

Пример:

```
<canvas id="cnv" width="400" height="300"></canvas>
```

Создаем канву размерами 400×300 пикселов.

Контекст рисования

Создав на странице саму канву, мы можем получить ее *контекст рисования* — набор инструментов, используемый для рисования на ней. Он возвращается методом `getContext`, поддерживаемым канвой; в качестве значения единственного параметра этого метода указывается строка `"2d"`.

Пример:

```
var oCanvas = document.getElementById("cnv");
var ctxCanvas = oCanvas.getContext("2d");
```

Получаем контекст рисования созданной ранее канвы `cnv`.

Контекст рисования представляется объектом класса `CanvasRenderingContext2D`. Все операции по рисованию выполняются с применением его свойств и методов, которые мы далее рассмотрим.

Прямоугольники

Для рисования прямоугольника без заливки (т. е. одного лишь контура прямоугольника) предназначен метод `strokeRect`:

<контекст рисования>`.strokeRect(`*<горизонтальная координата>*, *<вертикальная координата>*, *<ширина>*, *<высота>*`)`

Все параметры указываются в виде чисел в пикселах. Результат этот метод нет возвращает.

Координаты отсчитываются от левого верхнего угла канвы. Горизонтальная координата отсчитывается по направлению вправо, вертикальная — вниз.

Пример:

```
ctxCanvas.strokeRect(20, 20, 360, 260);
```

Аналогичный метод `fillRect` рисует прямоугольник с заливкой.

Пример:

```
ctxCanvas.fillRect(40, 40, 320, 220);
```

Метод `clearRect` очищает заданпую прямоугольпую область от любой присутствовавшей там графики. Вызывается он так же, как методы `strokeRect` и `fillRect`.

Примеры:

```
ctxCanvas.fillRect(0, 0, 400, 300);
ctxCanvas.clearRect(100, 100, 200, 100);
```

Рисуем большой прямоугольник с заливкой, занимающий всю канву, после чего создаем в его середине прямоугольную "прореху".

```
ctxCanvas.clearRect(0, 0, 400, 300);
```

Очищаем всю канву от всей присутствующей на ней графики.

Задание цвета, уровня прозрачности и толщины линий

Цвет контура мы задаем с помощью свойства `strokeStyle`. Все фигуры, которые мы впоследствии нарисуем, будут иметь контур указанного цвета. Сам цвет может быть указан любым способом, поддерживаемым CSS.

Примеры указания цвета:

```
ctxCanvas.strokeStyle = "red";
ctxCanvas.strokeStyle = "#ff0000";
ctxCanvas.strokeStyle = "rgb(255, 0, 0)";
```

Все эти выражения задают для контура красный непрозрачный цвет.

```
ctxCanvas.strokeStyle = "rgba(0, 127, 0, 0.5)";
```

Задаем для контура темно-зеленый полупрозрачный цвет.

Изначально, сразу после загрузки и вывода канвы на страницу, линии контура будут иметь черный непрозрачный цвет.

Свойство `fillStyle` определяет цвет заливки, который также может быть задан любым поддерживаемым CSS способом. По умолчанию цвет заливок черный и непрозрачный.

Примеры:

```
ctxCanvas.fillStyle = "rgb(0, 0, 255)";
```

Задаем для заливки синий цвет.

```
ctxCanvas.strokeStyle = "rgba(255, 0, 0, 1)";
ctxCanvas.fillStyle = "rgba(255, 0, 0, 1)";
ctxCanvas.fillRect(0, 100, 400, 100);
ctxCanvas.strokeStyle = "rgba(0, 255, 0, 0.5)";
ctxCanvas.fillStyle = "rgba(0, 255, 0, 0.5)";
ctxCanvas.fillRect(100, 0, 200, 300);
```

Рисуем прямоугольник, используя и для контура, и для заливки непрозрачный красный цвет, после чего поверх него рисуем прямоугольник с заливкой, но уже полупрозрачным зеленым цветом.

> ***Внимание!***
>
> Нельзя присваивать значение свойства `strokeStyle` свойству `fillStyle` и наоборот. Это вызовет ошибку выполнения программы.

Свойство `lineWidth` задает толщину линий контура (в пикселах в виде числа).

Пример:

```
ctxCanvas.lineWidth = 20;
```

Задаем толщину линий равной 20 пикселов.

Свойство `globalAlpha` указывает уровень прозрачности для любой графики, которую мы впоследствии нарисуем. Уровень прозрачности также задается в виде числа от 0.0 (полностью прозрачный) до 1.0 (полностью непрозрачный; значение по умолчанию).

Пример:

```
ctxCanvas.globalAlpha = 0.1;
```

Рисование сложных фигур

Канва может выводить не только прямоугольники. Мы имеем возможность нарисовать фигуру практически любой сложности.

Как рисуются сложные фигуры

Рисование контура такой фигуры начинается с вызова метода `beginPath`. Этот метод не принимает параметров и не возвращает результат.

Теперь мы можем начать собственно рисование контура сложной фигуры. Как это делается, мы скоро узнаем.

После окончания рисования сложного контура мы можем захотеть, чтобы Web-обозреватель сделал его замкнутым. Для этого мы вызовем метод `closePath`, который также не принимает параметров и не возвращает результат. После его вызова последняя точка контура будет соединена с самой первой, в которой началось его рисование.

Завершает рисование контура вызов одного из двух методов: `stroke` или `fill`. Первый метод просто завершает рисование контура, второй, помимо этого, замыкает контур, если он не замкнут, и рисует заливку получившейся фигуры. Оба этих метода не принимают параметров и не возвращают результат.

А теперь рассмотрим методы, которые предназначены для рисования разнообразных линий, составляющих сложный контур.

Перо. Перемещение пера

При рисовании сложного контура используется концепция *пера* — воображаемого инструмента рисования. Перо можно перемещать в любую точку на канве. Рисование каждой линии контура начинается в точке, где в данный момент находится перо. После рисования каждой линии перо перемещается в ее конечную точку, из которой тут же можно начать рисование следующей линии контура.

Изначально, сразу после загрузки Web-страницы, перо находится в точке с координатами [0,0], т. е. в верхнем левом углу канвы. Переместить перо в точку канвы, где мы собираемся начать рисование контура, позволяет метод `moveTo`:

```
<контекст рисования>.moveTo(<горизонтальная координата>,
<вертикальная координата>)
```

Параметры указываются в пикселах в виде чисел. Метод `moveTo` не возвращает результат.

Пример:

```
ctxCanvas.moveTo(200, 150);
```

Это выражение перемещает перо в центр нашей канвы — в точку с координатами [200,150].

Прямые линии

Прямые линии рисовать проще всего. Для этого служит метод `lineTo`:

<контекст рисования>`.lineTo(`*<горизонтальная координата>*`,`
<вертикальная координата>`)`

Параметры задаются в виде чисел в пикселах. Метод `lineTo` не возвращает результат.

Пример:

```
ctxCanvas.strokeStyle = "red";
ctxCanvas.beginPath();
ctxCanvas.moveTo(20, 20);
ctxCanvas.lineTo(380, 20);
ctxCanvas.lineTo(200, 280);
ctxCanvas.closePath();
ctxCanvas.stroke();
```

Рисуем красный треугольник без заливки.

Дуги

Дуги рисуются тоже довольно просто. Для этого предусмотрен метод `arc`:

<контекст рисования>`.arc(`*<горизонтальная координата>*`,`
<вертикальная координата>`,` *<радиус>*`,` *<начальный угол>*`,` *<конечный угол>*`,`
`true|false)`

Первые три параметра указываются в пикселах, четвертый и пятый — в радианах; отметим, что углы отсчитываются от горизонтальной оси. Если шестой параметр имеет значение `true`, то дуга рисуется против часовой стрелки, а если `false` — по часовой стрелке. Метод `arc` не возвращает результат.

Тот факт, что углы задаются в радианах, несколько осложняет работу. Нам придется пересчитывать величины углов из градусов в радианы с помощью следующего выражения:

```
radians = (Math.PI / 180) * degrees;
```

Здесь переменная `degrees` хранит значение угла в градусах, а переменная `radians` будет хранить то же значение, но в радианах.

Пример:

```
ctxCanvas.strokeStyle = "green";
ctxCanvas.fillStyle = "blue";
ctxCanvas.beginPath();
ctxCanvas.arc(200, 150, 100, 0, Math.PI * 2, false);
ctxCanvas.stroke();
```

Здесь мы рисуем круг с зеленым контуром и синей заливкой.

Кривые Безье

Для рисования кривых Безье с двумя контрольными точками используется метод bezierCurveTo:

```
<контекст рисования>.bezierCurveTo
↳(<горизонтальная координата первой контрольной точки>,
<вертикальная координата первой контрольной точки>,
<горизонтальная координата второй контрольной точки>,
<вертикальная координата второй контрольной точки>,
<горизонтальная координата конечной точки>,
<вертикальная координата конечной точки>)
```

Назначение параметров этого метода понятно из их описания. Все они задаются в пикселах в виде чисел. Метод не возвращает результат.

Пример:

```
ctxCanvas.beginPath();
ctxCanvas.moveTo(100, 100);
ctxCanvas.bezierCurveTo(120, 80, 160, 20, 100, 200);
ctxCanvas.stroke();
```

Метод quadraticCurveTo (не возвращающий результат) рисует кривые Безье с одной контрольной точкой:

```
<контекст рисования>.quadraticCurveTo
↳(<горизонтальная координата контрольной точки>,
<вертикальная координата контрольной точки>,
<горизонтальная координата конечной точки>,
<вертикальная координата конечной точки>)
```

Пример:

```
ctxCanvas.beginPath();
ctxCanvas.moveTo(100, 100);
ctxCanvas.quadraticCurveTo(200, 100, 200, 200);
ctxCanvas.stroke();
```

Web-сценарий из листинга 3.83 рисует сектор окружности красного цвета.

Листинг 3.83. Пример создания сектора окружности

```
ctxCanvas.beginPath();
ctxCanvas.strokeStyle = "red";
ctxCanvas.fillStyle = "red";
ctxCanvas.moveTo(100, 100);
ctxCanvas.quadraticCurveTo(200, 100, 200, 200);
ctxCanvas.lineTo(100, 200);
ctxCanvas.lineTo(100, 100);
ctxCanvas.fill();
```

А код из листинга 3.84 рисует прямоугольник со скругленными углами.

Листинг 3.84. Пример создания прямоугольника со скругленными углами

```
ctxCanvas.beginPath();
ctxCanvas.moveTo(20, 0);
ctxCanvas.lineTo(180, 0);
ctxCanvas.quadraticCurveTo(200, 0, 200, 20);
ctxCanvas.lineTo(200, 80);
ctxCanvas.quadraticCurveTo(200, 100, 180, 100);
ctxCanvas.lineTo(20, 100);
ctxCanvas.quadraticCurveTo(0, 100, 0, 80);
ctxCanvas.lineTo(0, 20);
ctxCanvas.quadraticCurveTo(0, 0, 20, 0);
ctxCanvas.stroke();
```

Прямоугольники

Нарисовать прямоугольник в составе сложного контура можно, вызвав метод `rect`:

<контекст рисования>`.rect(`*<горизонтальная координата>*`,`
<вертикальная координата>`,` *<ширина>*`,` *<высота>*`)`

По окончании рисования прямоугольника перо будет установлено в точку с координатами [0,0], т. е. в верхний левый угол канвы.

Пример:

```
ctxCanvas.beginPath();
ctxCanvas.moveTo(0, 0);
ctxCanvas.fillStyle = "red";
ctxCanvas.rect(50, 50, 50, 50);
ctxCanvas.fillStyle = "green";
ctxCanvas.rect(75, 75, 50, 50);
ctxCanvas.fillStyle = "blue";
ctxCanvas.rect(100, 100, 50, 50);
ctxCanvas.fill();
```

Рисуем фигуру, состоящую их трех накладывающихся друг на друга разноцветных квадратов.

Задание стиля линий

Канва позволяет задать стиль линий, а именно форму их начальных и конечных точек и точек соединения линий друг с другом.

Свойство `lineCap` задает форму начальных и конечных точек линий. Его значение может быть одной из следующих строк:

- `"butt"` — начальная и конечная точки как таковые отсутствуют (значение по умолчанию);

- "round" — начальная и конечная точки имеют вид кружков;
- "square" — начальная и конечная точки имеют вид квадратиков.

Пример:

```
ctxCanvas.beginPath();
ctxCanvas.lineWidth = 10;
ctxCanvas.lineCap = "round";
ctxCanvas.moveTo(20, 20);
ctxCanvas.lineTo(180, 20);
ctxCanvas.stroke();
```

Рисуем толстую прямую линию, начальная и конечная точки которой имеют вид кружков.

Свойство lineJoin задает форму точек соединения линий друг с другом. Его значение может быть одной из следующих строк:

- "miter" — точки соединения имеют вид острого или тупого угла (значение по умолчанию);
- "round" — точки соединения, образующие острые углы, скругляются;
- "bevel" — острые углы, образуемые соединяющимися линиями, как бы срезаются.

Код из листинга 3.85 рисует контур треугольника толстыми линиями, причем точки соединения этих линий, образующие острые углы, будут скругляться.

Листинг 3.85. Создание треугольника со скругленными вершинами

```
ctxCanvas.beginPath();
ctxCanvas.lineWidth = 10;
ctxCanvas.lineJoin = "round";
ctxCanvas.moveTo(20, 20);
ctxCanvas.lineTo(380, 20);
ctxCanvas.lineTo(200, 280);
ctxCanvas.closePath();
ctxCanvas.stroke();
```

Свойство miterLimit задает дистанцию от точки соединения, на которую могут выступать острые углы, образованные соединением линий, когда для свойства lineJoin задано значение "miter". Углы, выступающие на бóльшую дистанцию, будут срезаны. Значение по умолчанию — 10 пикселов.

В листинге 3.86 приведен код, который рисует контур треугольника толстыми линиями без срезания углов, образованных соединяющимися линиями.

Листинг 3.86. Создание треугольника без скругления вершин

```
ctxCanvas.beginPath();
ctxCanvas.lineWidth = 10;
```

```
ctxCanvas.lineJoin = "miter";
ctxCanvas.miterLimit = 0;
ctxCanvas.moveTo(20, 20);
ctxCanvas.lineTo(380, 20);
ctxCanvas.lineTo(200, 280);
ctxCanvas.closePath();
ctxCanvas.stroke();
```

Определение вхождения точки в состав контура

Иногда бывает необходимо выяснить, входит ли точка с заданными координатами в состав контура сложной фигуры. Сделать это мы можем с помощью метода `isPointInPath`:

<контекст рисования>`.isPointInPath(`*<горизонтальная координата>*`,`
<вертикальная координата>`)`

Обе *координаты* проверяемой точки указываются в виде чисел в пикселах. Метод возвращает `true`, если точка с такими координатами входит в состав контура, и `false` — в противном случае.

Пример:

```
ctxCanvas.beginPath();
ctxCanvas.rect(50, 50, 50, 50);
ctxCanvas.stroke();
if (ctxCanvas.isPointInPath(60, 50)) {
  window.alert("Точка входит в состав контура");
}
```

Вывод текста

Для вывода текста, представляющего собой один лишь контур без заливки, используется метод `strokeText`:

<контекст рисования>`.strokeText(`*<выводимый текст>*`,`
<горизонтальная координата>`,` *<вертикальная координата>*
`[,` *<максимальная ширина>*`])`

С первыми тремя параметрами все ясно. Четвертый, необязательный, параметр определяет максимальное значение ширины, которую может принять выводимый на канву текст. Если выводимый текст получается шире, Web-обозреватель выводит его либо шрифтом с уменьшенной шириной символов (если данный шрифт поддерживает такое начертание), либо шрифтом меньшего размера. Метод `strokeText` не возвращает результат.

Пример:

```
ctxCanvas.strokeStyle = "blue";
ctxCanvas.strokeText("JavaScript", 200, 50, 100);
```

Метод `fillText` выводит заданный текст в виде одной заливки, без контура. Вызывается он так же, как метод `strokeText`.

Пример:

```
ctxCanvas.fillStyle = "yellow";
ctxCanvas.fillText("HTML, CSS", 50, 40);
```

Свойство `font` задает параметры шрифта, которым будет выводиться текст. Эти параметры указывают в том же формате, что и у значения атрибута CSS `font`, в виде строки.

Пример:

```
ctxCanvas.font = "italic 16pt Verdana";
ctxCanvas.strokeText("JavaScript", 200, 50, 100);
```

Свойство `textAlign` устанавливает горизонтальное выравнивание выводимого текста относительно точки, в которой он будет выведен (координаты этой точки задаются вторым и третьим параметрами методов `strokeText` и `fillText`). Это свойство может принимать следующие значения:

- `"left"` — выравнивание по левому краю;
- `"right"` — выравнивание по правому краю;
- `"start"` — выравнивание по левому краю, если текст выводится по направлению слева направо, и по правому краю в противном случае (значение по умолчанию);
- `"end"` — выравнивание по правому краю, если текст выводится по направлению слева направо, и по левому краю в противном случае;
- `"center"` — выравнивание по центру.

Пример:

```
ctxCanvas.textAlign = "center";
ctxCanvas.fillText("HTML, CSS", 100, 100);
```

Свойство `textBaseline` позволяет задать вертикальное выравнивание выводимого текста относительно точки, в которой он будет выведен. Доступны следующие значения:

- `"top"` — выравнивание по верху прописных букв;
- `"hanging"` — выравнивание по верху строчных букв;
- `"middle"` — выравнивание по средней линии строчных букв;
- `"alphabetic"` — выравнивание по базовой линии букв европейских алфавитов (значение по умолчанию);
- `"ideographic"` — выравнивание по базовой линии иероглифических символов (она находится чуть ниже базовой линии букв европейских алфавитов);
- `"bottom"` — выравнивание по низу букв.

Пример:

```
ctxCanvas.textBaseline = "top";
ctxCanvas.fillText("HTML, CSS", 100, 100);
```

Еще нам может пригодиться метод measureText, позволяющий узнать ширину текста, выводимого на канву:

```
<контекст рисования>.measureText(<текст>)
```

Текст указывается в виде строки. Метод возвращает объект с единственным свойством width, которое и хранит ширину текста в пикселах, заданную в виде числа.

Пример:

```
var s = "HTML, CSS, JavaScript, PHP, MySQL";
ctxCanvas.font = "bold 24pt Tahoma";
var w = ctxCanvas.measureText(s).width;
ctxCanvas.fillText(s, 100, 100, 2 * w / 3);
```

Вычисляем ширину строки, после чего выводим ее на канву, указав для него максимальную ширину, равную 2/3 от вычисленной.

Использование сложных цветов

Помимо однотонных цветов, канва HTML 5 позволяет использовать для закрашивания линий и заливок градиенты и даже графические изображения.

Линейный градиент

Линейный градиент создают в три этапа. Первый этап — вызов метода createLinearGradient — собственно создание линейного градиента:

```
<контекст рисования>.createLinearGradient
↳(<горизонтальная координата начальной точки>,
<вертикальная координата начальной точки>,
<горизонтальная координата конечной точки>,
<вертикальная координата конечной точки>)
```

Координаты начальной и конечной точек градиента отсчитываются относительно канвы.

Метод createLinearGradient возвращает объект класса CanvasGradient, представляющий созданный нами линейный градиент. С его помощью мы указываем цвета, формирующие градиент.

Второй этап — расстановка ключевых точек градиента. Здесь нам понадобится метод addColorStop класса CanvasGradient:

```
<градиент>.addColorStop(<положение ключевой точки>, <цвет>)
```

Первый параметр задается в виде числа от 0.0 (начало прямой) до 1.0 (конец прямой). Результат этот метод не возвращает.

Третий этап — использование готового линейного градиента. Для этого представляющий его объект класса CanvasGradient следует присвоить свойству strokeStyle или fillStyle канвы.

Пример:

```
var lgSample = ctxCanvas.createLinearGradient(0, 0, 100, 50);
lgSample.addColorStop(0, "black");
lgSample.addColorStop(0.4, "rgba(0, 0, 255, 0.5)");
lgSample.addColorStop(1, "#ff0000");
ctxCanvas.fillStyle = lgSample;
ctxCanvas.fillRect(0, 0, 200, 100);
```

Создаем линейный градиент, который будет "распространяться" по прямой с координатами начальной и конечной точек [0,0] и [100,50], а затем формируем на нем три ключевые точки:

- первую — находящуюся в начальной точке воображаемой прямой и задающую черный цвет;
- вторую — находящуюся в точке, отстоящей на 40% длины воображаемой прямой от ее начальной точки, и задающую полупрозрачный синий цвет;
- третью — находящуюся в конечной точке воображаемой прямой и задающую красный цвет.

Наконец, на основе созданного таким образом градиента рисуем прямоугольник.

Покрытие градиентом закрашиваемой фигуры

Предположим, что мы нарисовали на канве три прямоугольника и применили к ним градиент. Первый прямоугольник нарисован в точке [0,0] (в начале воображаемой прямой градиента, в смысле, в его первой ключевой точке), второй — в точке [30,20] (во второй ключевой точке), третий — в точке [80,40] (в конце градиента — его третьей ключевой точке). Иначе говоря, мы "расставили" наши прямоугольники во всех ключевых точках градиента.

Как будут окрашены эти прямоугольники? Давайте посмотрим.

- Первый прямоугольник будет окрашен, в основном, в черный цвет, заданный в первой ключевой точке градиента.
- Второй прямоугольник будет окрашен, в основном, в полупрозрачный синий цвет, заданный во второй ключевой точке градиента.
- Третий прямоугольник будет окрашен, в основном, в красный цвет, заданный в третьей ключевой точке градиента.

Следовательно, созданный нами градиент не "втиснулся" в каждый из нарисованных прямоугольников целиком, а как бы зафиксировался на самой канве, а прямоугольники только "проявили" фрагменты этого градиента, соответствующие их размерам. Другими словами, градиент задается для целой канвы, а фигуры, к которым он применен, окрашиваются соответствующими его фрагментами.

Если мы захотим, чтобы какая-то фигура была окрашена градиентом полностью, то зададим для этого градиента такие координаты воображаемой прямой, чтобы он "покрыл" всю фигуру.

Радиальный градиент

Радиальный градиент также создают в три этапа. Первый этап — вызов метода `createRadialGradient` — создание радиального градиента:

```
<контекст рисования>.createRadialGradient
(<горизонтальная координата центра внутренней окружности>,
<вертикальная координата центра внутренней окружности>,
<радиус внутренней окружности>,
<горизонтальная координата центра внешней окружности>,
<вертикальная координата центра внешней окружности>,
<радиус внешней окружности>)
```

Параметры этого метода определяют координаты центров и радиусы обеих окружностей, описывающих радиальный градиент. Они задаются в пикселах в виде чисел.

Метод `createRadialGradient` возвращает объект класса `CanvasGradient`, представляющий созданный нами радиальный градиент.

Второй этап — расстановка ключевых точек — выполняется с помощью уже знакомого нам метода `addColorStop` класса `CanvasGradient`. Только в данном случае первый параметр определит относительное положение создаваемой ключевой точки на промежутке между внутренней и внешней окружностями. Он задается в виде числа от 0.0 (начало промежутка, т. е. внутренняя окружность) до 1.0 (конец промежутка, т. е. внешняя окружность).

И третий этап — использование созданного градиента.

Пример:

```
var rgSample = ctxCanvas.createRadialGradient(100, 100, 10, 150, 100, 120);
rgSample.addColorStop(0, "#cccccc");
rgSample.addColorStop(0.8, "black");
rgSample.addColorStop(1, "#00ff00");
ctxCanvas.fillStyle = rgSample;
ctxCanvas.fillRect(0, 0, 200, 100);
```

Создаем радиальный градиент, "распространяющийся" от внутренней окружности, центр которой находится в точке с координатами [100,100], а радиус равен 10 пикселам, к внешней окружности с центром в точке [150,100] и радиусом в 120 пикселов. Далее добавляем три ключевые точки:

- расположенную в начальной точке воображаемого промежутка между окружностями (т. е. на внутренней окружности) и задающую серый цвет;
- расположенную в точке, отстоящей на 80% длины воображаемого промежутка между окружностями от его начальной точки, и задающую черный цвет;
- расположенную в конечной точке воображаемого промежутка между окружностями (т. е. на внешней окружности) и задающую зеленый цвет.

Радиальный градиент ведет себя точно так же, как линейный — фиксируется на канве и частично "проявляется" на фигурах, к которым применен.

Графический цвет

Графический цвет — это обычное изображение, которым закрашиваются линии или заливки. Таким изображением может быть содержимое как графического файла, так и другой канвы.

Графический цвет создают в три этапа. Первый этап необходим только в том случае, если мы используем в качестве цвета содержимое графического файла, который не выведен на страницу. Такой файл нужно как-то загрузить, удобнее всего — с помощью объекта класса `Image`, который представляет графическое изображение.

Сначала с помощью давно знакомого нам оператора `new` создаем объекта класса `Image`. Далее мы присваиваем свойству `src` этого объекта интернет-адрес загружаемого графического файла в виде строки. Все — теперь можем использовать данный экземпляр объекта `Image` для создания графического цвета.

Второй этап — собственно создание графического цвета с помощью метода `createPattern`:

<контекст рисования>`.createPattent(`*<графическое изображение или канва>*`,` *<режим повторения>*`)`

Первый параметр задает графическое изображение в виде объекта класса `Image`, тег `<img>` или канву в виде представляющего их объекта.

Часто бывает так, что размеры заданного графического изображения меньше, чем фигуры, к которой должен быть применен графический цвет. В этом случае изображение повторяется столько раз, чтобы полностью "вымостить" линию или заливку. Режим такого повторения задает второй параметр метода `createPattern`. Его значение должно быть одной из следующих строк:

- `"repeat"` — изображение будет повторяться по горизонтали и вертикали;
- `"repeat-x"` — изображение будет повторяться только по горизонтали;
- `"repeat-y"` — изображение будет повторяться только по вертикали;
- `"no-repeat"` — изображение не будет повторяться никогда; в этом случае часть фигуры останется не занятой им.

Метод `createPattern` возвращает объект класса `CanvasPattern`, представляющий созданный нами графический цвет:

Третий этап — использование готового графического цвета — выполняется так же, как для градиентов.

Примеры:

```
var imgSample = new Image();
imgSample.src = "graphic_color.jpg";
var cpSample = ctxCanvas.createPattern(imgSample, "repeat");
ctxCanvas.fillStyle = cpSample;
ctxCanvas.fillRect(0, 0, 200, 100);
```

Загружаем графический файл graphic_color.jpg, создаем на его основе повторяющийся по горизонтали и вертикали графический цвет и рисуем с его помощью прямоугольник.

```
<img id="pattern" src="pattern.gif">
. . .
var oPattern = document.getElementById("pattern");
var cpSample = ctxCanvas.createPattern(oPattern, "repeat");
ctxCanvas.fillStyle = cpSample;
ctxCanvas.fillRect(0, 0, 200, 100);
```

Выполняем то же самое с применением изображения, выведенного на страницу.

```
<canvas id="pattern" width="100" height="100"></canvas>
. . .
var oPattern = document.getElementById("pattern");
var cpSample = ctxCanvas.createPattern(oPattern, "repeat");
ctxCanvas.fillStyle = cpSample;
ctxCanvas.fillRect(0, 0, 200, 100);
```

А теперь — с применением графики, выведенной на другой канве.

Графический цвет не фиксируется на канве, а полностью применяется к рисуемой фигуре. В этом его принципиальное отличие от градиентов.

Вывод внешних изображений

Внешним по отношению к канве называется изображение, выведенное на страницу, хранящееся в отдельном файле, или содержимое другой канвы. Вывести такое изображение на канву проще всего методом `drawImage`.

Этот метод имеет три различных формата вызова. Сначала рассмотрим самый простой формат.

<контекст рисования>`.drawImage(`*<графическое изображение или канва>*`,`
<горизонтальная координата>`,` *<вертикальная координата>*`)`

Первый параметр задает графическое изображение или другую канву в виде объекта. Второй и третий параметры определяют координаты точки канвы, где должен находиться верхний левый угол выводимого изображения; они задаются в пикселах в виде чисел. Метод `drawImage` не возвращает результат.

Примеры:

```
var imgSample = new Image();
imgSample.src = "someimage.jpg";
ctxCanvas.drawImage(imgSample, 0, 0);
```

Загружаем изображение из файла someimage.jpg и выводим его на канву так, чтобы его верхний левый угол находился в точке [0,0], т. е. в верхнем левом углу канвы.

```
<img id="image" src="someimage.jpg">
. . .
var oImage = document.getElementById("image");
ctxCanvas.drawImage(oImage, 0, 0);
```

Выводим на канву изображение, присутствующее на странице.

Если нам нужно при выводе внешнего изображения изменить его размеры, к нашим услугам — расширенный формат метода `drawImage`:

```
<контекст рисования>.drawImage(<графическое изображение или канва>,
<горизонтальная координата>, <вертикальная координата>,
<ширина>, <высота>)
```

Пример:

```
ctxCanvas.drawImage(oImage, 0, 0, 400, 300);
```

Растягиваем выводимое изображение так, чтобы занять канву целиком.

Рассмотрим теперь самый сложный случай — вырезание из внешнего изображения фрагмента и вывод его на канву с изменением размеров. Для этого применяется третий по счету формат метода `drawImage`:

```
<контекст рисования>.drawImage(<графическое изображение или канва>,
<горизонтальная координата вырезаемого фрагмента>,
<вертикальная координата вырезаемого фрагмента>,
<ширина вырезаемого фрагмента>, <высота вырезаемого фрагмента>,
<горизонтальная координата вывода>, <вертикальная координата вывода>,
<ширина вывода>, <высота вывода>)
```

Второй и третий параметры определяют координаты левого верхнего угла вырезаемого фрагмента. Они задаются относительно внешнего изображения (в пикселах в виде чисел).

Четвертый и пятый параметры определяют ширииу и высоту вырезаемого из внешнего изображения фрагмента. Они также задаются относительно внешнего изображения (в пикселах в виде чисел).

Пример:

```
ctxCanvas.drawImage(oImage, 20, 40, 40, 20, 0, 0, 400, 300);
```

Вырезаем из присутствующего на странице изображения фрагмент с верхним левым углом в точке [20,40], шириной в 40 и высотой в 20 пикселов и выводим этот фрагмент на канву, растягивая его так, чтобы занять канву целиком.

Создание тени у рисуемой графики

Канва позволяет создавать тень у всех рисуемых фигур. Для задания ее параметров применяют четыре свойства:

- `shadowOffsetX` — смещение тени по горизонтали относительно фигуры (в виде числа в пикселах; значение по умолчанию — 0);

- `shadowOffsetY` — смещение тени по вертикали относительно фигуры (в виде числа в пикселах; значение по умолчанию — 0);
- `shadowBlur` — степень размытия тени в виде числа; чем больше это число, тем сильнее размыта тень (значение по умолчанию — 0, т. е. отсутствие размытия);
- `shadowColor` — цвет тени (по умолчанию — черный непрозрачный).

Пример:

```
ctxCanvas.shadowOffsetX = 2;
ctxCanvas.shadowOffsetY = -2;
ctxCanvas.shadowBlur = 4;
ctxCanvas.shadowColor = "rgba(128, 128, 128, 0.5)";
ctxCanvas.fillText("Двое: я и моя тень.", 150, 50);
```

Преобразование системы координат

Еще канва позволяет выполнять преобразование системы координат: ее смещение, поворот или масштабирование. Любая графика, которую мы нарисуем впоследствии, будет создаваться в уже измененной системе координат.

Сохранение и загрузка состояния

Первое, что нам нужно рассмотреть применительно к преобразованиям, — сохранение и загрузка состояния канвы. Эти возможности нам очень пригодятся в дальнейшем.

При сохранении состояния канвы сохраняются:

- все заданные трансформации (будут описаны далее);
- значения свойств `globalAlpha`, `globalCompositeOperation` (будет описано далее), `fillStyle`, `lineCap`, `lineJoin`, `lineWidth`, `miterLimit` и `strokeStyle`;
- все заданные маски (будут описаны далее).

Метод `save` сохраняет состояние канвы. Он не принимает параметров и не возвращает результат.

Состояние канвы сохраняется в памяти компьютера и впоследствии может быть восстановлено. Более того, сохранять состояние канвы можно несколько раз; при этом все предыдущие состояния остаются в памяти и их также можно восстановить.

Восстановить сохраненное ранее состояние можно вызовом метода `restore`. Он также не принимает параметров и не возвращает результат.

При вызове этого метода будет восстановлено самое последнее из сохраненных состояний канвы. При последующем его вызове будет восстановлено предпоследнее сохраненное состояние и т. д. Этой особенностью часто пользуются для создания сложной графики.

Смещение начала координат канвы

Для смещения начала координат канвы в другую точку следует вызвать не возвращающий результат метод `translate`:

```
<контекст рисования>.translate(<горизонтальная координата>,
<вертикальная координата>)
```

Листинг 3.87. Смещение начала координат канвы

```
ctxCanvas.save();
ctxCanvas.translate(100, 100);
ctxCanvas.fillRect(0, 0, 50, 50);
ctxCanvas.translate(100, 100);
ctxCanvas.fillRect(0, 0, 50, 50);
ctxCanvas.restore();
ctxCanvas.fillRect(0, 0, 50, 50);
```

Код из листинга 3.87 делает следующее:

1. Сохраняет текущее состояние канвы.
2. Смещает начало координат в точку [100,100].
3. Рисует квадрат размерами 50×50 пикселов, верхний левый угол которого находится в точке начала координат.
4. Опять смещает начало координат в точку [100,100]. Обратим внимание, что координаты этой точки теперь отсчитываются от нового начала системы координат, установленного предыдущим вызовом метода `translate`.
5. Снова рисует квадрат размерами 50×50 пикселов, верхний левый угол которого находится в начале координат.
6. Восстанавливает сохраненное состояние канвы, в том числе и положение начала системы координат (это положение по умолчанию, т. е. верхний левый угол канвы).
7. Рисует третий по счету квадрат размерами 50×50 пикселов, верхний левый угол которого находится в начале координат.

В результате мы увидим три квадрата, расположенные на воображаемой диагонали, тянущейся от верхнего левого угла канвы вниз и вправо.

> ***Внимание!***
>
> К сожалению, не существует простого способа вернуться к системе координат по умолчанию. Можно лишь выполнить смещение системы координат в нужную точку или предварительно сохранить состояние системы координат перед любыми преобразованиями и потом восстановить его.

Поворот системы координат

Не возвращающий результат метод `rotate` позволяет повернуть оси системы координат на произвольный угол вокруг точки начала координат:

<контекст рисования>.rotate(*<угол поворота>*)

Угол поворота задается в виде числа в радианах и отсчитывается от горизонтальной оси. Отметим, что поворот всегда выполняется по часовой стрелке.

Пример:

```
ctxCanvas.translate(200, 150);
for (var i = 0; i < 3; i++) {
  ctxCanvas.rotate(Math.PI / 6);
  ctxCanvas.strokeRect(-50, -50, 100, 100);
}
```

Сдвигаем начало координат в центр канвы (точку [200,150]), после чего трижды поворачиваем систему координат на $\pi/6$ радиан (30°) и рисуем в центре канвы квадрат без заливки. Обратим внимание, что каждый последующий поворот системы координат выполняется с учетом того, что она уже была повернута ранее.

Изменение масштаба системы координат

Не возвращающий результат метод `scale` изменяет масштаб системы координат канвы в бóльшую или меньшую сторону:

<контекст рисования>.scale(*<масштаб по горизонтали>*,
<масштаб по вертикали>)

Параметры этого метода задают масштаб для горизонтальной и вертикальной оси системы координат в виде чисел. Числа меньше 1.0 задают уменьшение масштаба, а числа больше 1.0 — увеличение; если иужно оставить масштаб по какой-то из осей неизменным, достаточно указать значение 1.0 соответствующего параметра. Применение метода `scale` иллюстрирует листинг 3.88.

Листинг 3.88. Изменение масштаба системы координат

```
ctxCanvas.save();
ctxCanvas.strokeRect(0, 0, 50, 50);
ctxCanvas.scale(3, 1);
ctxCanvas.strokeRect(0, 0, 50, 50);
ctxCanvas.restore();
ctxCanvas.save();
ctxCanvas.scale(1, 3);
ctxCanvas.strokeRect(0, 0, 50, 50);
ctxCanvas.restore();
ctxCanvas.save();
ctxCanvas.scale(3, 3);
ctxCanvas.strokeRect(0, 0, 50, 50);
ctxCanvas.restore();
```

Код из листинга 3.88 делает следующее:

1. Сохраняет текущее состояние канвы.
2. Рисует квадрат размерами 50×50 пикселов, верхний левый угол которого находится в начале координат.
3. Увеличивает масштаб горизонтальной координатной оси в 3 раза.
4. Рисует второй квадрат размерами 50×50 пикселов, верхний левый угол которого находится в начале координат.
5. Восстанавливает сохраненное ранее состояние канвы и сохраняет его снова.
6. Увеличивает масштаб вертикальной координатной оси в 3 раза.
7. Рисует третий квадрат размерами 50×50 пикселов, верхний левый угол которого находится в начале координат.
8. Восстанавливает сохраненное ранее состояние канвы и сохраняет его снова.
9. Увеличивает масштаб обеих координатных осей в 3 раза.
10. Рисует четвертый квадрат размерами 50×50 пикселов, верхний левый угол которого находится в начале координат.
11. Восстанавливает сохраненное ранее состояние канвы.

В результате мы увидим четыре прямоугольника с реальными размерами 50×50, 150×50, 50×150 и 150×150 пикселов.

Управление наложением графики

Когда мы рисуем какую-либо фигуру поверх уже существующей, новая фигура накладывается на старую, перекрывая ее. Это поведение канвы по умолчанию, которое мы можем изменить, указав другие значения для свойства `globalCompositeOperation`.

Вот перечень поддерживаемых значений:

- `"source-over"` — новая фигура накладывается на старую, перекрывая ее (значение по умолчанию);
- `"source-atop"` — отображается только та часть новой фигуры, которая накладывается на старую; остальная часть новой фигуры не выводится. Старая фигура выводится целиком и находится ниже новой;
- `"source-in"` — отображается только та часть новой фигуры, которая накладывается на старую. Остальные части новой и старой фигур не выводятся;
- `"source-out"` — отображается только та часть новой фигуры, которая не накладывается на старую. Остальные части новой фигуры и вся старая фигура не выводятся;
- `"destination-over"` — новая фигура перекрывается старой;
- `"destination-atop"` — отображается только та часть старой фигуры, которая накладывается на новую; остальная часть старой фигуры не выводится. Новая фигура выводится целиком и находится ниже старой;

- "destination-in" — отображается только та часть старой фигуры, на которую накладывается новая. Остальные части новой и старой фигур не выводятся;
- "destination-out" — отображается только та часть старой фигуры, на которую не накладывается новая. Остальные части новой фигуры и вся старая фигура не выводятся;
- "lighter" — цвета накладывающихся частей старой и новой фигур складываются, результирующий цвет получается более светлым, окрашиваются накладывающиеся части фигур;
- "xor" — отображаются только те части старой и новой фигур, которые не накладываются друг на друга;
- "copy" — выводится только новая фигура; все старые фигуры удаляются с канвы.

Заданный нами способ наложения действует только для графики, которую мы нарисуем после этого. На уже нарисованную графику он не влияет.

Пример:

```
ctxCanvas.fillStyle = "blue";
ctxCanvas.fillRect(0, 50, 400, 200);
ctxCanvas.fillStyle = "red";
ctxCanvas.globalCompositeOperation = "source-over";
ctxCanvas.fillRect(100, 0, 200, 300);
```

Этот код рисует два накладывающихся прямоугольника разных цветов и позволит изучить поведение канвы при разных значениях свойства `globalCompositeOperation`. Изменяем значение этого свойства, перезагружаем страницу нажатием клавиши <F5> и смотрим, что получится.

Использование масок

Маской называется особая фигура, задающая своего рода "окно", сквозь которое будет видна часть графики, нарисованной на канве. Вся графика, не попадающая в это "окно", будет скрыта; при этом сама маска на канву не выводится.

Вот действия, необходимые для создания маски.

1. Рисуем сложный контур, который станет маской.
2. Обязательно делаем его закрытым.
3. Вместо вызова методов `stroke` или `fill` вызываем метод `clip`. Этот метод не принимает параметров и не возвращает результат.
4. Рисуем графику, которая будет находиться под маской.

После этого нарисованная нами на шаге 4 графика будет частично видна сквозь маску. Требуемый результат достигнут.

Код, приведенный в листинге 3.89, рисует маску в виде треугольника, а потом — прямоугольник. Часть этого прямоугольника будет видна сквозь маску.

Листинг 3.89. Пример использования маски

```
ctxCanvas.beginPath();
ctxCanvas.moveTo(100, 150);
ctxCanvas.lineTo(200, 0);
ctxCanvas.lineTo(200, 300);
ctxCanvas.closePath();
ctxCanvas.clip();
ctxCanvas.fillRect(0, 100, 400, 100);
```

Работа с отдельными пикселами

И, наконец, мы можем работать с отдельными пикселами графики, нарисованной на канве. Это может пригодиться при создании очень сложного изображения или при наложении на графику специальных эффектов наподобие размытия.

Получение массива пикселов, представляющего нарисованную графику

Получить массив пикселов, представляющий нарисованную на канве графику или ее фрагмент, мы можем вызовом метода `getImageData`:

```
<контекст рисования>.getImageData(<горизонтальная координата>,
<вертикальная координата>, <ширина>, <высота>)
```

Первые два параметра указывают координату левого верхнего угла фрагмента нарисованной на канве графики, два последних — его ширину и высоту. Все параметры задаются в пикселах в виде чисел.

Метод `getImageData` возвращает объект класса `ImageData`, представляющий массив пикселов, что составляют присутствующий на канве фрагмент графики с указанными нами параметрами.

Пример:

```
var idSample = ctxCanvas.getImageData(200, 150, 100, 100);
```

Создание пустого массива пикселов

Чтобы самостоятельно нарисовать какое-либо изображение, можно сначала создать пустой массив пикселов, вызвав метод `createImageData`:

```
<контекст рисования>.createImageData(<ширина>, <высота>)
```

Оба параметра задаются в виде чисел в пикселах. Метод также возвращает объект класса `ImageData`.

Пример:

```
var idEmpty = ctxCanvas.createImageData(400, 300);
```

Создаем пустой массив пикселов тех же размеров, что и сама канва.

Манипуляция пикселами

Теперь мы можем начать работать с отдельными пикселами полученного массива, задавая для них цвет и значения полупрозрачности и тем самым формируя какое-либо изображение или изменяя уже существующее.

Класс `ImageData` поддерживает свойство `data`. Его значением является объект-коллекция, хранящая набор чисел:

- первое число представляет собой долю красного цвета в цвете первого пиксела массива;
- второе число — долю зеленого цвета в цвете первого пиксела;
- третье число — долю синего цвета в цвете первого пиксела;
- четвертое число — степень полупрозрачности цвета первого пиксела;
- пятое число — долю красного цвета в цвете второго пиксела;
- шестое число — долю зеленого цвета в цвете второго пиксела;
- седьмое число — долю синего цвета в цвете второго пиксела;
- восьмое число — степень полупрозрачности цвета второго пиксела;
- . . .
- *n*–3-е число — долю красного цвета в цвете последнего пиксела;
- *n*–2-е число — долю зеленого цвета в цвете последнего пиксела;
- *n*–1-е число — долю синего цвета в цвете последнего пиксела;
- *n*-е число — степень полупрозрачности цвета последнего пиксела.

Все значения, включая и степень полупрозрачности, должны укладываться в диапазон от 0 до 255. Для степени полупрозрачности значение 0 задает полпую прозрачность, а 255 — полную непрозрачность.

Нумерация пикселов в массиве идет слева направо и сверху вниз, т. е. по строкам.

Поскольку набор чисел, хранящийся в свойстве `data` класса `ImageData`, представляет собой коллекцию, для доступа к отдельным значениям мы можем использовать тот же синтаксис, что и в случае обычных массивов JavaScript. Также мы можем воспользоваться поддерживаемым всеми коллекциями свойством `length`, возвращающим размер коллекции.

Для пустого массива все пикселы будут иметь прозрачный черный цвет. (Набор, хранящийся в свойстве `data`, будет содержать иули.)

Примеры:

```
idEmpty.data[0] = 255;
idEmpty.data[1] = 0;
idEmpty.data[2] = 0;
idEmpty.data[3] = 255;
```

Задаем для первого пиксела (левого верхнего) полностью непрозрачный красный цвет.

```
idEmpty.data[0] = 0;
idEmpty.data[1] = 255;
idEmpty.data[2] = 0;
idEmpty.data[3] = 127;
```

А для последнего пиксела (правого нижнего) — полупрозрачный зеленый.

Вывод массива пикселов

Завершив формирование нового изображения в массиве пикселов, мы можем вывести его на канву, вызвав метод `putImageData`:

```
<контекст рисования>.putImageData(<массив пикселов>,
<горизонтальная координата вывода>, <вертикальная координата вывода>
[, <горизонтальная координата выводимого фрагмента>,
<вертикальная координата выводимого фрагмента>,
<ширина выводимого фрагмента>, <высота выводимого фрагмента>])
```

Второй и третий параметры задают координату точки, где будет находиться левый верхний угол выводимого массива пикселов.

Четвертый и пятый параметры задают координату точки, где находится левый верхний угол фрагмента массива пикселов, который должен быть выведен на канву, а шестой и седьмой — ширину и высоту выводимого фрагмента. Если эти параметры не указаны, будет выведен весь массив пикселов.

Все значения координат и размеров должны указываться в виде чисел и измеряться в пикселах.

Пример:

```
ctxCanvas(idEmpty, 50, 50);
```

В листинге 3.90 приведен код, рисующий на странице прямоугольник с градиентной заливкой. В заливке зеленый цвет плавно сменяется синим, а потом — красным; при этом цвета становятся все более и более прозрачными.

Листинг 3.90. Вывод массива пикселов

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Манипуляция пикселами</title>
    <script>
      window.addEventListener("load", function() {
        var oCanvas = document.getElementById("cnv");
        var ctxCanvas = oCanvas.getContext("2d");
        var idEmpty = ctxCanvas.createImageData(255, 255);
        for (var i = 0; i < idEmpty.data.length; i += 4) {
          idEmpty.data[i + 0] = i / 4 / 255;
          idEmpty.data[i + 1] = 255 - i / 4 / 255;
          idEmpty.data[i + 2] = i / 4 / 255;
```

```
          idEmpty.data[i + 3] = 255 - i / 4 / 255;
        }
        ctxCanvas.putImageData(idEmpty, 0, 0);
      }, false);
    </script>
  </head>
  <body>
    <canvas id="cnv" width="400" height="300"></canvas>
  </body>
</html>
```

3.21.8. Хранилище

Мы уже знаем способ сохранить произвольные данные на стороне клиента — файлы cookies. Однако лучше всего для этой цели пользоваться средствами DOM 3, о которых сейчас пойдет речь.

Хранилище — это инструмент для сохранения на стороне клиента произвольных данных любого типа. Хранилище позволяет запомнить любое значение, указав для него уникальное имя, а впоследствии извлечь его, сославшись на заданное ранее имя, и использовать в вычислениях.

Сессионное и локальное хранилища

Современные Web-обозреватели предлагают разработчикам сценариев два различных хранилища:

- *Сессионное хранилище* — хранит данные, пока в окне Web-обозревателя открыта текущая страница; после перехода на другую страницу или закрытия окна все сохраненные данные будут удалены. Сессионное хранилище подходит лишь для временного хранения данных при случайном обновлении страницы посетителем.
- *Локальное хранилище* — сохраняет данные даже после перехода на другую страницу или закрытия окна Web-обозревателя. Позволяет хранить данные неопределенно долгое время.

Нужно помнить, что хранилище любого типа хранит данные, относящиеся к какой-то одной странице. Данные, сохраненные любой другой страницей, при этом считать невозможно — это сделано ради безопасности.

Работа с хранилищем

Сессионное хранилище доступно через свойство `sessionStorage`, а локальное — через свойство `localStorage`. Оба этих свойства поддерживаются объектом `window`.

> ***Внимание!***
>
> Хранилища обоих типов доступны лишь в том случае, если страница была загружена с Web-сервера. При загрузке страницы с локального диска свойства `sessionStorage` и `localStorage` будут недоступны.

Хранилище любого типа представляется объектом класса `Storage`. Все манипуляции по сохранению и чтению данных выполняются посредством вызова методов этого объекта.

Примеры:

```
var stSession = window.sessionStorage;
```

Получаем сессионное хранилище.

```
if (window.localStorage) {
  var stLocal = window.localStorage;
  // Работаем с локальным хранилищем
} else {
  // Web-обозреватель не поддерживает локальное хранилище
}
```

Получаем доступ к локальному хранилищу с учетом того, что Web-обозреватель может его не поддерживать.

Метод `setItem` выполняет сохранение значения в хранилище:

```
<хранилище>.setItem(<имя>, <значение>)
```

Имя, под которым сохраненное значение можно будет впоследствии отыскать и прочитать, как и само *значение* указываются в виде строк. Результат этот метод не возвращает.

Пример:

```
stLocal.setItem("language", "JavaScript");
```

Для чтения сохраненного ранее значения мы применим метод `getItem`:

```
<хранилище>.getItem(<имя>)
```

Имя, опять же, указывается в виде строки. Метод возвращает извлеченное из хранилища значение в виде строки или `null`, если значения с указанным именем найти не удалось.

Пример:

```
var l = stLocal.getItem("language");
if (l) {
  // Используем полученное из хранилища значение
} else {
  // Такого значения в хранилище нет
}
```

Если понадобится удалить из хранилища сохраненное ранее значение, следует вызвать метод `removeItem`. Формат его вызова такой же, как и у метода `getItem`, а результат он не возвращает.

Пример:

```
stLocal.removeItem("language");
```

Единственное свойство объекта `Storage` — `length` — возвращает количество сохраненных в хранилище значений в виде числа.

Использование локального хранилища для временного хранения данных

Локальное хранилище HTML 5 — идеальное средство для временного хранения данных, введенных посетителем в форму. Код из листинга 3.91 демонстрирует его использование для этой цели.

Листинг 3.91. Пример временного хранения данных в хранилище

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Хранилище HTML 5</title>
    <script>
      window.addEventListener("load", function() {
        var stLocal = window.localStorage;
        var txtTitle = document.getElementById("txtTitle");
        var txtContent = document.getElementById("txtContent");
        var btnSave = document.getElementById("btnSave");
        var btnLoad = document.getElementById("btnLoad");
        btnLoad.disabled = !((stLocal.getItem("title")) &&
        (stLocal.getItem("content")));
        btnSave.addEventListener("click", function() {
          if ((txtTitle.value) && (txtContent.value)) {
            stLocal.setItem("title", txtTitle.value);
            stLocal.setItem("content", txtContent.value);
            btnLoad.disabled = false;
          }
        }, false);
        btnLoad.addEventListener("click", function() {
          txtTitle.value = stLocal.getItem("title");
          txtContent.value = stLocal.getItem("content");
        }, false);
      }, false);
    </script>
  </head>
  <body>
    <form>
      <p>Заголовок статьи<br>
      <input type="text" id="txtTitle"></p>
      <p>Содержимое статьи<br>
      <textarea id="txtContent"></textarea></p>
      <p><input type="submit" value="Отправить"></p>
```

```
      <p><input type="button" id="btnSave" value="Сохранить">
      <input type="button" id="btnLoad" value="Загрузить" disabled></p>
    </form>
  </body>
</html>
```

3.21.9. Средства геолокации

Еще DOM 3 предоставляет средства для *геолокации*, т. е. для получения местоположения клиентского компьютера — его географических координат.

Доступ к средствам геолокации

Средства геолокации доступны из свойства с "говорящим" названием `geolocation`, поддерживаемым объектом `navigator`. Они представляют собой объект класса `Geolocation`, с помощью методов которого и выполняется получение местоположения.

Пример:

```
if (navigator.geolocation) {
  var oGeo = navigator.geolocation;
  // Получаем данные геолокации
} else {
 // Web-обозреватель не поддерживает геолокацию
}
```

Получение данных геолокации

Проще всего однократно получить данные геолокации, т. е. географические координаты клиентского компьютера. Это можно сделать, обратившись к методу `getCurrentPosition` класса `Geolocation`:

```
<объект геолокации>.getCurrentPosition(<функция для получения данных>
[, <функция, вызывающаяся в случае ошибки>[, <параметры>]])
```

Единственным обязательным параметром этому методу передается функция, которая будет вызвана при успешном получении данных геолокации и собственно получит их. В качестве единственного параметра она получит объект класса `Position`, хранящий данные геолокации.

У объекта класса `Position`, полученного функцией, нас интересует свойство `coords`. Оно хранит объект класса `Coordinates`, представляющий координаты клиента и поддерживающий следующие свойства:

- `latitude` — широта в градусах;
- `longitude` — долгота в градусах;
- `altitude` — высота над уровнем моря в метрах; если значение высоты получить не удается, возвращается `null`;

- `accuracy` — точность определения широты и долготы в метрах;
- `altitudeAccuracy` — точность определения высоты в метрах; если значение высоты получить не удается, возвращается `null`;
- `heading` — направление движения, представляет собой угол в градусах, отмеренный от направления на север по часовой стрелке. Если клиентский компьютер никуда не движется, возвращает `NaN`, если значение направления получить не удается, — `null`;
- `speed` — скорость движения в метрах в секунду; если значение скорости получить не удается, возвращается `null`.

Все эти свойства возвращают числовые значения.

Пример приведен в листинге 3.92.

Листинг 3.92. Получение данных геолокации

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(function(pos) {
    var lat = pos.coords.latitude;
    var lon = pos.coords.longitude;
    var alt = pos.coords.altitude;
    var heading = pos.coords.heading;
    if (isNaN(heading)) heading = 0;
    var speed = pos.coords.speed;
  });
}
```

Обработка нештатных ситуаций

При получении данных геолокации возможны нештатные ситуации, например, ошибки в работе инструментов определения местоположения, встроенных в клиентский компьютер. При их возникновении будет выполнена функция, переданная методу `getCurrentPosition` вторым параметром.

В качестве единственного параметра эта функция получит объект класса `PositionError`, хранящий сведения о возникшей ошибке. Этот класс поддерживает два свойства:

- `code` — числовой код ошибки:
  - 1 — посетитель запретил странице доступ к данным геолокации;
  - 2 — в работе инструментов для определения местоположения, встроенных в компьютер, возникла ошибка;
  - 3 — истекло время, отведенное на определение местоположения.
- `message` — текстовое описание ошибки.

Пример приведен в листинге 3.93.

Листинг 3.93. Обработка нештатных ситуаций

```
navigator.geolocation.getCurrentPosition(function(pos) {
  // Получаем данные геолокации
}, function(error) {
  switch (error.code) {
    case 1:
      window.alert("Нет доступа к данным геолокации.");
      break;
    case 2:
      window.alert("Внутренняя ошибка.");
      break;
    case 3:
      window.alert("Истекло время, отведенное на получение данных геолокации.");
      break;
  }
});
```

Задание дополнительных параметров

При получении сведений о местоположении можно задать дополнительные параметры. Они указываются третьим параметром метода `getCurrentPosition` в виде объекта, поддерживающего перечисленные далее свойства.

- `enableHighAccuracy` — задает точность получения данных; если `true`, то устройство будет по возможности получать максимально точные данные геолокации, если `false` — данные меньшей точности. Следует помнить, что получение точных данных может потребовать больше времени и повысить энергопотребление устройства. Значение по умолчанию — `false`.
- `maximumAge` — указывает время (в виде числа в миллисекундах), в течение которого полученные данные геолокации будут кэшироваться устройством. Если задано 0, устройство не будет кэшировать данные, если `Infinity` — эти данные будут кэшироваться до тех пор, пока не устареют (пока компьютер не изменит свое местоположение). Значение по умолчанию — `Infinity`.
- `timeout` — указывает время (в виде числа в миллисекундах), отведенное на получение сведений о местоположении; по истечении этого времени будет сгенерирована ошибка, которую можно отследить. Если задано `Infinity`, то время ограничиваться не будет. Значение по умолчанию — `Infinity`.

Пример приведен в листинге 3.94.

Листинг 3.94. Пример указания дополнительных параметров `getCurrentPosition`

```
navigator.geolocation.getCurrentPosition(function(pos) {
  // Получаем данные геолокации
}, function(error) {
  // Отслеживаем нештатные ситуации
}, {
```

```
  enableHighAccuracy: true,
  maximumAge: 10000,
  timeout: 5000
});
```

Задаем получение точных координат, кэширование их в течение 10 секунд и отводим время на их получение, равное 5 секундам.

Отслеживание местоположения компьютера

Наконец, мы можем постоянно отслеживать местоположение клиентского компьютера, чтобы выяснить, куда он перемещается и достиг ли он точки назначения. Для этого нужно использовать метод `watchPosition`, формат вызова которого совпадает с таковым у метода `getCurrentPosition`.

Как только местоположение клиентского компьютера изменится, будет вызвана функция, переданная этому методу первым параметром. А при возникновении нештатной ситуации — функция, переданная вторым параметром.

Метод `watchPosition` возвращает особый идентификатор, который позволяет отменить отслеживание местоположения. Это можно сделать вызовом не возвращающего результат метода `clearWatch`:

<объект геолокации>`.clearWatch(`*<идентификатор>*`)`

В листинге 3.95 показан код, отслеживающий местоположение клиента, чтобы выяснить, достиг ли он точки назначения. Координаты точки назначения хранятся в переменных `latD` (широта) и `lonD` (долгота).

Листинг 3.95. Пример отслеживания местоположения компьютера

```
var wp = navigator.geolocation.watchPosition(function(pos) {
  if ((latD == pos.coords.latitude) && (lonD == pos.coords.longitude)) {
    navigator.geolocation.clearWatch(wp);
    window.alert("Вы в точке назначения!");
  }
}, function(error) {
  window.alert(error.message);
});
```

3.22. JavaScript-библиотеки

Мы уже не раз упоминали, что разные Web-браузеры могут по-разному выполнять код программы. По этой причине при написании приложений приходится учитывать особенности каждого Web-браузера. Проблема заключается в том, что установить все версии каждого Web-браузера на один компьютер практически невозможно, а значит, обеспечить полную кроссбраузерность самостоятельно не получится.

При использовании библиотек любой программист может сообщить о проблеме в каком-либо Web-браузере, а разработчик библиотеки, опираясь на это сообщение, имеет возможность обработать ошибку. После исправления ошибки всем остальным программистам достаточно сменить версию библиотеки. Таким образом, используя возможности какой-либо библиотеки можно забыть о проблеме с кроссбраузерностью приложения.

Наиболее часто используются следующие JavaScript-библиотеки:

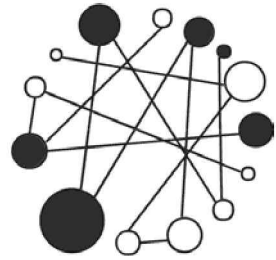
- jQuery — **http://jquery.com/**;
- Prototype — **http://www.prototypejs.org/**;
- ExtJS — **http://www.extjs.com/**;
- MooTools — **http://mootools.net/**;
- Dojo — **http://dojotoolkit.org/**;
- Yahoo! UI Library (YUI) — **http://developer.yahoo.com/yui/**.

Из этого списка хотим особо выделить библиотеку jQuery, предоставляющую функциональность, которую может использовать практически любой разработчик. Она обеспечивает кроссбраузерную поддержку приложений (работает в Internet Explorer 6.0+, Mozilla Firefox 2+, Safari 3.0+, Opera 9.0+ и Chrome), имеет небольшой размер и не засоряет глобальное пространство имен тривиальными идентификаторами. Большой популярности jQuery способствовали также дополнительные модули (их более 1500), реализующие готовые компоненты или добавляющие новую функциональность. Например, библиотека jQuery UI добавляет возможность перемещения и изменения размеров любых элементов с помощью мыши, позволяет сортировать и выделять элементы, а также предоставляет готовые компоненты ("аккордеон", панель с вкладками, диалоговые окна, календарь и др.).

Одним из авторов написана отдельная книга, полностью посвященная библиотекам jQuery и jQuery UI, а также технологии AJAX, которая позволяет обмениваться данными с сервером без перезагрузки Web-страницы. Более подробную информацию об этой книге можно получить на странице **http://wwwadmin.ru/javascript/jquery/**.

На этом мы заканчиваем знакомство с клиентскими технологиями и переходим к изучению технологий, которые выполняются на стороне сервера. Но вначале на компьютер необходимо установить специальное программное обеспечение. Какое программное обеспечение необходимо, где его найти и как установить, мы рассмотрим в следующей главе.

ГЛАВА 4



Программное обеспечение Web-сервера. Устанавливаем и настраиваем программы под Windows

4.1. Необходимые программы

Для тестирования и настройки программ необходимо установить на компьютер специальное программное обеспечение:

- *Web-сервер Apache* — программное обеспечение, отвечающее за отображение документов, запрашиваемых при наборе URL-адреса в командной строке Web-браузера;
- *Интерпретатор PHP* — для выполнения программ, написанных на языке PHP;
- *MySQL* — сервер баз данных;
- *phpMyAdmin* — набор скриптов на PHP для управления базами данных.

Все эти программы можно бесплатно получить с сайтов производителей.

Необходимо сразу заметить, что программное обеспечение мы устанавливаем только для тестирования и не задаемся целью охватить все его настройки.

> ***Внимание!***
>
> Операционная система Windows Vista и более поздние ее версии при внесении в систему ключевых изменений (задание настроек, установка программ и т. п.) выводит на экран предупреждение подсистемы UAC. На такие предупреждения всегда следует отвечать положительно, в противном случае изменения не будут внесены, и программы могут оказаться неработоспособными.

> ***Примечание***
>
> В приведенных далее инструкциях по установке указываются точные версии устанавливаемых программ. Скорее всего, со времени подготовки книги будут выпущены новые версии. В этом случае рекомендуется использовать их, особенно если номера версий отличаются только последними цифрами. Вероятно, процесс установки мало отличается от описанного в книге, однако следует иметь в виду, что незначительные отличия все же могут присутствовать.

Прежде чем устанавливать программы, необходимо проверить сетевые настройки и отсутствие программ, занимающих порты 80 и 3306, т. к. эти порты используют Web-сервер Apache и сервер MySQL. Для проверки запустим Командную строку Windows, для чего в меню **Пуск** выберем последовательно пункты **Программы (Все программы)**, **Стандартные (Служебные — Windows)** и **Командная строка**. В ее окне набираем команду:

```
ping 127.0.0.1
```

Если число потерянных пакетов больше 0, то необходимо проверить сетевые настройки. Чтобы проверить порты 80 и 3306, в командной строке набираем команду:

```
netstat -anb
```

В списке не должно быть строк с портами 80 и 3306. Если они есть, то Apache или MySQL не смогут запуститься. Обычно эти порты занимают программы Skype и Web-сервер IIS. Перед установкой и использованием Apache и MySQL указанные программы не следует запускать.

Для Skype можно запретить использовать порт 80. Откроем окно программы, выберем в меню **Инструменты** нункт **Настройки**, в расположенном слева иерархическом списке выберем нункт **Дополнительно | Соединение**, сбросим флажок **Для дополнительных входящих соединений следует использовать порты 80 и 443**, нажмем кнопку **Сохранить** и перезапустим Skype.

4.2. Установка, настройка и запуск сервера Apache

Найти дистрибутив сервера Apache можно по адресу **http://www.apachehaus.com/cgi-bin/download.plx**. В приведенном списке выбираем пункт **Apache 2.4.x VC9 | Apache 2.4.10**. В результате на наш компьютер будет загружен файл httpd-2.4.10-x86.zip размером 6,46 Мбайт.

После распаковки содержимого полученного архива будет создана папка Apache24 с множеством файлов и папок. Переименуем папку Apache24 в Apache2 и переместим в корень системного диска (C:\).

> ***Внимание!***
>
> Для успешной работы Apache 2.4.x следует установить исполняемую среду Microsoft Visual C++ 2008. Ее дистрибутив можно найти по интернет-адресу
>
> **http://www.microsoft.com/downloads/en/details.aspx?FamilyID=A5C84275-3B97-4AB7-A40D-3802B2AF5FC2**.

4.2.1. Настройка сервера Apache

Для нормальной работы необходимо изменить значения некоторых директив в конфигурационных файлах сервера Apache.

Сначала внесем изменения в главный конфигурационный файл сервера Apache — httpd.conf, находящийся в папке Apache2\conf. Откроем его в Notepad++ и отыщем следующую директиву:

```
Define SRVROOT "/Apache24"
```

Заменим ее на:

```
Define SRVROOT "c:/Apache2"
```

Следует обратить внимание на символы косой черты при указании нути к папке. Путь к папке htdocs в операционной системе Windows записывается как C:\Apache2\htdocs. А в файле конфигурации сервера Apache тот же путь будет выглядеть по-другому:

```
C:/Apache2/htdocs
```

Чтобы иметь возможность использовать файл конфигурации .htaccess, необходимо включить его поддержку.

Для этого находим раздел

```
<Directory "${SRVROOT}/htdocs">
...
</Directory>
```

Внутри раздела отыскиваем строки

```
Options Indexes FollowSymLinks
AllowOverride None
```

и заменяем их на

```
Options -Indexes +Includes +FollowSymLinks
AllowOverride All
```

Находим строки

```
<Directory />
    AllowOverride none
    Require all denied
</Directory>
```

и меняем их на

```
<Directory />
    Options -Indexes +Includes +FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

Внутри раздела `<Directory "${SRVROOT}/cgi-bin">` заменяем строку

```
AllowOverride None
```

на

```
AllowOverride All
```

Заменяем строку

```
#AddHandler cgi-script .cgi
```

на

```
AddHandler cgi-script .cgi .pl
```

Убираем символ комментария (#) перед строками

```
#LoadModule rewrite_module modules/mod_rewrite.so
#AddType text/html .shtml
#AddOutputFilter INCLUDES .shtml
#Include conf/extra/httpd-autoindex.conf
#Include conf/extra/httpd-default.conf
#Include conf/extra/httpd-languages.conf
```

Проделав все это, сохраним файл httpd.conf.

Теперь настроим сервер на работу с русским языком. Открываем файл httpd-languages.conf (расположен в папке C:\Apache2\conf\extra) и заменяем строку

```
#DefaultLanguage nl
```

на

```
DefaultLanguage ru
```

Далее находим строку

```
LanguagePriority en ca cs da de el eo es et fr he hr it ja ko ltz nl nn no pl
pt pt-BR ru sv tr zh-CN zh-TW
```

и ставим русский язык (ru) на первое место:

```
LanguagePriority ru en ca cs da de el eo es et fr he hr it ja ko ltz nl nn no
pl pt pt-BR sv tr zh-CN zh-TW
```

В конец файла добавляем строку

```
AddDefaultCharset windows-1251
```

если собираемся создавать страницы на HTML 4, или

```
AddDefaultCharset UTF-8
```

если будем использовать HTML 5.

Сохраняем и закрываем файл httpd-languages.conf.

Последнее, что нам нужно сделать, — добавить Apache в список исключений Брандмауэра Windows. В меню **Пуск** выбираем опцию **Пастройка | Папель управлеппя** и пункт **Брапдмауэр Windows**.

- Если мы пользуемся Windows XP, то в открывшемся окне выбираем вкладку **Исключеппя**. Если в списке нет пункта **Apache ПТТР Server**, нажимаем кнопку **Добавпть программу**. В открывшемся окне нажимаем кнопку **Обзор**. Находим файл httpd.exe (C:\Apache2\bin\httpd.exe) и нажимаем кнопку **Открыть**, а затем **OK**. Теперь убедимся, что флажок напротив пункта **Apache ПТТР Server** установлен. Нажимаем **OK** для выхода из окна свойств Брандмауэра Windows.

- Если мы пользуемся Windows Vista или более новой версией этой системы, щелкнем в появившемся окне гиперссылку **Разрешение взаимодействия с приложением или компонентом в брандмауэре Windows**. Далее сразу же нажмем кнопку **Изменить параметры**. Нажмем кнопку **Разрешить другое приложение**, после чего — кнопку **Обзор**, выберем в появившемся далее окне файл httpd.exe (C:\Apache2\bin\httpd.exe), нажмем кнопки **Открыть** и **Добавить**. Убедимся, что флажок, находящийся в столбце **Частная** нункта **Apache HTTP Server**, установлен, нажмем кнопку **OK** и закроем окно брандмауэра.

4.2.2. Запуск и останов Apache

Проще всего занустить Apache как консольное приложение Windows. Для этого откроем Командную строку и наберем в ней такую последовательность команд:

```
cd \
cd Apache2\bin
httpd
```

Если мы не донустили ошибок в файлах конфигурации, сервер будет запущен.

> ***Внимание!***
> Не закрывайте окно Командной строки, в котором запущен Apache. Иначе сервер будет остановлен.

Для проверки открываем Web-браузер и в адресной строке набираем:

```
http://localhost/
```

Нажимаем клавишу <Enter>. Если сервер установлен правильно, то в окне Web-браузера отобразится стартовая страница сервера Apache (рис. 4.1).

Чтобы остановить Apache, достаточно перейти в то окно Командной строки, где он был занущен, и нажать комбинацию клавиш <Ctrl>+<C>. Сервер остановится через несколько секунд, и в окне вновь появится приглашение для ввода команд.

4.2.3. Установка Apache как службы Windows

Однако во многих случаях будет удобнее установить Apache как службу Windows. Сделать это очень просто.

Откроем Командную строку. Если мы используем Windows Vista или более позднюю ее версию, запустим Командную строку с повышенными правами, для чего щелкнем на ее ярлыке правой кнопкой мыши и выберем в контекстном меню пункт **Запуск от имени администратора**.

В Командной строке наберем следующую последовательность команд:

```
cd \
cd apache2\bin
httpd -k install
```

Через несколько секунд установка будет завершена.

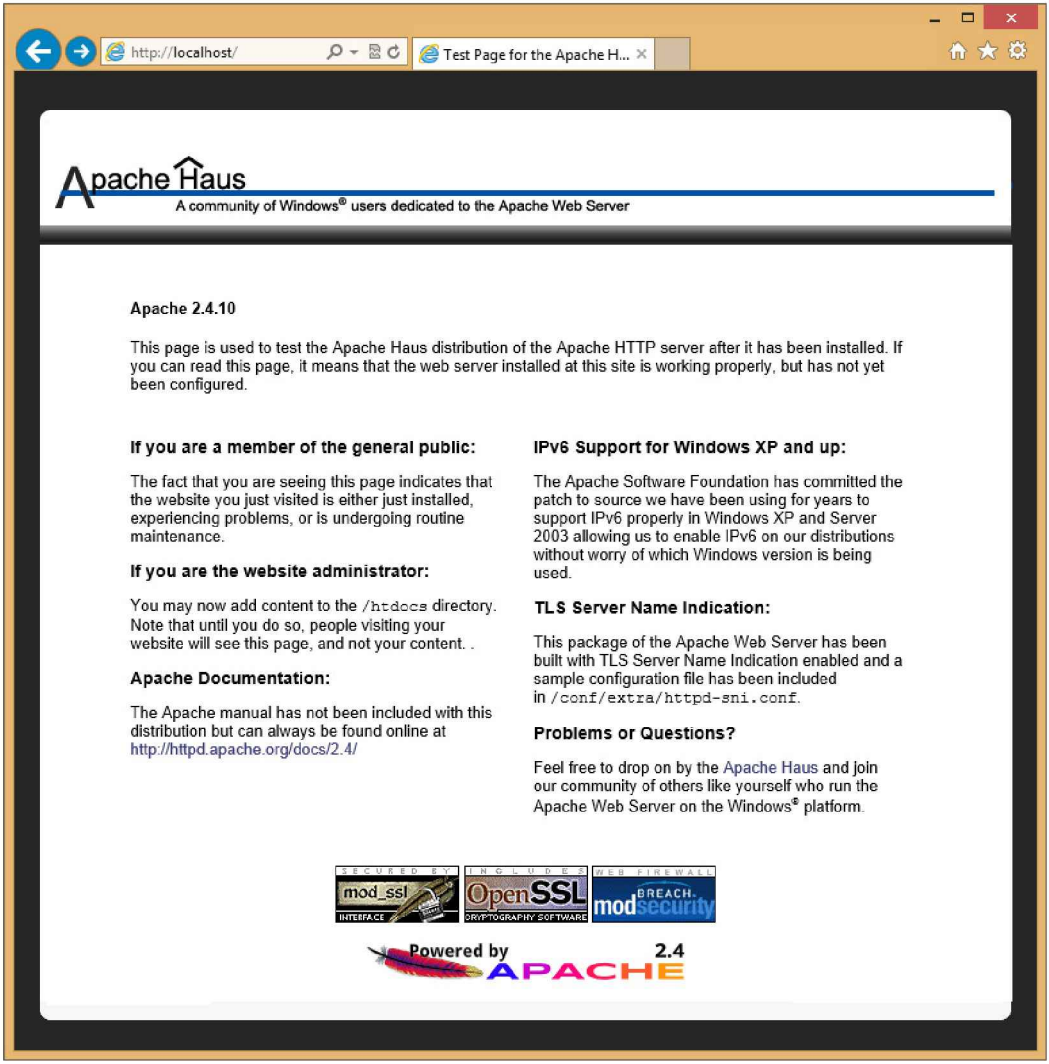


Рис. 4.1. Стартовая страница сервера Apache

По умолчанию все службы Windows работают под учетной записью **Локальная система**. Эта учетная запись имеет минимальные права, в частности, не может получить доступ к файловой системе. И, следовательно, установленный как служба Apache не сможет загрузить ни один файл, находящийся в папке C:\Apache2\bin.

Выходом может быть запуск службы Apache от имени того же пользователя, под которым мы работаем. Нам нужно лишь указать учетную запись этого пользователя в параметрах службы.

Выберем в меню **Пуск** пункт **Пастройка | Панель управления**, в открывшемся окне — пункт **Администрирование**, а затем **Службы**. В списке служб найдем пункт **Apache2.4**, щелкнем на нем правой кнопкой мыши и в контекстном меню выберем пункт **Свойства**. Переключимся на вкладку **Вход в систему**.

Установим переключатель **С учетной запиcью** и нажмем сначала кнопку **Обзор**, а затем кнопки **Дополпптельпо** и **Попск**. В списке **Результаты попска** выберем нункт, представляющий "нашего" пользователя, и дважды нажмем **OK**. В полях ввода **Пароль** и **Подтверждеппе** занесем пароль этого пользователя, после чего нажмем **OK**.

Теперь нам нужно дать "нашему" пользователю права на работу в качестве службы и части операционной системы. Для этого выберем в окне Панели управления нункт **Адмпппстрпроваппе**, а потом — **Локальпая полптика безопасиости**. В иерархическом списке, расположенном в левой части появившегося на экране окна, выберем нункт **Локальпые полптики | Пазпачеппе прав иользователя**. В правой части появится список с набором пунктов, представляющих различные низкоуровневые права пользователя.

Сначала отыщем нункт **Вход в качестве службы** и посмотрим, присутствует ли "наш" пользователь в перечне, находящемся в столбце **Параметр безопаспостп** этого нункта. Если его там нет, дважды щелкнем на пункте и, когда на экране появится окно задания параметров, нажмем кнопку **Добавпть пользователя или группу**. Далее последовательно нажимаем кнопки **Дополпительпо** и **Попск**, в списке **Результаты иоиска** выбираем нункт, представляющий нужного пользователя, и трижды нажимаем кнопку **OK**.

После этого найдем пункт **Работа в режиме оиерациоипой системы** и также проверяем, присутствует ли "наш" пользователь в перечне. Скорее всего, перечень будет пуст, и нам придется добавить в него нужного пользователя, согласно описанным ранее инструкциям.

Не забудем перезапустить службу Apache. Это можно сделать, переключившись в окно списка служб и воспользовавшись кнопкой в панели инструментов или пунктами контекстного меню.

Удалить Apache из списка установленных в системе служб можно, открыв Командную строку (пользователям Windows Vista и последующих версий нужно открыть ее с повышенными правами) и набрав в ней:

```
cd \
cd apache2\bin
httpd -k uninstall
```

4.3. Структура каталогов сервера Apache

Итак, сервер установлен и запущен. Теперь давайте рассмотрим каталоги сервера Apache, их содержание и назначение. В папке C:\Apache2\ находятся следующие каталоги:

- bin — здесь располагается главный исполняемый файл сервера (httpd.exe) и исполняемые файлы вспомогательных утилит;
- cgi-bin — каталог для CGI-программ (программ, написанных на языках Perl, C и т. д.);

- conf — папка, где находятся конфигурационный файл сервера (httpd.conf) и другие файлы конфигурации сервера Apache;
- error — каталог для файлов с сообщениями об ошибках (например, если запрашиваемый файл не найден);
- htdocs — папка, в которой должны располагаться файлы в форматах HTML и PHP, а также другие файлы, которые будут доступны при наборе в адресной строке Web-браузера **http://localhost/** (например, изображения, каскадные таблицы стилей и т. д.).

  С этим каталогом мы будем работать постоянно. Поэтому удобно добавить ярлык к нему на Рабочий стол. Для этого щелкаем на названии каталога правой кнопкой мыши. В контекстном меню выбираем пункт **Отправить**. В появившемся подменю выбираем пункт **Рабочий стол (создать ярлык)**.

  После установки сервера на компьютер в каталоге htdocs находится страница с приветствием (см. рис. 4.1), которое мы видим при наборе в командной строке Web-браузера **http://localhost/**, файл таблицы стилей и десяток графических файлов. Удалим их. Создадим любой HTML-документ и сохраним его в каталоге htdocs под названием index.html. Теперь при наборе **http://localhost/** мы должны увидеть содержимое сохраненного HTML-документа, а не стандартное приветствие сервера;
- icons — здесь содержится ряд изображений, используемых в листингах каталогов;
- include — набор заголовочных файлов, необходимых для разработки дополнительных модулей (нам они не понадобятся);
- lib — набор библиотечных файлов, предназначенных для разработки дополнительных модулей (они также нам не пужны);
- logs — в этой папке находятся журналы регистрации посещений (access.log) и ошибок (error.log), позволяющие получить подробную информацию обо всех запросах и ошибках. Открыть эти файлы можно с помощью любого текстового редактора (например, Notepad++);
- modules — этот каталог содержит подключаемые модули.

4.4. Файл конфигурации httpd.conf

Файл httpd.conf (C:\Apache2\conf\httpd.conf) — это основной файл конфигурации сервера. Открыть и отредактировать файл можно с помощью любого текстового редактора, например, Блокнота или Notepad++. После каждого изменения в файле конфигурации необходимо перезагрузить сервер, остановив его и запустив снова. До перезагрузки он будет работать со старыми параметрами.

4.4.1. Основные понятия

В файле httpd.conf содержатся директивы, влияющие на работу сервера Apache. Директива представляет собой ключевое слово, за которым следует одно или не-

сколько значений. Директивы бывают простыми (изменяющие только одно свойство сервера), а могут объединяться в разделы (позволяют изменять сразу несколько свойств какого-нибудь объекта).

Если в начале строки указан символ #, то такая строка является комментарием:

```
# ServerAdmin: Your address, where problems with the server should be
# e-mailed.  This address appears on some server-generated pages, such
# as error documents.  e.g. admin@your-domain.com
ServerAdmin admin@example.com
```

В этом примере первые три строки закомментированы, а четвертая с помощью директивы `ServerAdmin` задает электронный адрес администратора сервера.

Вставлять комментарий в середину строки нельзя.

4.4.2. Разделы файла конфигурации

Директивы могут объединяться в разделы, что позволяет ограничить область действия директив отдельным каталогом, набором файлов или набором URL. Существуют следующие разделы:

- `Directory` и `DirectoryMatch` — указывают, что директивы применимы к заданному каталогу и всем подкаталогам:

  ```
  <Directory "C:/Apache2/htdocs">
     Options -Indexes
  </Directory>
  ```

  `DirectoryMatch` позволяет использовать регулярные выражения;

- `Files` и `FilesMatch` — указывают, что директивы применимы только к определенным файлам. Символ `*` соответствует любой последовательности символов, а символ `?` — любому одиночному символу.

- В качестве примера запретим доступ к текстовым файлам:

  ```
  <Files *.txt>
     Require all denied
  </Files>
  ```

  `FilesMatch` позволяет использовать регулярные выражения;

- `IfModule` — указывает, что директивы будут задействованы лишь при загрузке указанного модуля:

  ```
  <IfModule dir_module>
     DirectoryIndex index.html
  </IfModule>
  ```

- `Limit` и `LimitExcept`. `Limit` — указывает, что директивы будут использоваться, только когда HTTP-запрос выполнен с помощью одного из указанных методов (`GET`, `POST` или `HEAD`). `LimitExcept`, наоборот, разрешает доступ для методов, которые не указаны;

```
<Limit GET POST OPTIONS PROPFIND>
   Require all granted
</Limit>
```

- `Location` и `LocationMatch` — указывают, что заключенные в них директивы действуют лишь в случае обращения с указанного интернет-адреса:

```
<Location /server-status>
   SetHandler server-status
   Require host localhost
</Location>
```

`LocationMatch` позволяет использовать регулярные выражения;

- `VirtualHost` — указывает, что директивы применимы только к документам указанного виртуального хоста. Применяется, когда сервер обслуживает множество Web-сайтов с разными именами хостов.

```
<VirtualHost 192.168.0.1:80>
   ServerAdmin webmaster@site.ru
   DocumentRoot /www/docs/site.ru
   ServerName site.ru
</VirtualHost>
```

4.4.3. Общие директивы. Создание домашнего каталога пользователя, доступного при запросе http://localhost/~nik/

Перечислим основные общие директивы сервера Apache:

- `ServerName` — определяет имя сервера:

```
ServerName localhost:80
```

- `ServerAdmin` — задает E-mail администратора сервера:

```
ServerAdmin admin@mailserver.ru
```

- `ServerRoot` — указывает местонахождение каталогов сервера:

```
ServerRoot "C:/Apache2"
```

- `DocumentRoot` — определяет местонахождение корневого каталога для документов на сервере:

```
DocumentRoot "C:/Apache2/htdocs"
```

- `UserDir` — задает имя каталога, в котором ищутся домашние каталоги пользователей при получении запроса вида **http://localhost/~user/**:

```
UserDir "c:/My Website"
```

Создадим каталог для пользователя nik. Для этого добавим в C:\Apache2 папку user. В папке user создаем папку nik, в которую добавляем файл index.html со следующим содержанием:

```
<html>
<head><title>Страничка пользователя Nik</title></head>
<body>Привет всем</body>
</html>
```

Далее с помощью Блокнота открываем файл httpd-userdir.conf (который находится в папке C:\Apache2\conf\extra) и изменяем значение директивы `UserDir` на

```
UserDir "C:/Apache2/user"
```

Находим строку

```
<Directory "C:/Documents and Settings/*/My Documents/My Website">
```

и заменяем ее на

```
<Directory "C:/Apache2/user">
```

Сохраняем и закрываем файл. Теперь файл httpd-userdir.conf необходимо подключить к основному конфигурационному файлу. Открываем файл httpd.conf и убираем символ комментария (`#`) перед строками

```
#Include conf/extra/httpd-userdir.conf
#LoadModule userdir_module modules/mod_userdir.so
```

Сохраняем и закрываем файл httpd.conf. Перезапускаем сервер Apache. Далее открываем Web-браузер и в адресной строке набираем **http://localhost/~nik/**. В итоге в окне Web-браузера должна отобразиться надпись "Привет всем";

- `PidFile` — указывает местоположение файла, в котором будет регистрироваться исходный процесс сервера:

```
PidFile logs/httpd.pid
```

- `Listen` — связывает Apache с определенным портом и (или) IP-адресом:

```
Listen 80
Listen 12.34.56.78:80
```

- `Options` — позволяет включить или отключить те или иные опции в различных частях сайта. Если опция помечена знаком "+", то она добавляется к числу уже включенных опций, а если знаком "–", то опция отключается. Могут быть заданы следующие опции:

  - `All` — включает все опции, кроме `MultiViews`:

```
Options All
```

  - `None` — отключает все опции, кроме `MultiViews`:

```
Options None
```

  - `ExecCGI` — позволяет выполнять CGI-программы в каталоге, отличном от указанного в директиве `ScriptAlias`, например, в каталоге с обычными документами. Для правильной работы необходимо указать директиву `AddHandler` или `SetHandler`:

```
<Directory "C:/Apache2/htdocs">
   Options +ExecCGI
   SetHandler cgi-script
</Directory>
```

- `FollowSymLinks` — разрешает использование символических ссылок:

  ```
  Options +FollowSymLinks
  ```

- `SymLinksIfOwnerMatch` — разрешает символические ссылки, если ссылка указывает на объект, который принадлежит тому же пользователю, что и ссылка:

  ```
  Options +SymLinksIfOwnerMatch
  ```

- `Includes` — разрешает использование серверных расширений (SSI):

  ```
  Options +Includes
  ```

- `IncludesNOEXEC` — разрешает серверные расширения, но запрещает команду `#exec` и директиву `#include` для загрузки CGI-программ:

  ```
  Options +IncludesNOEXEC
  ```

- `Indexes` — если эта опция включена и заданный по умолчанию файл не найден, то сервер генерирует листинг файлов. Если опция выключена, то вместо файла отображается сообщение об ошибке 403.

  ```
  <Directory "C:/Apache2/htdocs">
     Options -Indexes
  </Directory>
  ```

  На виртуальном хостинге эта опция должна быть обязательно выключена, иначе пользователь будет видеть все содержимое каталога, в том числе и файлы паролей;

- `MultiViews` — включает content-соответствие — средство, с помощью которого сервер определяет, какой документ наиболее приемлем для посетителя:

  ```
  Options +MultiViews
  ```

4.4.4. Переменные сервера и их использование

Очень часто бывает необходимо записать какое-либо значение, например путь к папке, сразу в нескольких местах файла конфигурации. В таком случае мы можем объявить *переменную сервера*, сохранить в ней это значение, после чего обратиться к этой переменной по ее имени.

Для объявления переменной сервера предусмотрена директива `Define`:

```
Define <имя переменной> <значение переменной>
```

В имени переменной допустимы лишь символы латиницы и цифры, причем начинаться имя должно с буквы. Обычно имена переменных набирают прописными буквами.

Пример:

```
Define SRVROOT "c:/Apache2"
```

Объявляем переменную `SRVROOT` со значением "c:/Apache2" (путь к папке, где установлен Apache).

Использовать объявленную переменную мы можем, вставив ссылку на нее в формате `${`*<имя переменной>*`}`.

Примеры:

```
ServerRoot "${SRVROOT}"
```

Указываем в качестве значения директивы `ServerRoot` значение объявленной ранее переменной `SRVROOT`.

```
<Directory "${SRVROOT}/htdocs">
. . .
</Directory>
```

А здесь мы используем значение переменной `SRVROOT` для указания части нути к папке с файлами сайта htdocs в разделе `Directory`.

4.4.5. Директивы управления производительностью

При увеличении нагрузки на сервер создаются новые процессы, а при уменьшении — эти процессы закрываются. Частые запуски и остановки порожденных процессов снижают производительность сервера.

Поэтому необходимо правильно настроить следующие директивы:

- `StartServers` — число копий процесса сервера, которые будут созданы при запуске сервера;
- `MinSpareServers` — минимальное число порожденных процессов;
- `MaxSpareServers` — максимальное число порожденных процессов;
- `MaxRequestWorkers` — максимальное число возможных подключений к серверу.

Указанные директивы не применимы к платформе Windows. Их нужно заменять на `StartThreads`, `MinSpareThreads`, `MaxSpareThreads` и `MaxThreads`. Также применяются следующие директивы:

- `ThreadsPerChild` — задает максимальное число потоков, порождаемых каждым дочерним процессом сервера Apache:

  ```
  ThreadsPerChild 250
  ```

- `MaxConnectionsPerChild` — определяет, сколько запросов может обработать порожденный процесс за время его существования. Для снятия ограничений необходимо указать 0. На платформе Windows директива всегда должна иметь значение 0:

  ```
  MaxConnectionsPerChild 0
  ```

4.4.6. Директивы обеспечения постоянного соединения

За обеспечение постоянного соединения отвечают следующие директивы:

- `Timeout` — задает промежуток времени в секундах, в течение которого сервер продолжает попытки возобновления приостановленной передачи данных:

```
Timeout 300
```

- `KeepAlive` — разрешает постоянные соединения:

```
KeepAlive On
```

- `MaxKeepAliveRequests` — ограничивает число допустимых запросов на одно соединение:

```
MaxKeepAliveRequests 100
```

  Для снятия ограничений необходимо указать 0;

- `KeepAliveTimeout` — определяет тайм-аут для постоянного соединения:

```
KeepAliveTimeout 15
```

4.4.7. Директивы работы с языками

Для работы с языками предназначены следующие директивы:

- `AddDefaultCharset` — указывает кодовую таблицу для документов по умолчанию:

```
AddDefaultCharset windows-1251
```

- `AddCharset` — устанавливает взаимосвязь между кодовой таблицей символов и расширением файла:

```
AddCharset ISO-2022-JP .jis
```

- `RemoveCharset` — удаляет взаимосвязь между кодовой таблицей символов и расширением файла:

```
RemoveCharset .jis
```

- `AddLanguage` — устанавливает взаимосвязь между языком и расширением файла:

```
AddLanguage ru .ru
```

- `RemoveLanguage` — удаляет все взаимосвязи между языками и расширениями файла:

```
RemoveLanguage .ru
```

- `DefaultLanguage` — определяет, какой язык должен быть указан в заголовке, если для расширения файла не задан определенный язык:

```
DefaultLanguage ru
```

- `LanguagePriority` — задает приоритет различных языков:

```
LanguagePriority ru en ca cs da de el
```

4.4.8. Директивы перенаправления

Перечислим основные директивы перенаправления:

- `Alias` и `AliasMatch` — позволяют предоставить доступ не только к файлам, находящимся в каталоге, указанном в директиве `DocumentRoot`, но и к другим каталогам сервера. В директиве `AliasMatch` можно использовать регулярные выражения:

```
AliasMatch ^/manual(?:/(?:de|en|es|ru))?(/.*)?$ "C:/Apache2
/manual$1"
```

- `ScriptAlias` и `ScriptAliasMatch` — задают местоположение каталога для CGI-сценариев:

```
ScriptAlias /cgi-bin/ "C:/Apache2/cgi-bin/"
```

  Директива `ScriptAliasMatch` позволяет использовать регулярные выражения;

- `Redirect` и `RedirectMatch` — сообщают, что искомый документ больше не находится в данном месте, и указывают, где можно его найти. Директива `RedirectMatch` позволяет использовать регулярные выражения. Имеют дополнительный параметр, указывающий состояние переадресации, который может принимать следующие значения:
  - `permanent` — ресурс перемещен навсегда (код 301);
  - `temp` — ресурс перемещен временно (код 302);
  - `seeother` — ресурс был заменен другим ресурсом (код 303);
  - `gone` — ресурс удален навсегда (код 410).

  Пример:

```
Redirect permanent /file1.html /file2.html
RedirectMatch 301 ^/manual(?:/(de|en|es|ru)){2,}(/.*)?$
/manual/$1$2
```

- `RedirectPermanent` — выполняет перенаправление с состоянием `permanent`:

```
RedirectPermanent /file1.html /file2.html
```

- `RedirectTemp` — выполняет перенаправление с состоянием `temp`:

```
RedirectTemp /file1.html /temp_file.html
```

4.4.9. Обработка ошибок

С помощью директивы `ErrorDocument` можно указать документ, который будет выдан Web-браузеру при возникновении указанной ошибки:

```
ErrorDocument 404 /err/error404.html
```

Обычно указываются директивы (и разрабатываются соответствующие документы) для следующих ошибок:

- `401` — пользователь не авторизован;
- `403` — нет доступа. При отсутствии индексного файла в каталоге и отключенной опции `Indexes` директивы `Options` генерируется именно эта ошибка;
- `404` — ресурс не найден.

4.4.10. Настройки MIME-типов

При передаче файла сервер указывает MIME-тип документа. Это позволяет Web-браузеру правильно обработать получаемый файл. MIME-тип указывается в формате:

```
<Категория>/<Тип файла>
```

Примеры:

- `text/html` — для HTML-документов;
- `image/gif` — для изображений в формате GIF;
- `application/msword` — для документов в формате Word;
- `audio/mpeg` — для аудиофайлов MP3.

Конфигурации MIME-типов находятся в файле mime.types (C:\Apache2\conf\mime.types). Для настройки MIME-типов и смежных вопросов используются следующие директивы:

- `AddEncoding` — устанавливает взаимосвязь между определенной кодировкой и расширением файла:

  ```
  AddEncoding pkzip .zip
  ```

- `RemoveEncoding` — удаляет взаимосвязь между определенной кодировкой и расширением файла:

  ```
  RemoveEncoding .zip
  ```

- `TypesConfig` — указывает расположение конфигурационного файла с настройками MIME-типов:

  ```
  TypesConfig conf/mime.types
  ```

- `AddType` — позволяет добавить новый MIME-тип и связать его с определенным расширением:

  ```
  AddType application/x-httpd-php .php
  ```

- `RemoveType` — удаляет связи между MIME-типами и расширениями:

  ```
  RemoveType .cgi
  ```

- `ForceType` — указывает MIME-тип для набора файлов. Присваивает файлам, указанным в разделе `<Directory>` или `<Files>`, определенный MIME-тип, не принимая во внимание расширения файлов;
- `AddHandler` — связывает определенный обработчик с файловым расширением:

  ```
  AddHandler type-map .var
  ```

- `SetHandler` — обеспечивает обработку файлов в разделах `<Directory>` или `<Files>` с помощью определенного обработчика:

```
<Files *.html>
   SetHandler type-map
</Files>
```

- `RemoveHandler` — отменяет связывание определенного обработчика с файловым расширением:

```
AddHandler server-parsed .html
RemoveHandler .html
```

В директивах `AddHandler` и `SetHandler` могут быть указаны следующие обработчики:

- `default-handler` — обработчик по умолчанию, который используется для обслуживания HTML-документов, файлов изображений (т. е. файлов, не требующих предварительной обработки);
- `send-as-is` — посылает файл, содержащий в себе HTTP-заголовки, как есть (без добавления пакетных или HTTP-заголовков). Заголовки можно указывать в самом файле, отделяя их от основного содержимого нустой строкой;
- `cgi-script` — обрабатывает файл как CGI-скрипт;
- `imap-file` — обрабатывает файл как карту-изображение;
- `server-info` — возвращает конфигурационную информацию сервера. Необходимо, чтобы был подключен модуль mod_info.so:

```
<Location /info>
   SetHandler server-info
</Location>
```

- `server-status` — возвращает отчет о состоянии сервера. Необходимо, чтобы был подключен модуль mod_status.so:

```
<Location /status>
   SetHandler server-status
</Location>
```

- `type-map` — обрабатывает файл как файл сопоставления типов:

```
AddHandler type-map .var
```

В этом примере все файлы с расширением var будут использоваться как файлы сопоставления типов. Пример файла сопоставления типов:

```
URI: index.html.en
Content-Language: en
Content-type: text/html; charset=ISO-8859-1
URI: index.html.ru.koi8-r
Content-Language: ru
Content-type: text/html; charset=KOI8-R
```

- `Action` — устанавливает соответствие между заданным названием обработчика или MIME-типа с определенной программой, обеспечивающей механизм исполнения. Данная директива позволяет создавать собственные обработчики:

```
Action image/gif /cgi-bin/images.cgi

Action my-file-type /cgi-bin/program.cgi
AddHandler my-file-type .xyz
```

- `CacheNegotiatedDocs` — задает режим кэширования сервером результатов переговоров; если директива имеет значение `on`, то документы, установленные в результате переговоров между сервером и Web-браузером о согласовании MIME-типа, языка и способа кодирования, могут быть помещены в кэш:

```
CacheNegotiatedDocs on
```

  По умолчанию директива имеет значение `off`.

4.4.11. Управление листингом каталога

Управлять отображением листинга каталога позволяют следующие директивы:

- `DirectoryIndex` — задает название документа, который будет возвращен по запросу, если не указано название документа (например, **http://localhost/**):

```
DirectoryIndex index.php index.html
```

  Значение `disabled` отключает эту функцию.

- `IndexOptions` — определяет способ генерирования листинга каталога с помощью опций. Если опция помечена знаком "+", то она добавляется к числу уже включенных опций, а если знаком "–", то она отключается. Для использования этой директивы необходимо, чтобы опция `Indexes` директивы `Options` была включена. Могут быть указаны следующие опции:

  - `DescriptionWidth` — задает ширину столбца описания в символах. Если указан знак `*`, то ширина столбца станет равной ширине самого длинного описания:

```
IndexOptions +DescriptionWidth=30
IndexOptions +DescriptionWidth=*
```

  - `FancyIndexing` — включает режим, в котором листинг каталога будет иметь интерфейс, напоминающий диспетчер файлов;
  - `FoldersFirst` — устанавливает, что вначале отображаются названия папок, а затем названия файлов;
  - `HTMLTable` — заставляет оформлять листинг каталога как HTML-таблицу в заданном формате, а не как список;
  - `IconsAreLinks` — инструктирует сделать пиктограммы ссылками;

- `IconHeight` и `IconWidth` — задают размер пиктограмм, отображаемых в листинге каталога (по умолчанию 20×22 пикселов):

```
IndexOptions +IconHeight=20 +IconWidth=22
```

- `IgnoreCase` — позволяет игнорировать регистр символов;
- `IgnoreClient` — отключает пересортировку листинга файлов по столбцам;
- `NameWidth` — устанавливает максимальную длину имени файла, отображаемую в листинге. Если указан знак `*`, то используется длина самого длинного имени файла;
- `ScanHTMLTitles` — инструктирует отображать в описании файла информацию из тега `<title>`;
- `SuppressColumnSorting` — отключает сортировку листинга файлов по столбцам;
- `SuppressDescription` — удаляет столбец с описанием файлов;
- `SuppressHTMLPreamble` — удаляет стандартные открывающие и закрывающие теги (`<html>` и `<body>`). Применяется, если заданы директивы `HeaderName` и `ReadmeName`. Указанные этими директивами файлы должны иметь открывающие теги (для файла, указанного в `HeaderName`) и закрывающие (для файла, указанного в `ReadmeName`);
- `SuppressIcon` — выключает отображение пиктограмм в листинге каталога;
- `SuppressLastModified` — удаляет столбец с датой и временем последнего обновления файла;
- `SuppressRules` — отключает вывод разделительных линий сверху и снизу листинга;
- `SuppressSize` — удаляет столбец с размерами файлов;
- `TrackModified` — включает кэширование листинга каталога;
- `VersionSort` — устанавливает режим сортировки файлов с учетом номера версии;

- `AddIcon` — задает пиктограмму для названия файла или его части (например, расширения):

```
AddIcon /icons/binary.gif .bin .exe
```

- `AddIconByType` — задает пиктограмму для MIME-типов:

```
AddIconByType (TXT,/icons/text.gif) text/*
```

- `DefaultIcon` — устанавливает пиктограмму, используемую по умолчанию:

```
DefaultIcon /icons/unknown.gif
```

- `AddIconByEncoding` — связывает пиктограмму с типом кодировки:

```
AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
```

- `AddDescription` — устанавливает описание для файла или набора файлов, соответствующих шаблону:

  ```
  AddDescription "Описание файла" name.html
  ```

  Описание отображается в столбце **Описание** листинга каталога. Оно может включать HTML-форматирование;

- `HeaderName` — позволяет изменить стандартный заголовок листинга каталога:

  ```
  HeaderName HEADER.html
  ```

  Если указана опция `SuppressHTMLPreamble`, то содержимое файла заменит весь верхний колонтитул;

- `ReadmeName` — позволяет изменить стандартный нижний колонтитул листинга каталога:

  ```
  ReadmeName README.html
  ```

  Если указана опция `SuppressHTMLPreamble`, то содержимое файла заменит весь нижний колонтитул;

- `IndexIgnore` — служит для указания файлов, которые не должны быть показаны в листинге каталога:

  ```
  IndexIgnore HEADER* README* .htaccess
  ```

- `IndexOrderDefault` — позволяет изменить первоначальную сортировку листинга каталога (по умолчанию файлы сортируются по имени). Первый аргумент задает порядок сортировки. Может принимать два значения: `Ascending` (по возрастанию) и `Descending` (по убыванию). Второй аргумент задает имя поля: `Name`, `Date`, `Size` или `Description`:

  ```
  IndexOrderDefault Descending Date
  ```

- `IndexStyleSheet` — указывает таблицу стилей, задающую оформление для сгенерированной сервером страницы списка файлов:

  ```
  IndexStyleSheet "/css/list.css"
  ```

- `IndexHeadInsert` — задает дополнительный HTML-код, который будет вставлен в тег `<head>` сгенерированной страницы со списком файлов:

  ```
  IndexHeadInsert "<link rel=\"shortcut icon\" ↳
  href=\"/icons/favicon.ico\">"
  ```

4.4.12. Директивы протоколирования

Как уже упоминалось, события, происходящие на сервере, регистрируются Apache в журналах. По умолчанию в каталоге logs (C:\Apache2\logs) расположены два файла журналов: access.log и error.log. Эти журналы позволяют получить подробную информацию обо всех запросах и ошибках. Открыть эти файлы можно с помощью любого текстового редактора (например, Notepad++).

Файл access.log содержит следующую информацию: IP-адрес, дату и время запроса, метод (`GET` или `POST`), имя запрошенного файла, протокол, код состояния запроса (код 200 означает, что файл успешно найден, а 404 — означает, что файл не найден) и размер файла. Кроме того, файл может содержать информацию о ссылающейся странице (с которой перешел пользователь на наш сайт с другого сайта), а также информацию о Web-браузере посетителя. Пример строки журнала:

```
127.0.0.1 — - [25/May/2008:22:34:24 +0400] "GET /test.php HTTP/1.1" 200 59
```

Файл error.log содержит информацию об ошибке: дату и время запроса, IP-адрес, информацию об ошибке. Кроме того, файл может содержать информацию о ссылающейся странице (на которой была ошибочная ссылка на наш сайт), а также информацию о Web-браузере посетителя:

```
[Sun May 25 22:34:24 2008] [error] [client 127.0.0.1] File does not exist:
C:/Apache2/htdocs/m
```

Запись об ошибке дублируется и в файле access.log:

```
127.0.0.1 — - [25/May/2008:22:34:24 +0400] "GET /m HTTP/1.1" 404 283
```

Местоположение и формат журналов задаются с помощью следующих директив:

- `CustomLog` — указывает, где расположен журнал регистрации, а также его формат:

  ```
  CustomLog logs/access.log common
  ```

- `LogFormat` — определяет фактический формат журнала регистрации. Псевдоним формата (`common`) указывается в директиве `CustomLog`:

  ```
  LogFormat "%h %l %u %t \"%r\" %>s %b" common
  ```

  В строке формата могут присутствовать следующие символы, которые заменяются фактическими значениями:

  - `%h` — адрес удаленного хоста (адрес клиента, сделавшего запрос);
  - `%l` — удаленное имя пользователя. Практически всегда содержит прочерк;
  - `%u` — имя пользователя, прошедшего аутентификацию;
  - `%t` — дата и время запроса;
  - `%r` — метод, имя запрошенного ресурса и протокол;
  - `%>s` — статус запроса;
  - `%b` — число отправленных байтов;
  - `%{Referer}i` — страница, с которой пришел клиент;
  - `%{User-Agent}i` — Web-браузер, используемый клиентом.

  Существуют и другие переменные директивы `LogFormat`, но они встречаются крайне редко, т. к. программы обработки log-файлов настроены на форматы `common` и `combined`. С помощью этих программ можно получить статистические данные в более удобном формате;

- `ErrorLog` — определяет местоположение журнала регистрации ошибок:

```
ErrorLog logs/error.log
```

- `LogLevel` — позволяет установить уровень регистрации ошибок и диагностических сообщений в журнале error.log. По умолчанию директива настроена на регистрацию аварийных ситуаций (`warn`). Могут быть заданы следующие значения: `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert` или `emerg`:

```
LogLevel warn
```

- `HostnameLookups` — если директива имеет значение `On`, то Apache будет регистрировать полное имя хоста клиента, а не только IP-адрес. Значение по умолчанию:

```
HostnameLookups Off
```

4.4.13. Файл конфигурации .htaccess. Управляем сервером Apache из обычной папки

На виртуальном хостинге не предоставляется доступ к главному файлу конфигурации, т. к. один сервер может обслуживать множество сайтов, принадлежащих различным людям. В этом случае для конфигурирования отдельных каталогов используется файл .htaccess. При изменении этого файла нет необходимости перезагружать сервер. Файлы .htaccess анализируются при каждом запросе файла из каталога.

Если сервер находится в полном нашем распоряжении, то настраивать конфигурацию необходимо в файле httpd.conf, а использование файлов .htaccess нужно запретить, поскольку это сильно влияет на производительность и защиту. Файл httpd.conf анализируется только один раз (при запуске сервера), а файлы .htaccess анализируются при каждом запросе. Если использование файлов .htaccess запрещено, то Apache даже не будет искать эти файлы в каталогах.

Для настройки файлов .htaccess предназначены следующие директивы:

- `AccessFileName` — задает имя файла конфигурации:

```
AccessFileName .htaccess
```

- `AllowOverride` — позволяет ограничить перечень директив, которые позволено изменять в файлах .htaccess. Директива может принимать следующие значения:
  - `All` — позволяет пользователям переопределять в файлах .htaccess глобальные параметры доступа:

```
AllowOverride All
```

  - `None` — отключает использование файла .htaccess:

```
AllowOverride None
```

  - `AuthConfig` — разрешает директивы авторизации (`AuthName`, `AuthType`, `AuthUserFile`, `AuthGroupFile` и др.):

```
AllowOverride AuthConfig
```

- `FileInfo` — разрешает директивы, управляющие типами документов (`AddType`, `AddLanguage`, `AddEncoding`, `ErrorDocument`, `LanguagePriory` и др.):

```
AllowOverride FileInfo
```

- `Indexes` — позволяет использование директив, управляющих индексацией каталога (`AddIcon`, `DirectoryIndex`, `FancyIndexing`, `HeaderName` и др.):

```
AllowOverride Indexes
```

- `Limit` — делает возможным использование директив, управляющих доступом к хостам (`Allow`, `Deny` и `Order`):

```
AllowOverride Limit
```

- `Options` — разрешает директивы, управляющие каталогами (`Options` и `XbitHack`):

```
AllowOverride Options
```

4.4.14. Защита содержимого папки паролем

Ограничить доступ к определенной папке можно с помощью следующих директив:

- `AuthType` — задает тип аутентификации. Параметр `Basic` указывает на базовую аутентификацию по имени пользователя и паролю:

```
AuthType Basic
```

- `AuthName` — определяет текст, который будет отображен во всплывающем окне запроса:

```
AuthName "Restricted area"
```

- `AuthUserFile` — указывает местоположение файла паролей;
- `AuthGroupFile` — определяет местоположение файла групп;
- `Require` — задает дополнительные требования, которые должны быть выполнены для предоставления доступа. Могут быть указаны следующие параметры:
  - `valid-user` — доступ предоставляется любому пользователю, имя которого задано в файле, указанном директивой `AuthUserFile`, при условии правильно введенного пароля;
  - `user` — доступ разрешается только указанным пользователям;
  - `group` — доступ разрешается только указанным группам пользователей.

Ограничить доступ к определенной папке можно двумя способами:

- добавив код в файл конфигурации сервера (httpd.conf). При помощи раздела `<Directory>` необходимо указать путь к защищаемой папке:

```
<Directory "${SRVROOT}/htdocs/test">
AuthType Basic
AuthName "Restricted area"
```

```
AuthUserFile "${SRVROOT}/data/pass.conf"
<Limit GET POST>
   Require valid-user
</Limit>
</Directory>
```

- разместив в защищаемой папке файл .htaccess с такими директивами:

```
AuthType Basic
AuthName "Restricted area"
AuthUserFile "${SRVROOT}/data/pass.conf"
<Limit GET POST>
   Require valid-user
</Limit>
```

На виртуальном хостинге доступен только второй способ, предполагающий использование файла .htaccess. На своем локальном компьютере необходимо включить поддержку этого файла в главном файле конфигурации, т. к. по умолчанию использование файла .htaccess запрещено. Для этого находим раздел

```
<Directory "${SRVROOT}/htdocs">
.....
</Directory>
```

Внутри раздела находим директиву

```
AllowOverride None
```

и меняем ее значение на

```
AllowOverride All
```

Сохраняем файл и перезапускаем сервер Apache, чтобы изменения вступили в силу. Затем открываем Notepad++ и набираем приведенный ранее код. Сохраняем набранный текст под названием .htaccess, предварительно создав папку (например, test) в C:\Apache2\htdocs. Создаем любой HTML-документ и сохраняем его в папке test под именем index.html. Содержимое этого файла будет отображаться при успешном входе в папку.

Теперь сформируем файл паролей. Для этого создадим папку data в C:\Apache2. Обратите внимание, мы будем сохранять файл вне корневого каталога документов сервера. Файл паролей не должен быть доступен через Web-интерфейс.

Создать файл паролей (pass.conf) можно с помощью программы htpasswd.exe, расположенной в папке bin (C:\Apache2\bin). Для выполнения программы необходима командная строка. Например, можно воспользоваться файловым менеджером Far (рис. 4.2). Запускаем Far и переходим в папку C:\Apache2\bin.

В командной строке должно быть приглашение

```
C:\Apache2\bin>
```

Убираем правую панель с помощью комбинации клавиш <Ctrl>+<F2>, затем левую с помощью <Ctrl>+<F1> (можно убрать сразу обе панели, нажав <Ctrl>+<O>).

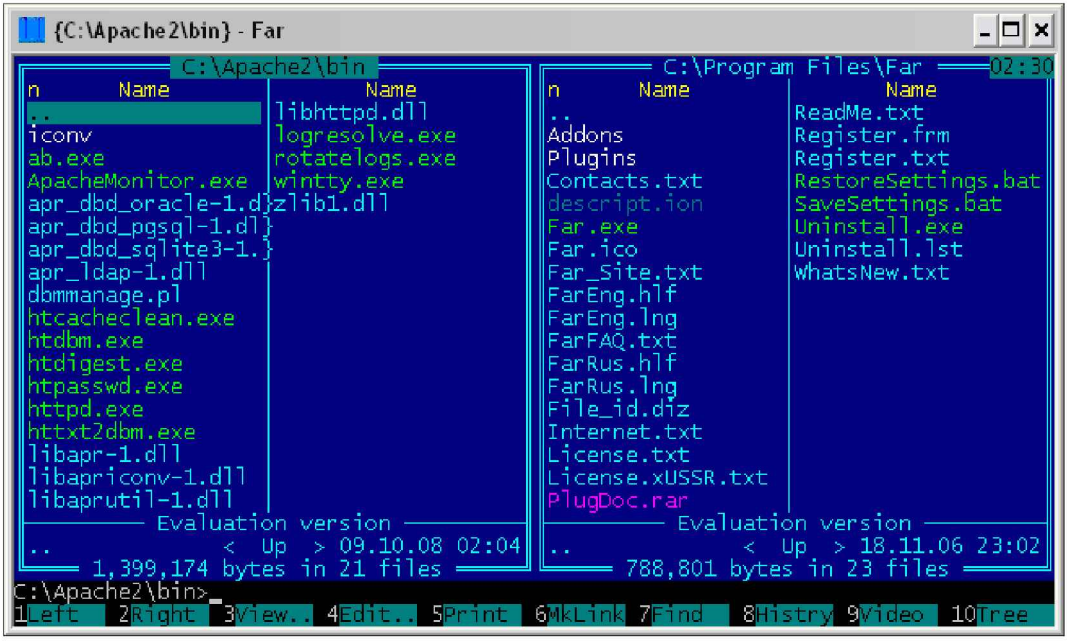


Рис. 4.2. Программа Far

В строке приглашения набираем команду, которая создаст файл C:\Apache2\data\pass.conf и добавит в него информацию о пользователе user1:

```
htpasswd -c C:\Apache2\data\pass.conf user1
```

Нажимаем клавишу <Enter>. В итоге появится приглашение ввести пароль:

```
C:\Apache2\bin>htpasswd -c C:\Apache2\data\pass.conf user1
Automatically using MD5 format.
New password:
```

Вводим пароль (например, "pass1") и нажимаем клавишу <Enter>. Программа попросит повторить пароль:

```
C:\Apache2\bin>htpasswd -c C:\Apache2\data\pass.conf user1
Automatically using MD5 format.
New password: *****
Re-type new password:
```

Повторяем и нажимаем клавишу <Enter>:

```
C:\Apache2\bin>htpasswd -c C:\Apache2\data\pass.conf user1
Automatically using MD5 format.
New password: *****
Re-type new password: *****
Adding password for user user1
```

В итоге будет создан файл pass.conf в папке data со следующими данными:

```
user1:$apr1$IjJpX5aC$TgcfytE5C9dxlCVROx2N/0
```

Как видим, пароль pass1 в этом файле отсутствует, точнее, он присутствует в зашифрованном виде. Тем не менее, чтобы увеличить безопасность сервера, файлы с паролями следует сохранять в папках, не доступных извне, как мы и сделали.

Попробуем теперь создать пароль еще для одного пользователя. Для этого в командной строке набираем:

```
htpasswd -b C:\Apache2\data\pass.conf user2 pass2
```

Обратите внимание: флаг `-c` мы заменили флагом `-b`, а также указали пароль сразу после имени пользователя. Если использовать флаг `-c`, то файл будет перезаписан, и соответственно вся старая информация будет удалена.

После нажатия клавиши <Enter> информацня о новом пользователе и его пароле будет добавлена в конец файла pass.conf, который будет выглядеть так:

```
user1:$apr1$IjJpX5aC$TgcfytE5C9dxlCVROx2N/0
user2:$apr1$rGGVbrC8$EmuYUAExTKRxvHwkzN1xJ0
```

Открываем Web-браузер и в адресной строке набираем:

```
http://localhost/test/
```

Если все сделано правильно, то при попытке открыть любой документ в этой папке будет выведено окно для ввода пароля (рис. 4.3).

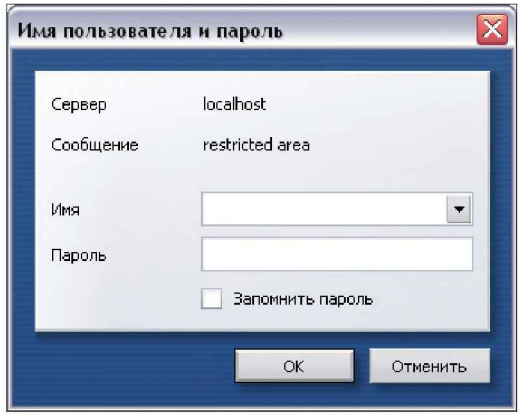


Рис. 4.3. Окно для ввода пароля

Примечание

Не рекомендуется набирать пароли в командной строке, поскольку введенные таким образом команды сохраняются в истории командной строки в незашифрованном виде и могут стать доступными злоумышленникам. Поэтому нужно не лениться и набирать пароли в ответ на приглашение программы htpasswd.exe.

4.4.15. Управление доступом

Директива `Require` пригодится нам и в случае, если мы хотим управлять доступом к папке в зависимости от IP-адреса, с которого поступил запрос, или метода. Здесь мы будем использовать один из следующих параметров:

- `all granted` — доступ разрешен для всех:

```
<Directory "${SRVROOT}/htdocs">
  Require all granted
</Directory>
```

- `all denied` — доступ запрещен для всех:

```
<Directory "${SRVROOT}/htdocs/private">
  Require all denied
</Directory>
```

- `ip` — разрешен доступ лишь с указанных IP-адресов или подсетей:

```
Require ip 192.168 10.0.0.2
```

  Разрешает доступ лишь с IP-адресов из подсети 192.168.* и адреса 10.0.0.2;

- `host` — разрешен доступ лишь с указанных доменных имен:

```
Require host www.vgi.ru
```

- `method` — разрешен доступ лишь с применением указанных методов:

```
Require method post
```

Вставив между наименованием директивы `Require` и остальными ее параметрами слово `not`, мы укажем, что при выполнении указанного нами условия доступ должен, напротив, не предоставляться.

Пример:

```
Require not ip 192.168
```

Предоставляем доступ со всех IP-адресов за исключением находящихся в подсети 192.168.*.

Если нам нужно указать сложное условие, состоящее из нескольких простых условий, мы применим одну из перечисленных далее директив.

- `RequireAll` — должны выполняться условия, заданные во всех директивах `Require`:

```
<Directory "${SRVROOT}/htdocs/protected">
  <RequireAll>
    Require ip 192.168
    Require user admin
  </RequireAll>
</Directory>
```

  Предоставляем доступ к папке protected лишь пользователю admin при условии, что он обратился с IP-адреса, находящегося в подсети 192.168.*.

- `RequireOnly` — должно выполняться хотя бы одно условие из заданных в директивах `Require`:

```
<Directory "${SRVROOT}/htdocs/protected">
  <RequireOnly>
```

```
    Require ip 192.168
    Require user admin
  </RequireOnly>
</Directory>
```

Предоставляем доступ к папке protected пользователю admin и всем прочим пользователям, обратившимся с IP-адреса, который находится в подсети 192.168.*.

- `RequireNone` — не должно выполняться ни одно из условий, заданных в директивах `Require`:

```
<Directory "${SRVROOT}/htdocs/protected">
  <RequireNone>
    Require not ip 192.168
    Require user guest
  </RequireNone>
</Directory>
```

Предоставляем доступ к папке protected всем пользователям, за исключением guest и обратившихся с IP-адреса, который не входит в подсеть 192.168.*.

4.4.16. Регулярные выражения, используемые в директивах

Некоторые директивы позволяют использовать регулярные выраженя. Эти выражения мало чем отличаются от регулярных выражений JavaScript *(см. разд. 3.15.10)*. В них донустимы следующие метасимволы и специальные конструкции:

- `^` — привязка к началу строки;
- `$` — привязка к концу строки;
- `[]` — позволяет указать символы, которые могут встречаться на этом месте в строке. Можно перечислять символы подряд или указать диапазон через тире;
- `[^]` — значение можно инвертировать, если после первой скобки указать символ `^`. Таким образом можно указать символы, которых не должно быть на этом месте в строке.

  Для использованя специальных символов в качестве обычных необходимо перед специальным символом указать символ `"\"`;
- `\d` — любая цифра;
- `\w` — любая латинская буква, цифра или знак подчеркивання;
- `\s` — любой непечатный символ (пробел, табуляция, перевод страницы, новая строка или перевод каретки);
- . (точка) — любой символ, кроме символа перевода строки (`\n`);
- `\D` — не цифра;
- `\W` — не латинская буква, не цифра и не знак подчеркивання;

- \S — не непечатный символ;
- \< и \> — нустая строка перед указанным шаблоном и после него;
- {n} — в точности n вхождений предыдущего символа или подвыражения в строку;
- {n,} — n или более вхождений символа в строку;
- {n,m} — не менее n вхождений символа в строку и не более m. Цифры указываются через запятую без пробела;
- * — ноль или большее число вхождений символа в строку;
- + — один или большее число вхождений символа в строку;
- ? — ноль или одно число вхождений символа в строку;
- n|m — один из символов n или m.

Регулярное выражение можно разбить на подвыражения с помощью круглых скобок. Каждая группа символов, соответствующих подвыражению, сохраняется в памяти. В дальнейшем группу символов можно извлечь, указав после символа $ номер скобки:

```
AliasMatch ^/manual(?:/(?:de|en|es|fr|ru))?(/.*)?$ "C:/Apache2/manual$1"
```

4.4.17. Создание виртуальных серверов

Виртуальные серверы создаются с помощью раздела <VirtualHost> и позволяют размещать на одном сервере несколько сайтов.

Попробуем создать два новых сайта на сервере. Один сайт будет доступен по IP-адресу 127.0.0.1 и имени site1, а второй — по IP-адресу 127.0.0.2 и имени site2. Для этого в каталоге C:\Apache2 создаем две папки: site1 и site2.

В папку site1 добавляем файл index.html, например, следующего содержания:

```
<html>
<head><title>Новый сайт 1</title></head>
<body>Это сайт 1</body>
</html>
```

В папку site2 добавляем файл index.html следующего содержания:

```
<html>
<head><title>Новый сайт 2</title></head>
<body>Это сайт 2</body>
</html>
```

Открываем файл httpd-vhosts.conf (который расположен в папке C:\Apache2\conf\extra) и находим строки

```
<VirtualHost _default_:80>
DocumentRoot "${SRVROOT}/htdocs"
#ServerName www.example.com:80
</VirtualHost>
```

Они объявляют хост, используемый по умолчанию. Исправим строчку

```
#ServerName www.example.com:80
```

так, чтобы она выглядела следующим образом:

```
ServerName localhost
```

И добавим в конец файла следующие строки:

```
<VirtualHost 127.0.0.1:80>
   ServerAdmin webmaster@site1
   DocumentRoot "${SRVROOT}/site1"
   ServerName site1
</VirtualHost>
<Directory "${SRVROOT}/site1">
   Options -Indexes +Includes +FollowSymLinks
   AllowOverride All
   Require all granted
</Directory>

<VirtualHost 127.0.0.2:80>
   ServerAdmin webmaster@site2
   DocumentRoot "${SRVROOT}/site2"
   ServerName site2
</VirtualHost>
<Directory "${SRVROOT}/site2">
   Options -Indexes +Includes +FollowSymLinks
   AllowOverride All
   Require all granted
</Directory>
```

Сохраняем и закрываем файл. Теперь его необходимо подключить к главному конфигурационному файлу httpd.conf. Открываем файл httpd.conf и проверяем, убран ли символ комментария (#) перед строкой

```
Include conf/extra/httpd-vhosts.conf
```

Сохраняем файл httpd.conf и перезагружаем сервер. Теперь открываем Web-браузер и в адресной строке набираем:

```
http://127.0.0.2/
```

В итоге в окне Web-браузера должна отобразиться надпись "Это сайт 2".

> ***Примечание***
>
> Если при наборе в адресной строке **http://127.0.0.2/** надпись не отобразилась и на компьютере установлена операционная система Windows XP Service Pack 2, то необходимо установить пакет обновления KB884020. Скачать его можно со страницы
>
> **http://www.microsoft.com/downloads/details.aspx?FamilyID=17d997d2-5034-4bbb-b74d-ad8430a1f7c8&displaylang=ru**.

Для того чтобы можно было использовать доменные имена (site1 и site2), необходимо в конец файла hosts (расположенного в папке C:\Windows\System32\Drivers\etc) дописать две строки:

```
127.0.0.1       site1
127.0.0.2       site2
```

Теперь открываем Web-браузер и в адресной строке набираем:

```
http://site2/
```

В итоге в окне Web-браузера снова должна появиться надпись "Это сайт 2".

Теперь нам доступны три виртуальных хоста — localhost, site1 и site2. Причем два первых хоста расположены на одном IP-адресе. По аналогии можно создать и другие хосты.

> ***Внимание!***
>
> Название виртуального хоста необходимо указывать без точки. Например, site1, а не site1.ru. В противном случае вы не сможете попасть на реальный сайт site1.ru, не удалив строку из файла hosts (в каталоге C:\Windows\System32\Drivers\etc).

4.5. Установка PHP

Найти дистрибутив интерпретатора PHP можно по адресу **http://windows.php.net/download/**. В списке выбираем нункт **PHP 5.4 (5.4.35) | VC9 x86 Thread Safe | Zip**. В результате на наш компьютер будет загружен файл php-5.4.35-Win32-VC9-x86.zip размером 16,2 Мбайт.

> ***Внимание!***
>
> Для успешной работы PHP 5.4.35 следует установить исполняемую среду Microsoft Visual C++ 2008. Ее дистрибутив можно найти по интернет-адресу
>
> **http://www.microsoft.com/downloads/en/details.aspx?FamilyID=A5C84275-3B97-4AB7-A40D-3802B2AF5FC2**.
>
> PHP 5.5.0 и более новые ее версии не работают под Windows XP и 2003 Server.

Создаем в C:\ папку php5 и распаковываем загруженный архивный файл в нее. С помощью Notepad++ открываем файл php.ini-development (C:\php5\php.ini-development) и сохраняем как php.ini. Не спешите закрывать файл, т. к. в него необходимо внести изменения. Для этого находим строку

```
; extension_dir = "ext"
```

и заменяем ее на

```
extension_dir = "C:/php5/ext"
```

Если этого не сделать, то библиотеки нужно скопировать из C:\php5\ext в C:\WINDOWS\system32. Не будем засорять систему и оставим их там, где они уже есть. Вместо этого просто пропишем к ним путь.

Далее необходимо подключить некоторые библиотеки. Находим строчки

```
; extension=php_mysql.dll
; extension=php_mysqli.dll
```

и убираем точку с запятой перед ними:

```
extension=php_mysql.dll
extension=php_mysqli.dll
```

Таким образом мы включили поддержку баз данных MySQL. Кроме этой библиотеки нам понадобится возможность работы с графикой через PHP. Это достигается подключением библиотеки php_gd2.dll. Заменяем строку

```
;extension=php_gd2.dll
```

на

```
extension=php_gd2.dll
```

Еще одна библиотека, которая может пригодиться, позволяет соединяться и работать с серверами. Находим строку

```
;extension=php_curl.dll
```

и убираем точку с запятой:

```
extension=php_curl.dll
```

А следующая библиотека содержит функции для работы с многобайтовыми кодировками. Меняем строку

```
;extension=php_mbstring.dll
```

на

```
extension=php_mbstring.dll
```

Указываем кодировку по умолчанию. Для этого находим строку

```
;default_charset = "UTF-8"
```

и меняем ее на

```
default_charset = "UTF-8"
```

если собираемся создавать страницы на HTML 5, или на

```
default_charset = "windows-1251"
```

если предпочитаем HTML 4.

Находим строку

```
;include_path = ".;c:\php\includes"
```

и меняем ее на

```
include_path = ".;C:\php5\includes"
```

Предварительно создадим папку includes в C:\php5. Здесь будут храниться подключаемые файлы.

Находим строку

```
;session.save_path = "/tmp"
```

и меняем ее на

```
session.save_path = "c:/php5/tmp"
```

Предварительно создадим папку tmp в C:\php5. Здесь будут храниться временные файлы сессий.

Заменяем строку

```
session.use_trans_sid = 0
```

на

```
session.use_trans_sid = 1
```

Это позволит без затруднений работать с сесснями PHP.

Чтобы использовать упрощенный стиль тегов включения кода PHP, замеияем строку

```
asp_tags = Off
```

на

```
asp_tags = On
```

Отыскиваем строку

```
short_open_tag = Off
```

и меняем ее на

```
short_open_tag = On
```

Проверяем значение директивы

```
display_errors = On
```

Находим строку

```
upload_max_filesize = 2M
```

и увеличиваем максимально допустимый размер загружаемых файлов до 16 Мбайт:

```
upload_max_filesize = 16M
```

Находим строку

```
;upload_tmp_dir =
```

и заменяем ее на

```
upload_tmp_dir = "C:/php5/tmp"
```

Заменяем строку

```
;date.timezone =
```

на

```
date.timezone = "Europe/Moscow"
```

Примечание

Выбрать название зоны для вашей местности можно на странице **http://ru2.php.net/manual/ru/timezones.php**.

Включаем вывод всех сообщений об ошибках:

```
error_reporting = E_ALL | E_STRICT
```

Сохраняем и закрываем файл php.ini.

Теперь необходимо вписать поддержку PHP в файл конфигурации сервера Apache. Открываем файл httpd.conf и находим строки

```
<IfModule dir_module>
   DirectoryIndex index.html
</IfModule>
```

и вместо них вставляем следующие строки:

```
<IfModule dir_module>
 DirectoryIndex index.php index.html index.htm index.shtml index.html.var
</IfModule>

PHPIniDir "C:/php5"
LoadModule php5_module "C:/php5/php5apache2_4.dll"
AddType application/x-httpd-php .php
```

Сохраняем и закрываем файл httpd.conf.

Далее необходимо добавить каталог с установленным интерпретатором PHP в переменную `PATH` операционной системы. Для этого в меню **Пуск** выбираем нункт **Панель управлення** (или **Пастройка | Панель управлення**). В открывшемся окне выбираем нункт **Снстема**. Если мы пользуемся Windows Vista или более новой версией этой системы, щелкнем гиперссылку **Донолнительные нараметры снстемы**. Переходим на вкладку **Донолнительно** (рис. 4.4) и нажимаем кнопку **Переменные среды**.

В разделе **Снстемные неременные** (рис. 4.5) делаем двойной щелчок на строке **Path** (или выделяем строку и нажимаем кнопку **Изменнть**).

В начало к имеющемуся значению переменной `PATH` добавляем нуть к каталогу, в который мы установили PHP (C:\php5) через точку с запятой (рис. 4.6):

```
C:\php5;
```

Точку с запятой необходимо обязательно поставить, т. к. этот символ разделяет нути. Трижды нажимаем кнопку **OK**. После данных изменений следует перезагрузить компьютер.

Когда компьютер перезагрузится, открываем Notepad++ и набираем следующий код:

```
<?php
phpinfo();
?>
```

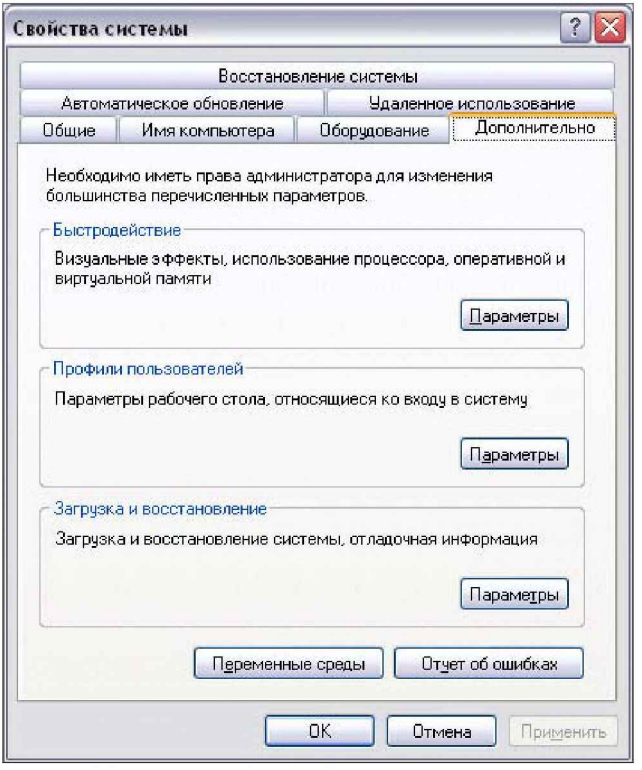


Рис. 4.4. Вкладка **Дополнительно** окна **Свойства системы**

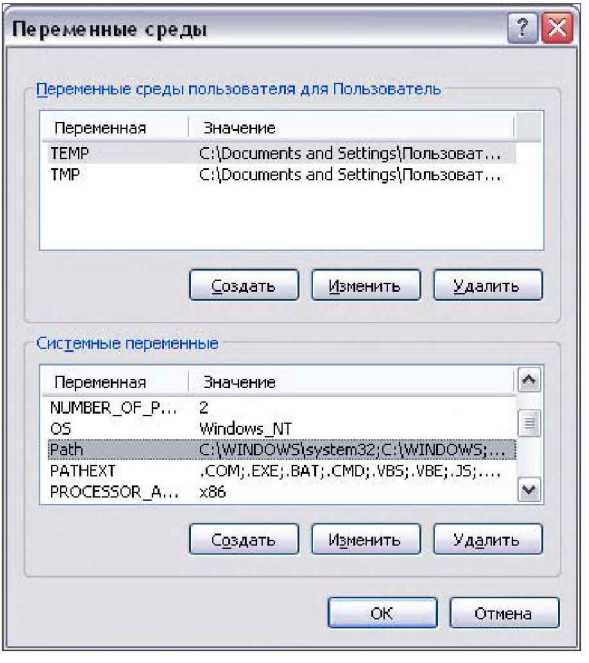


Рис. 4.5. Окно **Переменные среды**

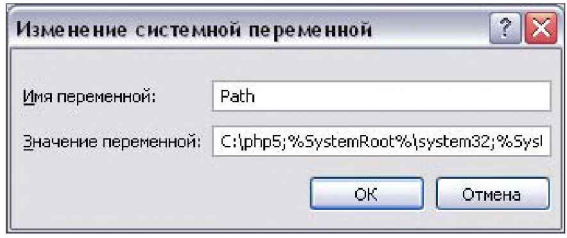
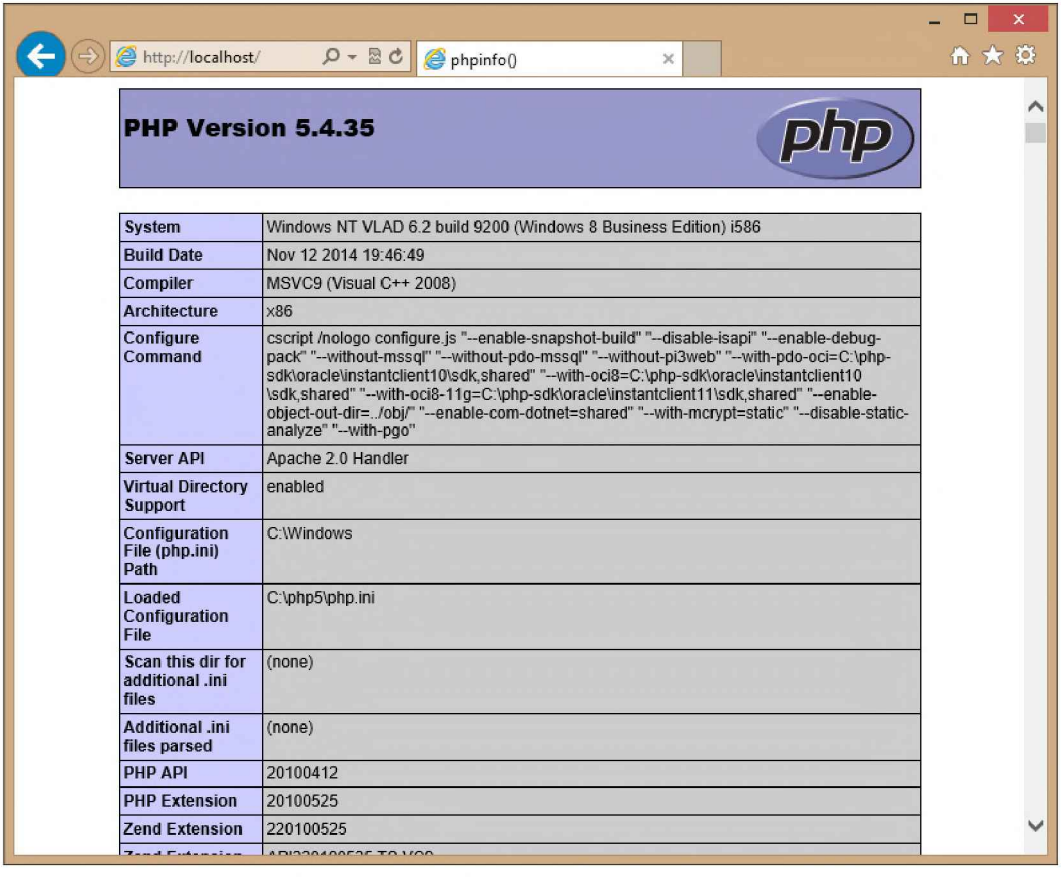


Рис. 4.6. Окно **Изменение системной переменной**

Сохраняем файл под именем index.php в C:\Apache2\htdocs. Открываем Web-браузер и в адресной строке набираем

```
http://localhost/
```

Если в окне Web-браузера отобразилась страничка с информацией об интерпретаторе PHP (рис. 4.7), то это уже хорошо. Тем не менее, это не гарантирует правильность настроек, т. к. PHP может работать и без конфигурационного файла php.ini. Чтобы проверить основные настройки, следует запустить код из листинга 4.1.



System	Windows NT VLAD 6.2 build 9200 (Windows 8 Business Edition) i586
Build Date	Nov 12 2014 19:46:49
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\php5\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20100412
PHP Extension	20100525
Zend Extension	220100525

Рис. 4.7. Результат выполнения функции `phpinfo()`

Листинг 4.1. Проверка корректности установки PHP 5.4

```
<?php
$err = array();
if (!file_exists('C:\\php5\\php.ini'))
   $err[] = 'Файл C:\\php5\\php.ini не существует';
$path = php_ini_loaded_file();
if (strtolower($path) !== 'c:\\php5\\php.ini')
   $err[] = 'Пути к файлу php.ini не совпадают';
if (!file_exists('C:\\php5\\ext\\'))
   $err[] = 'Папка C:\\php5\\ext\\ не существует';
$ext = ini_get("extension_dir");
if (strtolower($ext) !== 'c:/php5/ext')
   $err[] = 'Проверьте значение директивы extension_dir';
$inc = ini_get('include_path');
if (strtolower($inc) !== '.;c:\\php5\\includes')
   $err[] = 'Проверьте значение директивы include_path';
$ses = ini_get('session.save_path');
if (strtolower($ses) !== 'c:/php5/tmp')
   $err[] = 'Проверьте значение директивы session.save_path';
if (!file_exists('C:\\php5\\tmp\\'))
   $err[] = 'Папка C:\\php5\\tmp\\ не существует';
if (!file_exists('C:\\php5\\includes\\'))
   $err[] = 'Папка C:\\php5\\includes\\ не существует';
$upl = ini_get('upload_tmp_dir');
if (strtolower($upl) !== 'c:/php5/tmp')
   $err[] = 'Проверьте значение директивы upload_tmp_dir';
if (!extension_loaded('gd'))
   $err[] = 'Библиотека GD не подключена';
if (!extension_loaded('mbstring'))
   $err[] = 'Библиотека mbstring не подключена';
if (!extension_loaded('mysql'))
   $err[] = 'Библиотека mysql не подключена';
if (!extension_loaded('mysqli'))
   $err[] = 'Библиотека mysqli не подключена';
$path = strtolower($_SERVER['PATH']);
if (strpos($path, 'c:\php5') === false)
   $err[] = 'Не прописан путь к папке c:\php5 в Path';
if (count($err) == 0) echo 'Ошибок нет';
else {
   echo '<div style="color:red;">';
   echo implode('<br>', $err) . '</div>';
}
?>
```

Если после выполнения кода было выведено сообщение "Ошибок нет" — значит, все установлено нормально. Установка и настройка интерпретатора PHP завершена.

4.6. Установка MySQL

Найти дистрибутив MySQL можно по адресу **http://dev.mysql.com/downloads/mysql/**. Щелкаем гиперссылку **Looking for previous GA versions?**, в раскрывающемся списке **Select Version** выбираем нункт **5.5.40**, в раскрывающемся списке **Select Platform** — нункт **Microsoft Windows**. Ниже находим позицию **Windows (x86, 32-bit), MSI Installer** с размером дистрибутива 39,2 Мбайт, щелкаем расположенную в правой части этой позиции гиперссылку **Download**, а на следующей странице — гиперссылку **No thanks, just start my download**.

> ***Внимание!***
> MySQL 5.6.0 и более новые его версии не работают под Windows XP и 2003 Server.

Скачиваем и занускаем файл mysql-5.5.40-win32.msi.

1. Отобразится окно мастера установки (рис. 4.8), нажимаем кнопку **Next**.

Рис. 4.8. Установка сервера MySQL. Шаг 1

2. В следующем окне (рис. 4.9) читаем лицензионное соглашение, соглашаемся с ним, установив флажок **I accept the terms in the License Agreement**, и нажимаем кнопку **Next**.
3. В следующем окне (рис. 4.10) нажимаем кнопку **Typical**.
4. Мастер предложит нам начать установку (рис. 4.11). Нажимаем кнопку **Install** и положительно отвечаем на появившееся на экране предупреждение UAC.
5. Далее подряд появятся два окна с малоинтересной для нас рекламной информацией. Пронускаем их, нажав кнопку **Next**.

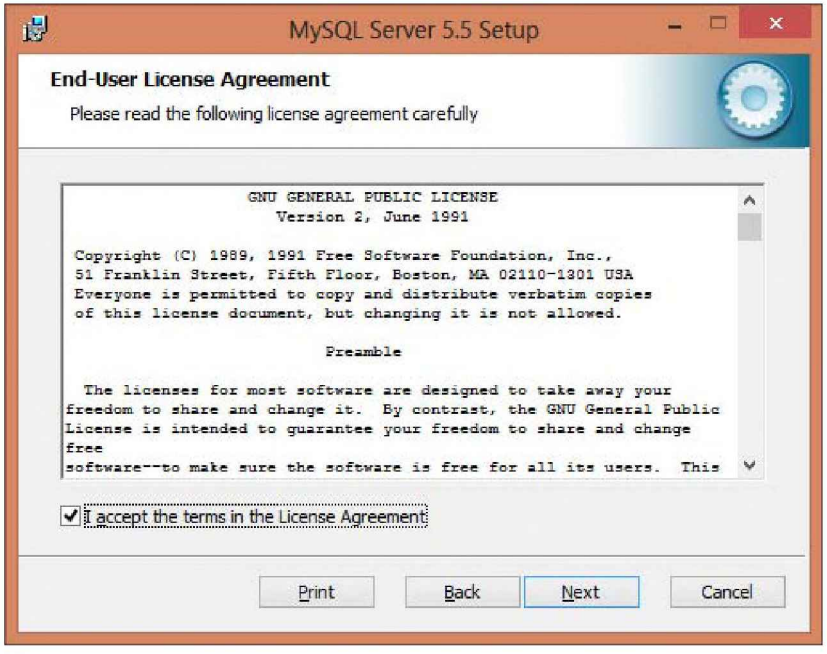


Рис. 4.9. Установка сервера MySQL. Шаг 2

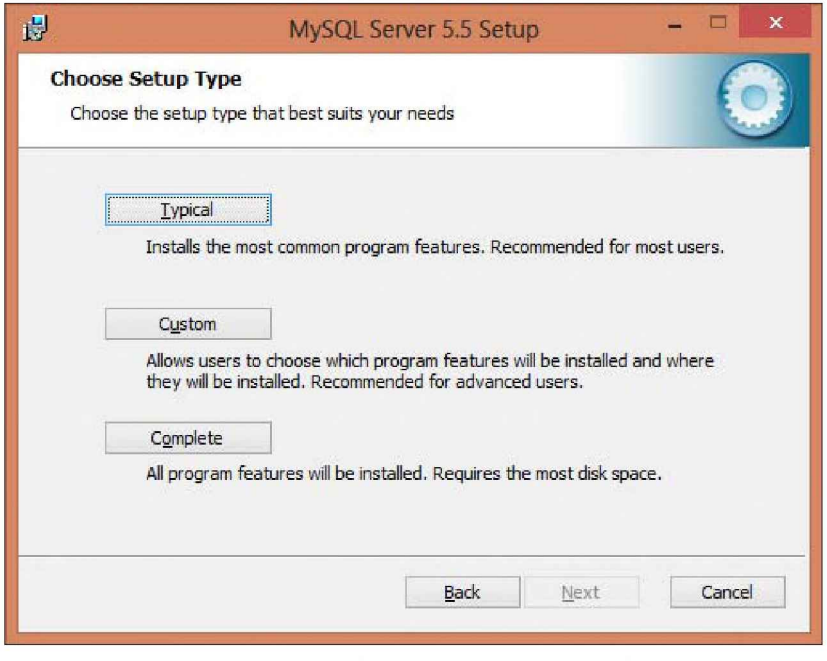


Рис. 4.10. Установка сервера MySQL. Шаг 3

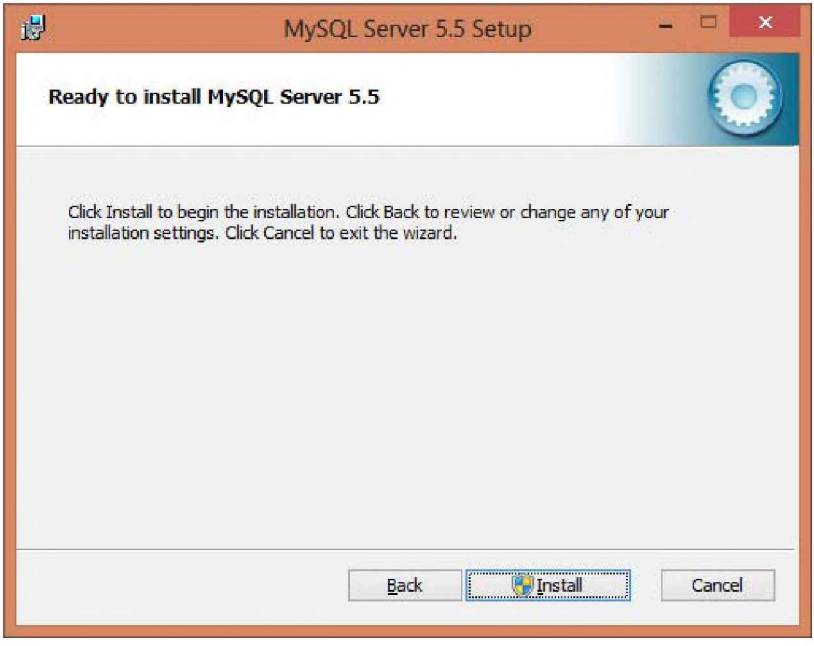


Рис. 4.11. Установка сервера MySQL. Шаг 4

6. В следующем окне (рис. 4.12) проверим, установлен ли флажок **Launch the MySQL Instance Configuration Wizard**; он указывает установщику запустить стандартную утилиту настройки сервера. Нажмем кнопку **Finish**.

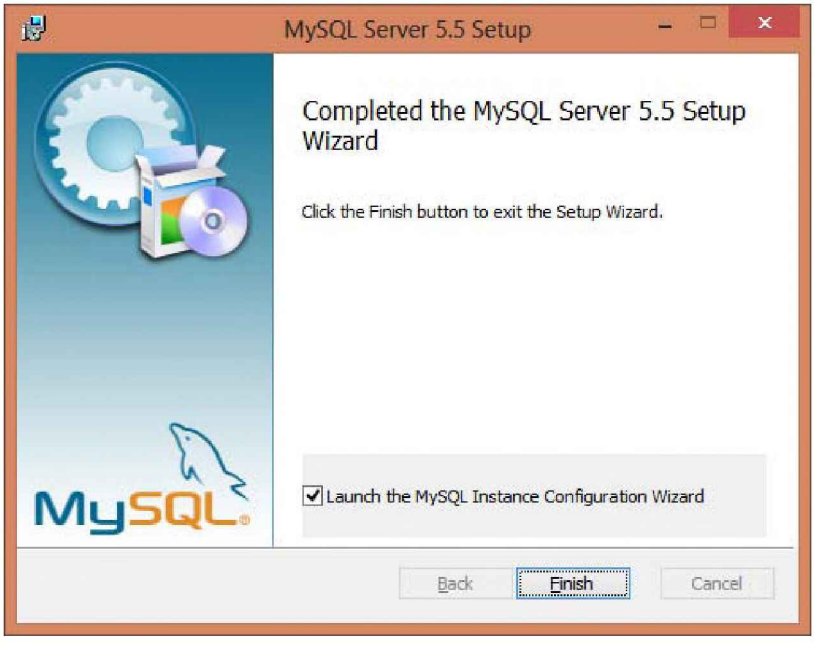


Рис. 4.12. Установка сервера MySQL. Шаг 5

7. Далее мастер (рис. 4.13) позволит частично настроить конфигурацию. Нажимаем кнопку **Next**.
8. Выбираем пункт **Standard Configuration** (рис. 4.14) и нажимаем кнопку **Next**.

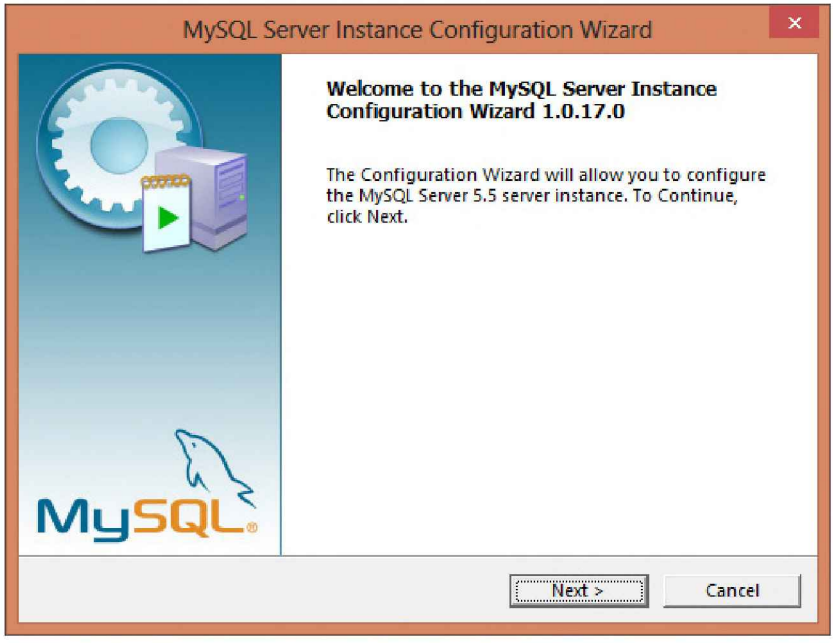


Рис. 4.13. Установка сервера MySQL. Шаг 6

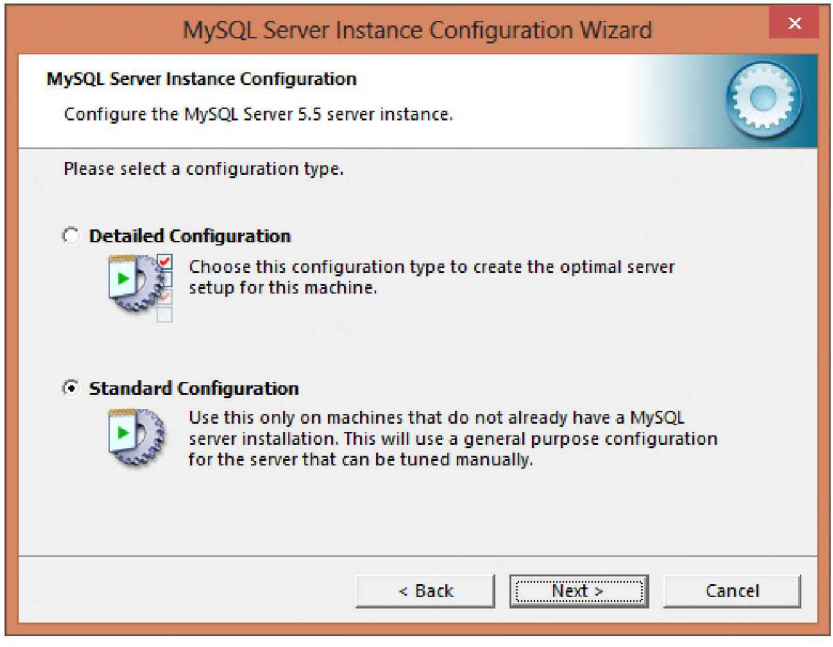


Рис. 4.14. Установка сервера MySQL. Шаг 7

9. Устанавливаем флажок **Install As Windows Service** (рис. 4.15). Из списка **Service Name** выбираем пункт **MySQL5**. Устанавливаем флажок напротив пункта **Launch the MySQL Server automatically**. Нажимаем кнопку **Next**.
10. В следующем окне (рис. 4.16) устанавливаем флажок напротив пункта **Modify Security Settings** и вводим пароль для привилегированного пользователя root

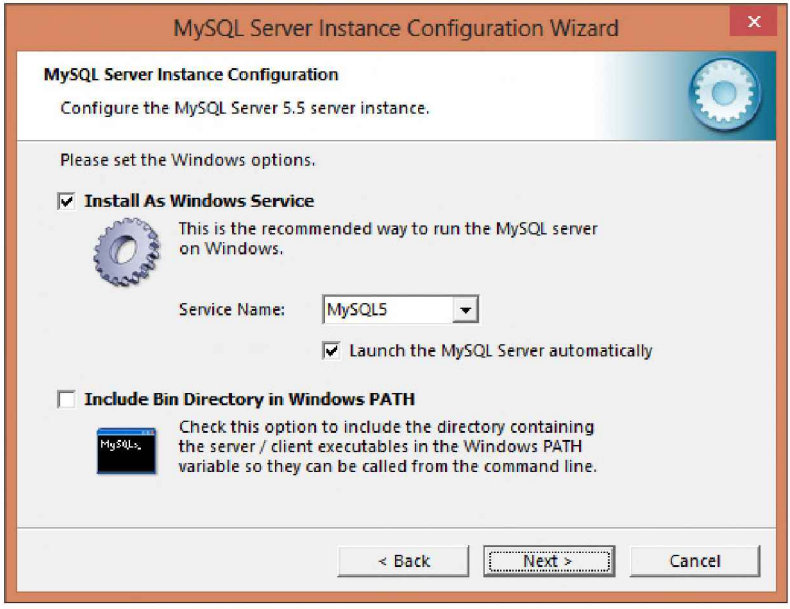


Рис. 4.15. Установка сервера MySQL. Шаг 8

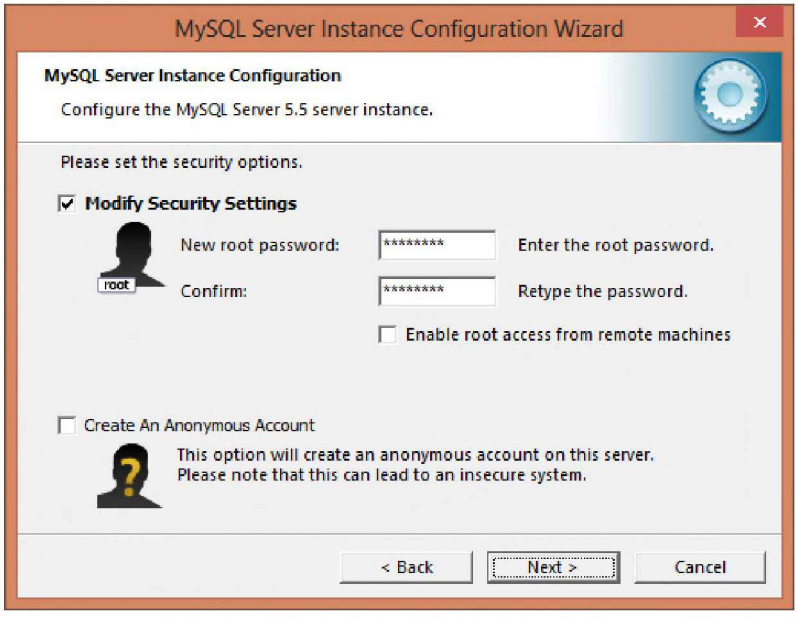


Рис. 4.16. Установка сервера MySQL. Шаг 9

(например, "123456", а лучше что-нибудь более сложное) в поле **New root password**, а затем повторяем пароль в поле **Confirm**. Если был установлен другой пароль, то запишем его. Нажимаем кнопку **Next**.

11. Нажимаем кнопку **Execute** для начала создания конфигурации (рис. 4.17).
12. Нажимаем кнопку **Finish** для завершения установки (рис. 4.18).

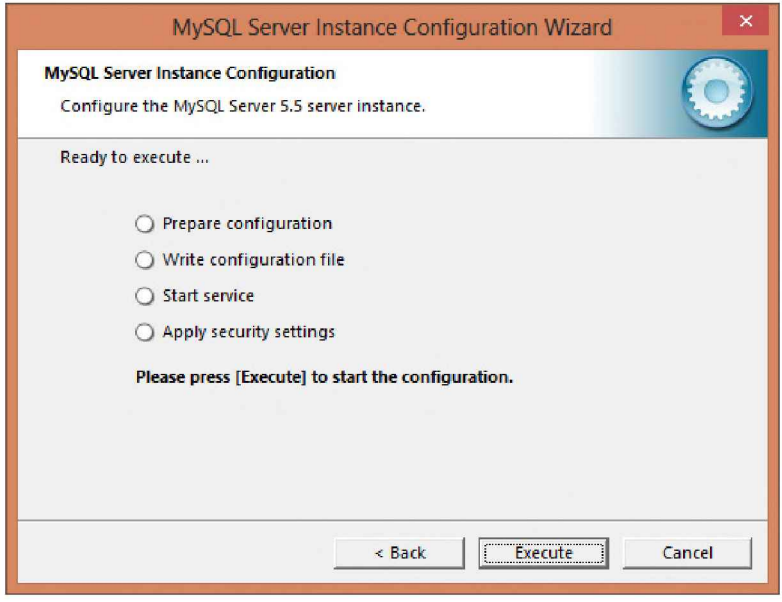


Рис. 4.17. Установка сервера MySQL. Шаг 10

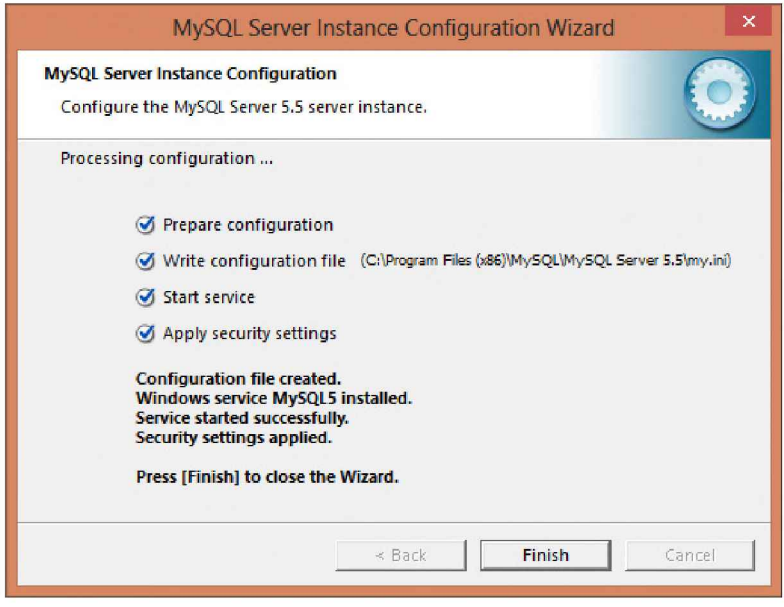


Рис. 4.18. Установка сервера MySQL. Шаг 11

Теперь проверим MySQL на работоспособность. Для этого открываем Notepad++ и набираем код, приведенный в листинге 4.2.

Листинг 4.2. Вывод содержимого базы данных MySQL

```
<?php
if ($db = mysql_connect('localhost', 'root', '123456')) {
   echo '<h2>Содержимое базы данных "mysql"</h2>';
   $query = 'SHOW TABLES FROM `mysql`';
   if ($res = mysql_query($query)) {
      while ($row = mysql_fetch_row($res)) {
         echo 'Таблица: ' . $row[0] . '<br>';
      }
   }
   else {
       echo 'Ошибка ' . mysql_errno() . ' ' . mysql_error();
   }
}
else {
   echo 'Ошибка ' . mysql_errno() . ' ' . mysql_error();
}
?>
```

Если вы при настройке MySQL указали другой пароль для пользователя root, нужно соответствующим образом изменить вторую строку в набранном файле. Сохраняем файл под именем test.php в C:\Apache2\htdocs. Открываем Web-браузер и в адресной строке набираем **http://localhost/test.php**.

Если в окне Web-браузера отобразился текст, приведенный в листинге 4.3, значит, все в порядке.

Листинг 4.3. Содержимое базы данных

```
Содержимое базы данных "mysql"

Таблица: columns_priv
Таблица: db
Таблица: event
Таблица: func
Таблица: general_log
Таблица: help_category
Таблица: help_keyword
Таблица: help_relation
Таблица: help_topic
Таблица: host
```

```
Таблица: ndb_binlog_index
Таблица: plugin
Таблица: proc
Таблица: procs_priv
Таблица: proxies_priv
Таблица: servers
Таблица: slow_log
Таблица: tables_priv
Таблица: time_zone
Таблица: time_zone_leap_second
Таблица: time_zone_name
Таблица: time_zone_transition
Таблица: time_zone_transition_type
Таблица: user
```

Если вместо этого текста появилось сообщение типа

```
Parse error: syntax error, unexpected T_IF, expecting ',' or ';' in
C:\Apache2\htdocs\test.php on line 6
```

то вы допустили какую-либо ошибку при наборе кода, например, забыли поставить точку с запятой в конце строки или пропустили скобку.

Если отобразилось сообщение

```
Ошибка 1045 Access denied for user 'root'@'localhost'
(using password: YES)
```

то пароль был введен неправильно. Если при установке сервера был введен другой пароль, то следует указать именно его.

Если отобразилось сообщение вроде

```
Ошибка 2005 Unknown MySQL server host 'localholst' (11004)
```

или

```
Ошибка 2002 php_network_getaddresses: getaddrinfo failed: Этот хост неизвестен.
```

значит, неправильно указано название сервера localhost.

Если отобразилось сообщение такого вида

```
Ошибка 1049 Unknown database 'mysyql'
```

то неправильно указано имя базы данных.

Если отобразилось сообщение

```
Fatal error: Call to undefined function mysql_connect() in
C:\Apache2\htdocs\test.php on line 3
```

значит, отсутствуют необходимые библиотеки. Убедитесь, что были исполнены все инструкции по установке PHP. Именно при этой установке мы подключали необходимые библиотеки для работы с MySQL.

Установка сервера MySQL закончена.

4.7. Установка phpMyAdmin

Программа phpMyAdmin позволит наглядно работать с базами данных. Для установки необходимо загрузить дистрибутив со страницы **http://www.phpmyadmin.net/home_page/downloads.php**. Выбираем пункт **phpMyAdmin-4.2.12-all-languages.zip** (размер файла — 8,9 Мбайт). Распаковываем архив в текущую папку. Переименовываем папку в pma и копируем ее в C:\Apache2\htdocs.

Открываем Notepad++ и набираем следующий код:

```
<?php
  $i = 0;
  $i++;
  $cfg['blowfish_secret'] = '12345678';
  $cfg['Servers'][$i]['host'] = 'localhost';
  $cfg['Servers'][$i]['extension'] = 'mysql';
  $cfg['Servers'][$i]['connect_type'] = 'tcp';
  $cfg['Servers'][$i]['compress'] = false;
  $cfg['Servers'][$i]['auth_type'] = 'config';
  $cfg['Servers'][$i]['user'] = 'root'; // Логин
  $cfg['Servers'][$i]['password'] = '123456'; // Пароль
?>
```

Сохраняем файл под названием config.inc.php в папке C:\Apache2\htdocs\pma. Теперь открываем Web-браузер и в адресной строке набираем **http://localhost/pma/**. В итоге должно отобразиться окно, показанное на рис. 4.19.

Если все надписи выводятся по-английски, выберем в раскрывающемся списке **Language** пункт **Русский — Russian**.

Внизу окна отобразится надпись "Дополнительные возможности phpMyAdmin не настроены в полной мере, некоторые функции были отключены". Не обращайте на нее внимания.

В списке **Соиоставлеиие соедииеиия с MySQL** задается кодовая таблица, используемая при работе с базами данных MySQL. Если мы собираемся использовать язык HTML 4, выбираем пункт **cp1251_general_ci**; поклонникам HTML 5 следует выбрать пункт **utf8mb4_general_ci**.

Попробуем создать новую базу данных. Для этого переключимся на вкладку **Базы даииых** и в поле **Создать базу даииых** наберем `test2`. В списке **Сравиеиие** выбираем пункт **cp1251_general_ci**. Нажимаем кнопку **Создать**. Через некоторое время отобразится сообщение "База данных test2 была создана". Вновь созданная база тотчас появится в иерархическом списке, расположенном слева, и в списке, что находится на вкладке **Базы даииых**.

В левом иерархическом списке выбираем созданную базу test2. Переходим на вкладку **SQL**. В текстовом поле набираем следующий текст:

```
CREATE TABLE `city` (
  `id_city` int(11) NOT NULL auto_increment,
```

```
  `name_city` varchar(255) default NULL,
  PRIMARY KEY  (`id_city`)
) ENGINE=MyISAM;

INSERT INTO `city` (`id_city`, `name_city`) VALUES
(1, 'Санкт-Петербург'),
(2, 'Москва'),
(3, 'Новгород'),
(4, 'Тверь'),
(5, 'Минск');
```

Нажимаем кнопку **Виеред**. В итоге отобразится код заданного нами запроса со статистическими сведениями.

Теперь добавим нового пользователя для созданной базы данных. Для этого переходим по ссылке **Сервер: localhost**. Далее выбираем вкладку **Пользователи**. В открывшемся окне переходим по ссылке **Добавить иользователя**. В поле **Пмя иользователя** набираем `petr`. В списке **Хост** выбираем пункт **Локальиый**. В поле **Пароль** набираем `123`. Повторяем пароль в поле **Подтверждеиие**. Нажимаем кнопку **Виеред**.

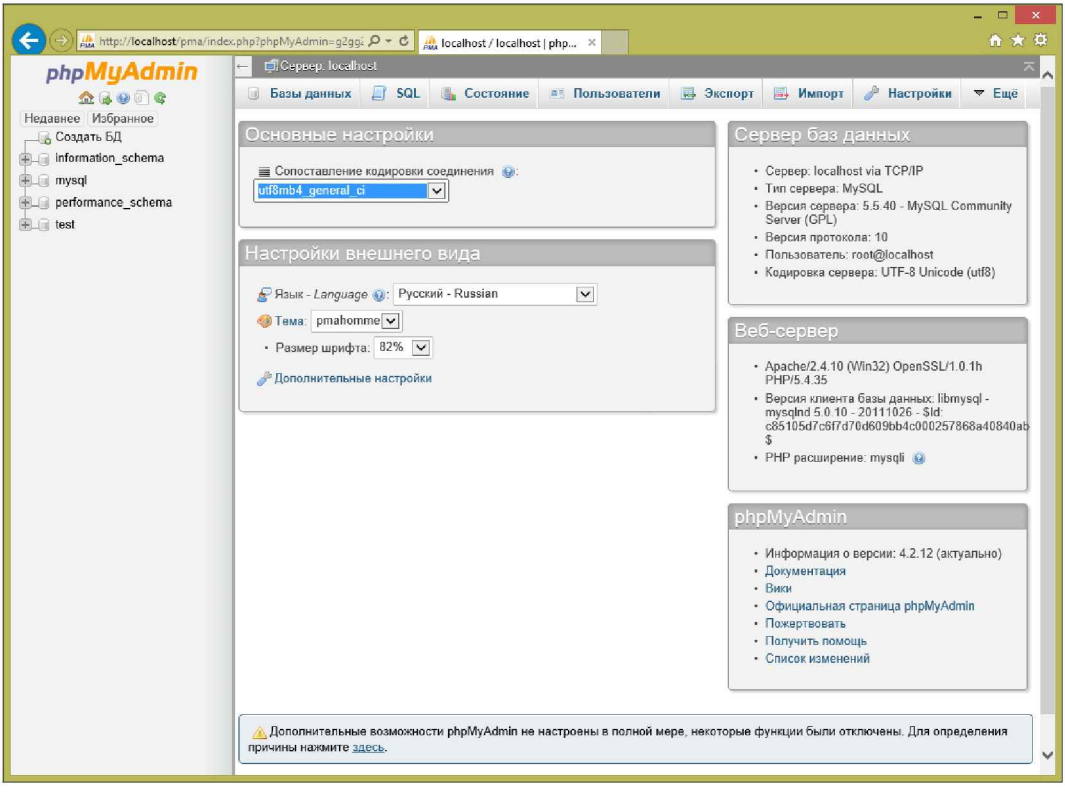


Рис. 4.19. Программа phpMyAdmin

Добавленный нами пользователь появится в списке, присутствующем на вкладке **Пользователи**. Находим его и нажимаем расположенную в представляющем его пункте списка гиперссылку **Редактировать иривилегии**. В появившейся странице щелкаем гиперссылку **База даииых**. В списке **Добавить иривилегии иа следующую базу даииых** выбираем базу test2. Отобразится окно **Редактироваиие иривилегий**. Устанавливаем флажки во всех разделах (**Даииые**, **Структура** и **Администрироваиие**) или сразу устанавливаем флажок **Отметить все**. Нажимаем кнопку **Виеред**.

После добавления пользователя необходимо перезагрузить привилегии. Для этого переходим по ссылке **Сервер: localhost**. Переходим по ссылке **Привилегии**. Далее выбираем ссылку **Перезагрузить иривилегии**. В итоге отобразится сообщение "Привилегии были успешно перезагружены".

Попробуем отобразить все города из нашей базы данных. Открываем Notepad++ и набираем код, приведенный в листинге 4.4.

Листинг 4.4. Вывод названий городов из базы данных

```
<?php
if ($db = mysql_connect('localhost', 'petr', '123')) {
   mysql_select_db('test2');
   $q = 'SELECT * FROM `city` ORDER BY `name_city` DESC';
   $res = mysql_query($q) or die(mysql_error());
   echo 'Содержимое таблицы city<br><br>';
   while ($row = mysql_fetch_assoc($res)) {
      echo $row['name_city'] . '<br>';
   }
}
else {
   echo 'Ошибка ' . mysql_errno() . ' ' . mysql_error();
}
?>
```

Сохраняем файл под названием test2.php в C:\Apache2\htdocs. Открываем Web-браузер и в адресной строке набираем **http://localhost/test2.php**.

Если в открытом документе вместо русских букв отобразились знаки вопроса:

```
Содержимое таблицы city

?????
?????-?????????
????????
??????
?????
```

значит, нужно настроить MySQL для работы с русским языком.

Пользователи Windows XP могут просто открыть файл C:\Program Files\MySQL\MySQL Server 5.5\my.ini с помощью Notepad++. Пользователям более поздних версий Windows придется скопировать его в какую-либо папку, не защищенную подсистемой UAC, и открыть в текстовом редакторе эту копию.

В разделе `[client]` после строки

```
port=3306
```

добавляем строку

```
character-sets-dir="C:/Program Files/MySQL/MySQL Server 5.5/share/charsets"
```

Находим секцию

```
[mysql]
default-character-set=latin1
```

и меняем на

```
[mysql]
default-character-set=cp1251
character-sets-dir="C:/Program Files/MySQL/MySQL Server 5.5/share/charsets"
```

Далее находим секцию `[mysqld]`. В этой секции меняем строку

```
default-character-set=latin1
```

на четыре строки:

```
default-character-set=cp1251
character-sets-dir="C:/Program Files/MySQL/MySQL Server 5.5/share/charsets"
init-connect="SET NAMES cp1251"
skip-character-set-client-handshake
```

Здесь была описана настройка MySQL для поддержки кодовой таблицы 1251. Чтобы указать для него поддержку UTF-8, следует во всех приведенных настройках заменить подстроку `cp1251` на `utf8`.

Сохраняем файл my.ini. Пользователям Windows Vista и более поздних ее версий следует скопировать измененную копию назад в папку C:\Program Files\MySQL\MySQL Server 5.5.

Перезагрузив компьютер, открываем Web-браузер и в адресной строке набираем **http://localhost/test2.php**. В итоге все должно отобразиться на русском языке:

```
Содержимое таблицы city

Тверь
Санкт-Петербург
Новгород
Москва
Минск
```

Если это все равно не произошло, то в коде листинга 4.4 после строки

```
mysql_select_db("test2");
```

добавьте следующий запрос

```
mysql_query("SET NAMES cp1251");
```

Установка phpMyAdmin закончена.

Теперь необходимо изменить тип запуска серверов. В меню **Пуск** выбираем пункт **Пастройка | Паиель уиравлеиия**. В открывшемся окне выбираем пункт **Адмиии-стрироваиие**, а затем **Службы**. Находим службу Apache2.4. Щелкаем правой кнопкой мыши на этой строке и в контекстном меню выбираем пункт **Свойства**. В открывшемся окне из списка **Тии заиуска** также выбираем нункт **Вручиую**. Нажимаем кнопку **Примеиить**, а затем **OK**.

Ту же самую операцию выполняем со службой MySQL5.

Теперь создадим два файла:

- StartServer.bat — для запуска серверов Apache и MySQL. Содержимое файла:

```
@echo off
NET start Apache2.4
NET start MySQL5
```

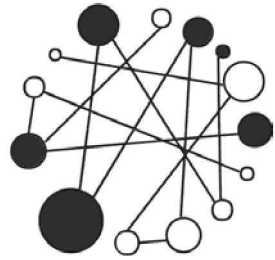
- StopServer.bat — для остановки серверов Apache и MySQL. Содержимое файла:

```
@echo off
NET stop Apache2.4
NET stop MySQL5
```

Разместите эти два файла на Рабочем столе и запускайте службы, только когда это необходимо. Запустили с помощью файла StartServer.bat, поработали, а затем обязательно остановите службы с помощью файла StopServer.bat. Не выключайте компьютер с запущенными службами. Обязательно остановите их перед выключением или перезагрузкой компьютера.

> ***ОБРАТИТЕ ВНИМАНИЕ!***
>
> Перед выходом в Интернет обязательно остановите службы, иначе ваш компьютер может оказаться в руках злоумышленника.

ГЛАВА 5



Основы PHP. Создаем динамические Web-страницы

5.1. Основные понятия

PHP — это язык программирования, выполняемый на стороне сервера. PHP в отличие от языка JavaScript не зависит от программного обеспечения клиента и поэтому будет выполнен в любом случае.

Последовательность инструкций (называемая *программой* или *скриптом*) выполняется *интерпретатором* языка PHP. Код программы может внедряться в HTML-код. Эта возможность отличает PHP от других языков, используемых в Интернете, например от языка Perl. PHP-код обрабатывается на сервере до того, как страница будет передана Web-браузеру. В итоге Web-браузер получит обычный HTML-код или другой вывод.

5.2. Первая программа на PHP

При изучении языков программирования принято начинать с программы, выводящей надпись "Hello, world". Не будем нарушать традицию и продемонстрируем, как это выглядит на PHP (листинг 5.1).

Листинг 5.1. Первая программа

```
<html>
<head>
<title>Первая программа</title>
</head>
<body>
<?php
echo "Hello, world";
?>
</body>
</html>
```

Набираем код в Notepad++ и сохраняем в формате PHP (например, index.php) в папке C:\Apache2\htdocs. Запускаем Web-браузер и в его адресной строке набираем **http://localhost/**. В итоге в окне Web-браузера отобразится надпись "Hello, world".

Теперь давайте рассмотрим исходный HTML-код (листинг 5.2).

Листинг 5.2. Исходный HTML-код

```
<html>
<head>
<title>Первая программа</title>
</head>
<body>
Hello, world</body>
</html>
```

Как нетрудно заметить, никаких признаков PHP в исходном коде нет. Кроме того, HTML-теги также можно выводить с помощью оператора `echo`.

Давайте заменим содержимое нашего файла на код листинга 5.3.

Листинг 5.3. Вывод HTML-тегов с помощью PHP

```
<?php
echo '<html>';
echo '<head>';
echo '<title>Первая программа</title>';
echo '</head>';
echo '<body>';
echo 'Hello, world';
echo '</body>';
echo '</html>';
?>
```

В итоге получим следующий исходный код:

```
<html><head><title>Первая программа</title></head><body>Hello,
world</body></html>
```

Как видно, в этом случае весь код отображается на одной строке. Чтобы отобразить каждый тег на отдельной строке, необходимо добавить символ перевода строки (листинг 5.4). Для системы UNIX таким символом будет `\n`. В операционной системе Windows символ перевода строки состоит из комбинации двух символов `\r\n`.

Листинг 5.4. Вывод каждого тега на отдельной строке

```
<?php
echo "<html>\n";
```

```
echo "<head>\n";
echo "<title>Первая программа</title>\n";
echo "</head>\n";
echo "<body>\n";
echo "Hello, world\n";
echo "</body>\n";
echo "</html>\n";
?>
```

Теперь каждый тег будет на своей строчке (листинг 5.5).

Листинг 5.5. Результат вывода предыдущей программы

```
<html>
<head>
<title>Первая программа</title>
</head>
<body>
Hello, world
</body>
</html>
```

Кроме того, при выводе HTML-тегов с помощью оператора `echo` следует помнить, что теги могут иметь параметры, значения которых заключаются в кавычки. Например, если попробовать вывести тег `<span>` так, как показано в листинге 5.6, то возникнет ошибка

```
Parse error: parse error, expecting `',' or `';'' in
C:\Apache2\htdocs\index.php on line 5
```

Листинг 5.6. Ошибочный код при выводе кавычек

```
<?php
echo "<html><head>\n";
echo "<title>Первая программа</title>\n";
echo "</head><body>\n";
echo "<span style="color:red">\n";
echo "Hello, world\n";
echo "</span>\n";
echo "</body></html>\n";
?>
```

Обойти данную проблему можно следующими способами:

- добавить защитный слэш перед каждой кавычкой:

```
echo "<span style=\"color:red\">\n";
```

- в операторе `echo` использовать не кавычки, а апострофы:

```
echo '<span style="color:red">';
```

Обратите внимание

Применение этого способа может повлечь за собой другие проблемы. Например, в этом случае нельзя использовать специальные символы (`\n`). Кроме того, если внутри присутствует переменная, то вместо ее значения мы увидим имя переменной.

Все выражения в PHP заканчиваются точкой с запятой. В отличие от JavaScript, где отсутствие этого символа не приводит к сообщению об ошибке, отсутствие точки с запятой в PHP влечет за собой остановку выполнения сценария и выдачу сообщения об ошибке. Это самая распространенная ошибка среди начинающих изучать язык PHP.

5.2.1. Особенности создания скриптов в кодовой таблице UTF-8

Если планируется сохранять файлы PHP-скриптов в кодировке UTF-8, нужно иметь в виду один момент.

При сохранении текстового файла в кодовой таблице UTF-8 Блокнот вставляет в его начало особые служебные символы, называемые сокращенно BOM. Эти символы не являются обязательными и не позволят нам установить заголовки ответа сервера с помощью функции `header()`. Поэтому сохранять файлы следует из программы Notepad++. В меню **Кодировки** устанавливаем флажок **Кодировать в UTF-8 (без BOM)**, а затем набираем код. При копировании кода через буфер обмена советуем вначале сохранить пустой файл в кодировке UTF-8 без BOM, вставить код из буфера обмена, а затем сохранить файл с помощью соответствующей кнопки в панели инструментов.

Если мы хотим создавать страницы в той кодовой таблице, которая указана в настройках сервера, нам не пужно выполнять никаких дополнительных действий. Но если планируется создание страниц в другой кодовой таблице, придется указать в программе эту кодовую таблицу.

- Если сервер настроен на кодировку windows-1251, и планируется использование кодировки UTF-8, шаблон программы будет выглядеть так:

```
<?php
header('Content-Type: text/html; charset=utf-8');
// Сюда вставляем примеры из этого раздела
?>
```

- Если же сервер настроен на кодировку UTF-8, но страницы будут создаваться в кодировке windows-1251, шаблон программы станет таким:

```
<?php
header('Content-Type: text/html; charset=windows-1251');
// Сюда вставляем примеры из этого раздела
?>
```

5.3. Методы встраивания PHP-кода

PHP-код встраивается в документ с помощью *дескрипторов*, иногда называемых также *тегами*:

- `<?php` и `?>`:

```
<?php echo "Hello, world\n"; ?>
```

  Отключить поддержку этих дескрипторов нельзя. Настоятельно рекомендуем использовать именно их.

  Приведенное выражение можно записать в более компактном виде, который стал доступен в PHP 5.4:

```
<?="Hello, world\n"?>
```

- `<?` и `?>`:

```
<? echo "Hello, world\n"; ?>
```

  Доступны, только если директива `short_open_tag` имеет значение `On` *(см. разд. 4.5)*. При работе с этими дескрипторами следует помнить, что могут возникнуть проблемы при выводе XML-документов, т. к. последовательность `<?xml ... ?>` будет воспринята как выделение PHP-кода.

- `<%` и `%>`:

```
<% echo "Hello, world\n"; %>
```

  Для использования этого дескриптора необходимо включить поддержку в файле php.ini. Для этого строку

```
asp_tags = Off
```

  нужно заменить на

```
asp_tags = On
```

  а затем перезапустить сервер Apache;

- `<script language="PHP">` и `</script>`. Удивлены? Внедрить PHP-код можно точно так же, как и JavaScript-код. Нужно только указать в параметре `language` значение `PHP`:

```
<script language="PHP"> echo "Hello, world\n"; </script>
```

  На практике такими дескрипторами никто не пользуется.

5.4. Комментарии в PHP-сценариях

Все, что расположено после `//` или `#` до конца строки в PHP, считается *однострочным комментарием*:

```
// Однострочный комментарий
# Однострочный комментарий
```

Однострочный комментарий можно записать после выражения:

```
echo "Hello, world"; // Однострочный комментарий
echo "Hello, world"; # Однострочный комментарий
```

Кроме того, существует *многострочный комментарий*. Он начинается с символов `/*` и заканчивается символами `*/`.

```
/*
Многострочный комментарий
*/
```

То, что комментарий называется многострочным, отнюдь не означает, что он не может располагаться на одной строке. Пример:

```
/* Комментарий на одной строке */
```

Следует иметь в виду, что многострочные комментарии не могут быть вложенными, так что при комментировании больших блоков необходимо проверять, что в них не встречается закрывающая комментарий комбинация символов `*/`.

Комментарии предназначены для вставки пояснений в текст скрипта, и интерпретатор полностью их игнорирует. Внутри комментария может располагаться любой текст, включая инструкции, которые выполнять не следует. Помните, комментарии нужны программисту, а не интерпретатору PHP. Вставка комментариев в код позволит через некоторое время быстро вспомнить предназначение фрагмента кода.

Программа Notepad++ позволяет быстро закомментировать фрагмент кода. Для этого необходимо выделить одну инструкцию (или сразу несколько инструкций) и из контекстного меню выбрать пункт **Вкл./Выкл. Комментарнй строкн**. В результате в начало каждой выделенной строки будет автоматически вставлен однострочный комментарий.

Если закомментированный блок выделить и повторно выбрать из контекстного меню нункт **Вкл./Выкл. Комментарнй строкн**, то символы комментариев будут автоматически удалены.

С помощью программы Notepad++ можно также вставить многострочный комментарий. Для этого необходимо выделить несколько инструкций и из контекстного меню выбрать нункт **Закомментнровать выделенное**. В результате в начало выделенного фрагмента будут вставлены символы `/*`, а в конец — символы `*/`.

Если выделить закомментированный фрагмент кода и выбрать в контекстном меню нункт **Раскомментнровать выделенное**, символы комментария будут удалены.

5.5. Вывод результатов работы скрипта

Вывести результат можно с помощью двух операторов:

- `echo` — мы уже применяли его для вывода строчки "Hello, world":

```
echo "Hello, world";
```

Можно вывести сразу несколько строк, указав их через запятую:

```
echo "Строка 1", "Строка 2";
```

- `print` — этот оператор позаимствован из языка Perl:

```
print "Hello, world";
```

Большие блоки текста можно выводить, например, следующим образом:

```
<?php
echo 'Строка1<br>
Строка2<br>
Строка3<br>
';
?>
```

Кроме того, можно воспользоваться синтаксисом, который условно называют "документ здесь":

```
<?php
echo <<<METKA
Строка1<br>
Строка2<br>
Строка3<br>
METKA;
?>
```

В этом примере многострочный текст располагается между метками (`METKA`):

```
echo <<<METKA
...
METKA;
```

Вторая (закрывающая) метка должна обязательно находиться на отдельной строке в самом ее начале. После этой метки должна стоять точка с запятой.

Для ускорения работы операторы осуществляют буферизацию данных. Иными словами, вначале строка помещается в память. Когда количество данных достигает определенной величины, данные отправляются Web-браузеру. Для примера выведем 5 строк, но перед выводом каждой строки укажем интерпретатору "заснуть" на одну секунду:

```
<?php
for ($i=1; $i<6; $i++) {
   echo "Строка ", $i, "<br>";
   sleep(1); // "Засыпаем" на 1 секунду
}
?>
```

Результат выполнения этого скрипта мы увидим весь целиком только через 5 секунд. В некоторых случаях необходимо отправлять данные сразу в Web-браузер. Иначе пользователь может подумать, что скрипт "завис". Вывести данные сразу позволяет функция `flush()`, указанная после оператора вывода:

```
<?php
for ($i=1; $i<6; $i++) {
   echo "Строка ", $i, "<br>";
   flush(); // Выводим строку сразу в Web-браузер
   sleep(1); // "Засыпаем" на 1 секунду
}
?>
```

В этом случае строки будут выводиться сразу, а не все одновременно, как это было в предыдущем примере. Следует заметить, что в некоторых случаях (например, если указано значение в директиве `output_buffering`) необходимо дополнительно вызывать функцию `ob_flush()`:

```
<?php
for ($i=1; $i<6; $i++) {
   echo "Строка ", $i, "<br>";
   flush(); // Выводим строку сразу в Web-браузер
   ob_flush();
   sleep(1); // "Засыпаем" на 1 секунду
}
?>
```

5.6. Переменные

Переменные — это участки памяти, в которых программа хранит данные. Каждая переменная должна иметь уникальное имя в программе, состоящее из латинских букв, цифр и знаков подчеркивания. Все имена переменных в PHP начинаются со знака `$`.

Правильные имена переменных: `$x`, `$strName`, `$y1`, `$_name`.

Неправильные имена переменных: `y`, `ИмяПеременной`, `$ИмяПеременной`.

Последнее имя неправильное, т. к. в нем используются русские буквы. Хотя на самом деле такой вариант также будет работать, но лучше русские буквы все же не применять.

При указании имени переменной важно учитывать регистр букв: `$strName` и `$strname` — разные переменные.

5.7. Типы данных и инициализация переменных

В PHP переменные могут содержать следующие типы данных:

- `integer` — целые числа;
- `double` — вещественные числа;
- `string` — строка;

- `bool` — логический тип данных. Может содержать значения `true` или `false`;
- `object` — экземпляры класса;
- `array` — массивы.

При инициализации переменной интерпретатор автоматически относит ее к одному из типов данных. Значение переменной присваивается с помощью оператора = таким образом:

```
$number = 7; // integer
$number2 = 7.8; // double
$string = "Строка"; // Переменной $string присвоено значение Строка
$string2 = 'Строка'; // Переменной $string2 также присвоено
                     // значение Строка
$boolean = true; // Переменной $boolean присвоено
                 // логическое значение true
```

PHP в любой момент времени изменяет тип переменной в соответствии с данными, хранящимися в ней:

```
$var = "Строка"; // тип string
$var = 7; // теперь переменная имеет тип integer
```

Функция `gettype(<Имя_переменной>)` возвращает тип данных переменной (листинг 5.7).

Листинг 5.7. Вывод типа данных переменной

```
<?php
$var = 7;
echo gettype($var); // Выведет: integer
$str = 'Строка';
echo gettype($str); // Выведет: string
$bool = true;
echo gettype($bool); // Выведет: boolean
?>
```

Кроме того, существуют функции проверки конкретного типа переменных:

- `is_int(<Переменная>)` — возвращает `true`, если переменная имеет тип `integer` (целое число);
- `is_integer(<Переменная>)` — возвращает `true`, если переменная имеет тип `integer` (целое число);
- `is_double(<Переменная>)` — возвращает `true`, если переменная имеет тип `double` (вещественное число);
- `is_float(<Переменная>)` — возвращает `true`, если переменная имеет тип `double` (вещественное число);
- `is_string(<Переменная>)` — возвращает `true`, если переменная имеет тип `string` (строка);

- `is_array(<Переменная>)` — возвращает `true`, если переменная имеет тип `array` (массив);
- `is_object(<Переменная>)` — возвращает `true`, если переменная имеет тип `object` (объект);
- `is_bool(<Переменная>)` — возвращает `true`, если переменная имеет тип `boolean` (логический тип данных).

5.8. Проверка существования переменной

С помощью функции `isset(<Переменная>)` можно проверить существование переменной. Если переменная существует, то возвращается `true`. Для примера переделаем нашу первую программу так, чтобы она "здоровалась" не со всем миром, а только с нами (листинг 5.8).

Листинг 5.8. Проверка существования переменной

```
<?php
if (isset($_GET['name'])) {
   echo 'Hello, ' . $_GET['name'];
}
else {
   echo 'Введите ваше имя<br>';
   echo '<form action="' . $_SERVER['SCRIPT_NAME'] . '">';
   echo '<input type="text" name="name">';
   echo '<input type="submit" value="OK">';
   echo '</form>';
}
?>
```

При первом запуске программы появится приглашение ввести имя. Вводим свое имя (например, Николай) и нажимаем **OK**. В итоге отобразится приветствие "Hello, Николай".

Функция `empty(<Переменная>)` проверяет наличие у переменной непустого, ненулевого значения. Возвращает `true`, если переменная пустая, не существует или имеет нулевое значение. Например, код

```
<?php
if (isset($Str)) echo "Существует";
else echo "Нет";
echo "<br>";
if (empty($Str)) echo "Пустая";
else echo "Нет";
?>
```

вернет следующие значения:

```
Нет
Пустая
```

А если предварительно инициализировать переменную `$Str`, например, так:

```
<?php
$Str = "Строка";
if (isset($Str)) echo "Существует";
else echo "Нет";
echo "<br>";
if (empty($Str)) echo "Пустая";
else echo "Нет";
?>
```

то вывод программы будет отображен Web-браузером так:

```
Существует
Нет
```

5.9. Удаление переменной

Удалить переменную можно с помощью функции `unset()`:

```
unset(<Переменная>);
```

Эта функция необходима, если переменная использовалась при обработке данных большого объема и теперь не пужна.

Удаление переменной позволит освободить память компьютера.

```
<?php
$Str = "Строка";
if (isset($Str)) echo "Существует";
else echo "Нет";
unset($Str);
echo "<br>";
if (isset($Str)) echo "Существует";
else echo "Нет";
?>
```

Вывод программы:

```
Существует
Нет
```

5.10. Константы. Создание и использование констант

Константы служат для хранения значений, которые не должны изменяться во время работы программы. Создать константу можно с помощью функции `define()`:

```
define(<Имя константы>, <Значение константы>[, <Регистр>]);
```

Необязательный параметр `<Регистр>` может содержать значения `true` или `false`. Если указано `true`, то интерпретатор не будет учитывать регистр символов при

поиске константы по ее имени, если же задано `false` или параметр не указан, регистр символов существенен:

```
<?php
error_reporting(E_ALL);
define("author1", "Николай");
echo author1, '<br>'; // "Николай"
echo AUTHOR1, '<br><br>';
// Предупреждение о неопределенной константе AUTHOR1
define("author2", "Сергей", true);
echo author2, '<br>'; // "Сергей"
echo AUTHOR2, '<br><br>'; // "Сергей"
define("author3", "Иван", false);
echo author3, '<br>'; // "Иван"
echo AUTHOR3;
// Предупреждение о неопределенной константе AUTHOR3
?>
```

После объявления константы ее имя указывается в программе без знака `$`.

Для проверки существования константы используется функция `defined(<Имя константы>)`. Функция возвращает `true`, если константа объявлена:

```
<?php
define("author", "Николай", true);
if (defined("author")) echo "Объявлена";
else echo "Не объявлена";
?>
```

В PHP существуют встроенные константы:

- `__FILE__` (до и после два символа подчеркивания) — содержит имя файла с программой;
- `__LINE__` (до и после два символа подчеркивания) — содержит номер строки, которую обрабатывает интерпретатор в данный момент;
- `PHP_OS` — содержит имя и версию операционной системы;
- `PHP_VERSION` — содержит версию PHP.

```
<?php
echo __FILE__ . "<br>";
echo __LINE__ . "<br>";
echo PHP_OS . "<br>";
echo PHP_VERSION . "<br>";
?>
```

В итоге получим HTML-код, отображаемый так:

```
C:\Apache2\htdocs\index.php
3
WINNT
5.4.35
```

5.11. Операторы PHP

Операторы позволяют выполнить определенные действия с данными. Например, операторы присваивания служат для сохранения данных в переменной, математические операторы позволяют произвести арифметические вычисления, а оператор конкатенации строк используется для соединения двух строк в одну. Рассмотрим операторы, доступные в PHP, более подробно.

5.11.1. Математические операторы

- `+` — сложение:

```
$Z = $X + $Y;
```

- `-` — вычитание:

```
$Z = $X - $Y;
```

- `*` — умножение:

```
$Z = $X * $Y;
```

- `/` — деление:

```
$Z = $X / $Y;
```

- `%` — остаток от деления:

```
$Z = $X % $Y;
```

- `++` — оператор инкремента. Увеличивает значение переменной на 1:

```
$Z++; // Эквивалентно $Z = $Z + 1;
```

- `--` — оператор декремента. Уменьшает значение переменной на 1:

```
$Z--; // Эквивалентно $Z = $Z - 1;
```

Операторы инкремента и декремента можно записывать в постфиксной или префиксной формах:

```
$Z++; $Z--; // Постфиксная форма
++$Z; --$Z; // Префиксная форма
```

При постфиксной форме (`$Z++`) сначала возвращается значение переменной, а потом выполняется операция, а при префиксной форме (`++$Z`) — вначале осуществляется операция и только потом возвращается значение. Продемонстрируем это на примере (листинг 5.9).

Листинг 5.9. Постфиксная и префиксная форма

```
<?php
$X = 5;
$Z = $X++; // $Z = 5, $X = 6
echo "<b>Постфиксная форма (\$Z=\$X++;):</b><br> ";
echo "\$Z = $Z <br>\$X = $X <br><br>";
```

```
$X = 5;
$Z = ++$X; // $Z = 6, $X = 6
echo "<b>Префиксная форма (\$Z=++\$X;):</b><br> \$Z = $Z <br>\$X = $X";
?>
```

В итоге получим следующий результат:

```
Постфиксная форма ($Z=$X++;):
$Z = 5
$X = 6
Префиксная форма ($Z=++$X;):
$Z = 6
$X = 6
```

5.11.2. Операторы присваивания

- `=` — присваивает переменной значение:

```
$Z = 5;
```

- `+=` — увеличивает значение переменной на указанную величину:

```
$Z += 5; // Эквивалентно $Z = $Z + 5;
```

- `-=` — уменьшает значение переменной на указанную величину:

```
$Z -= 5; // Эквивалентно $Z = $Z – 5;
```

- `*=` — умножает значение переменной на указанную величину:

```
$Z *= 5; // Эквивалентно $Z = $Z * 5;
```

- `/=` — делит значение переменной на указанную величину:

```
$Z /= 5; // Эквивалентно $Z = $Z / 5;
```

- `%=` — делит значение переменной на указанную величину и возвращает остаток:

```
$Z %= 5; // Эквивалентно $Z = $Z % 5;
```

5.11.3. Двоичные операторы

- `~` — двоичная инверсия:

```
$Z=~$X;
```

- `&` — двоичное И:

```
$Z = $X & $Y;
```

- `|` — двоичное ИЛИ:

```
$Z = $X | $Y;
```

- `^` — двоичное исключающее ИЛИ:

```
$Z = $X ^ $Y;
```

- << — сдвиг влево — сдвигает двоичное представление числа влево на один или более разрядов и заполняет младшие разряды нулями:

```
$Z = $X << $Y;
```

- >> — сдвиг вправо — сдвигает двоичное представление числа вправо на один или более разрядов и заполняет старшие разряды содержимым самого старшего разряда:

```
$Z = $X >> $Y;
```

5.11.4. Оператор конкатенации строк. Подстановка значений переменных. Запуск внешних программ

Оператор . (точка) осуществляет конкатенацию строк, т. е. соединяет их в одну строку:

```
$Z = "Строка1" . "Строка2";
// Переменная $Z будет содержать значение "Строка1Строка2"
```

Очень часто необходимо сформировать строку, состоящую из имени переменной и ее значения. Если написать

```
$X = "Строка1";
$Z = "Значение равно $X";
```

то переменная `$Z` будет содержать значение `"Значение равно Строка1"`, а если написать так:

```
$X = "Строка1";
$Z = 'Значение равно $X';
```

то переменная `$Z` будет содержать значение `"Значение равно $X"`. Помните, что строка в кавычках и строка в апострофах вернет разные результаты. В последнем случае, для того чтобы получить значение переменной, можно воспользоваться операцией конкатенации строк:

```
$X = "Строка1";
$Z = 'Значение равно ' . $X;
```

Рассмотрим еще один пример. Предположим, нужно объединить два слова в одно. Одно из слов задано с помощью переменной. Если написать

```
$X = "авто";
$Z = "$Xтранспорт"; // $Z = "" или Notice: Undefined variable
```

то переменная `$Z` будет содержать пустую строку, т. к. переменная `$Xтранспорт` не определена. В этом случае можно воспользоваться следующими способами:

- использовать конкатенацию строк:

```
$X = "авто";
$Z = $X . "транспорт"; // $Z = "автотранспорт"
```

- указать имя переменной в фигурных скобках так:

```
$X = "авто";
$Z = "${X}транспорт"; // $Z = "автотранспорт"
```

или так:

```
$X = "авто";
$Z = "{$X}транспорт"; // $Z = "автотранспорт"
```

К любому символу строки можно обратиться как к элементу массива. Достаточно указать его индекс в квадратных скобках. Индексация начинается с нуля:

```
$X = "Привет";
echo $X[0]; // Выведет: П
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> В кодировке UTF-8 один символ может кодироваться несколькими байтами. По этой причине обратиться к символу как к элементу массива можно только после перекодировки.

Если в переменную нужно записать большой объем текста, это можно сделать способом, продемонстрированным в листинге 5.10.

Листинг 5.10. Запись в переменную большого объема текста

```
<?php

$Y=<<<Metka1
<html>
<head>
<title>Строки</title>
</head>
<body>
Metka1;

echo $Y;
$X = "Привет";
echo $X[0]; // выведет "П"
?>
</body>
</html>
```

В данном примере многострочный текст располагается между метками (`Metka1`):

```
$Y=<<<Metka1
...
Metka1;
```

Название метки может быть любым. Вторая (закрывающая) метка должна быть написана с начала строки, и после нее должна стоять точка с запятой.

Если содержимое строки заключить в обратные кавычки, то это позволит запустить внешнюю программу и присвоить переменной результат ее работы (листинг 5.11).

Листинг 5.11. Запуск внешней программы

```
<?php
$X = `dir`;
echo '<textarea cols="70" rows="30">';
echo convert_cyr_string($X, "d", "w");
echo '</textarea>';
?>
```

Данный код выведет содержимое папки C:\Apache2\htdocs. При выводе используется кодировка Dos (кодовая страница 866), поэтому русские буквы будут искажены. Чтобы избежать этого, мы преобразуем кодировку с помощью функции `convert_cyr_string()`.

5.11.5. Приоритет выполнения операторов

В какой последовательности будет вычисляться приведенное далее выражение?

```
$X = 5 + 10 * 3 / 2;
```

Это зависит от приоритета выполнения операторов. В данном случае последовательность вычисления выражения будет следующей:

1. Число 10 будет умножено на 3, т. к. приоритет оператора умножения выше приоритета оператора сложения.
2. Полученное значение будет поделено на 2, поскольку приоритет оператора деления равен приоритету оператора умножения (а операторы с равными приоритетами выполняются слева направо), но выше чем у оператора сложения.
3. К полученному значению будет прибавлено число 5, т. к. оператор присваивания = имеет наименьший приоритет.
4. Значение будет присвоено переменной `$X`.

С помощью скобок можно изменить последовательность вычисления выражения:

```
$X = (5 + 10) * 3 / 2;
```

Теперь порядок вычислений будет другим:

1. К числу 5 будет прибавлено 10.
2. Полученное значение будет умножено на 3.
3. Полученное значение будет поделено на 2.
4. Значение будет присвоено переменной `$X`.

Перечислим операторы в порядке убывания приоритета:

1. `++`, `--` — инкремент, декремент.
2. `*`, `/`, `%`, — умножение, деление, остаток от деления.

3. +, - — сложение, вычитание.
4. <<, >> — двоичный сдвиг.
5. & — двоичное И.
6. ^ — двоичное исключающее ИЛИ.
7. | — двоичное ИЛИ.
8. =, +=, -=, *=, /=, %= — присваивание.

5.12. Преобразование типов данных

Что получится, если к числу прибавить строку?

```
$Str = "5"; // Строка
$Number = 3; // Число
$Str2 = $Number + $Str; // Переменная содержит число 8
$Str3 = $Str + $Number; // Переменная содержит число 8
```

Результат будет абсолютно другим, нежели в JavaScript, т. к. оператор + в PHP не используется для конкатенации строк. В этом случае интерпретатор попытается преобразовать переменные к одному типу данных, а затем выполнить операцию. В нашем примере переменная `$Str`, имеющая тип `string` (строка), будет преобразована к тину `integer` (число), а затем будет выполнено сложение двух чисел.

Но что произойдет, если строку невозможно преобразовать в число?

```
$Str = "Привет"; // Строка
$Number = 3; // Число
$Str2 = $Number + $Str; // Переменная содержит число 3
$Str3 = $Str + $Number; // Переменная содержит число 3
```

Как видим, строка, не содержащая числа, преобразуется к числу 0. А что будет, если из числа вычесть строку, число умножить на строку или число разделить на строку?

```
$Number = 15;
$Str = "5";
$Str2 = $Number - $Str; // Переменная содержит число 10
$Str3 = $Number * $Str; // Переменная содержит число 75
$Str4 = $Number / $Str; // Переменная содержит число 3
```

Итак, интерпретатор попытается преобразовать строку в число, а затем вычислить выражение.

В какой последовательности будут указаны число и строка, не важно:

```
$Str5 = $Str * $Number; // Переменная все равно содержит число 75
```

Но что получится, если в строке будут одни буквы?

```
$Number = 15;
$Str = "Строка";
```

```
$Str2 = $Number — $Str; // Переменная содержит число 15
$Str3 = $Str — $Number; // Переменная содержит число -15
$Str4 = $Number * $Str; // Переменная содержит число 0
$Str5 = $Str * $Number; // Переменная содержит число 0
$Str6 = $Number / $Str; // Ошибка деления на 0
$Str7 = $Str / $Number; // Переменная содержит число 0
```

С одной стороны, хорошо, что интерпретатор выполняет преобразование типов данных за нас. Но с другой стороны, можно получить результат, который вовсе не планировался. Поэтому лучше оперировать переменными одного типа, а при необходимости преобразования типов делать это самостоятельно.

Для преобразования типов данных можно вызвать функцию `settype()`, которая преобразует тип переменной в указанный:

```
settype(<Переменная>, <Тип>);
```

Пример:

```
$Number = 15;
$Str = "5";
settype($Number, "string");
settype($Str, "integer");
```

Можно также воспользоваться приведением типов. Для этого перед переменной в круглых скобках указывается тип, к которому нужно преобразовать значение переменной.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> **В отличие от функции `settype()` приведение типов не меняет тип исходной переменной.**

```
$Str = "5"; // Строка
$Number = 3; // Число
$Str2 = $Number + (integer)$Str; // Переменная содержит число 8
echo gettype($Str); // выведет string
```

Такой же результат можно получить при использовании следующих функций:

```
intval(<Переменная>);
doubleval(<Переменная>);
strval(<Переменная>);
```

Пример:

```
$Str = "5"; // Строка
$Number = 3; // Число
$Str2 = $Number + intval($Str); // Переменная содержит число 8
echo gettype($Str); // выведет string
```

5.13. Специальные символы

Специальные символы — это комбинации знаков, обозначающих служебные или непечатаемые символы, которые невозможно вставить обычным способом.

Перечислим специальные символы, доступные в PHP в строках, ограниченных двойными кавычками:

- `\n` — перевод строки;
- `\r` — возврат каретки;
- `\t` — знак табуляции;
- `\"` — кавычка;
- `\$` — знак доллара;
- `\\` — обратная косая черта.

Кроме того, можно задавать символы восьмеричными или шестнадцатеричными кодами, записывая код символа после обратной косой черты или символов `\x`, соответственно, например `"\40"` и `"\x20"` — строки, состоящие из одного пробела (символа с кодом 32).

В строках, ограниченных апострофами, эти символы не работают. В них доступны только два специальных символа:

- `\'` — апостроф;
- `\\` — обратная косая черта.

5.14. Массивы

Массив — это нумерованный набор переменных. Переменная в массиве называется *элементом* массива, а ее позиция в массиве задается *индексом*. Нумерация элементов массива начинается с нуля, а не с единицы. Это следует помнить. Общее число элементов в массиве называется *размером* массива.

Массивы, индексами которых являются числа, часто называют *списками*.

5.14.1. Инициализация массива

Массив инициализируют двумя способами:

- поэлементно:

```
$Mass[0] = 'Ноль';
$Mass[1] = 'Один';
$Mass[2] = 'Два';
$Mass[3] = 'Три';
```

Кроме того, можно не указывать индекс. PHP автоматически присвоит элементу индекс, на единицу больший последнего, т. е. добавит элемент в конец массива:

```
$Mass[] = 'Ноль';
$Mass[] = 'Один';
$Mass[] = 'Два';
$Mass[] = 'Три';
```

- указав все элементы массива сразу:

```
$Mass = array('Ноль', 'Один', 'Два', 'Три');
```

В PHP 5.4 стал доступен сокращенный синтаксис в стиле JavaScript:

```
$Mass = ['Ноль', 'Один', 'Два', 'Три'];
```

5.14.2. Получение и изменение элемента массива. Определение числа элементов массива

Обращение к элементам массива осуществляется с помощью квадратных скобок, в которых указывается индекс элемента. Нумерация элементов массива начинается с нуля:

```
$Mass = array('Ноль', 'Один', 'Два', 'Три');
$var = $Mass[1]; // Переменной $var будет присвоено значение "Один"
```

Обратиться к элементам массива можно с помощью инструкции `list()`:

```
$Mass[] = 'Ноль';
$Mass[] = 'Один';
$Mass[] = 'Два';
$Mass[] = 'Три';
list($var1, $var2, $var3, $var4) = $Mass;
echo $var2; // Переменной $var2 будет присвоено значение "Один"
```

При желании можно добавить новый элемент массива или изменить значение существующего:

```
$Mass[] = 'Четыре';
$Mass[0] = 'Нуль';
```

Получить число элементов массива позволяют функции `count()` и `sizeof()`:

```
$Mass = array('Ноль', 'Один', 'Два');
echo count($Mass); // Выведет: 3
echo sizeof($Mass); // Выведет: 3
```

5.14.3. Многомерные массивы

Любому элементу массива можно присвоить другой массив:

```
$Mass = array();
$Mass[0] = array(1, 2, 3, 4);
```

В этом случае получить значение массива можно, указав два индекса:

```
$var = $Mass[0][2]; // Переменной $var будет присвоено значение 3
```

5.14.4. Ассоциативные массивы

Основное отличие ассоциативных массивов от списков — возможность обращения к элементу массива не по числовому индексу, а по индексу, представляющему собой строку. Индексы ассоциативного массива называются *ключами*. Пример ассоциативного массива:

```
$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
echo $Mass['Один']; // Выведет: 1
```

Кроме перечисления, для инициализации ассоциативных массивов используется инструкция `array()`:

```
$Mass = array('Один' => 1, 'Два' => 2, 'Три' => 3);
echo $Mass['Один']; // Выведет: 1
```

И здесь возможен сокращенный синтаксис:

```
$Mass = ['Один' => 1, 'Два' => 2, 'Три' => 3];
```

Любой из этих подходов — с инструкцией `array()` или с применением сокращенного синтаксиса — удобно использовать для создания многомерных ассоциативных массивов:

```
$Mass['Иванов'] = array('Имя' => 'Иван', 'Отчество' => 'Иванович',
'Год рождения' => 1966);
$Mass['Семенов'] = ['Имя' => 'Сергей', 'Отчество' => 'Николаевич',
'Год рождения' => 1980];
```

Существует и другой способ:

```
$Mass = array(
   'Иванов' => array('Имя' => 'Иван', 'Отчество' => 'Иванович',
      'Год рождения' => 1966),
   'Семенов' => array('Имя' => 'Сергей', 'Отчество' => 'Николаевич',
      'Год рождения' => 1980)
);
```

Доступ к элементу такого массива осуществляется путем указания двух ключей:

```
echo $Mass['Иванов']['Год рождения']; // Выведет: 1966
```

Функции `array_keys()` и `array_values()` позволяют получить все ключи и все значения ассоциативного массива соответственно:

```
$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass2 = array_keys($Mass);
// Выводим ключи массива
foreach($Mass2 as $key) {
   echo $key . '<br>';
} // Выведет: Один<br>Два<br>Три<br>
```

```
$Mass3 = array_values($Mass);
// Выводим значения массива
foreach($Mass3 as $key) {
   echo $key . '<br>';
} // Выведет: 1<br>2<br>3<br>
```

5.14.5. Слияние массивов

Для слияния двух ассоциативных массивов предусмотрен оператор +:

```
$Mass1['Один'] = 1;
$Mass1['Два'] = 2;
$Mass2['Три'] = 3;
$Mass2['Четыре'] = 4;
$Mass3 = $Mass1 + $Mass2;
print_r($Mass3); // Выводим массив
```

В этом примере массив `$Mass3` будет содержать все элементы массивов `$Mass1` и `$Mass2`:

```
Array ( [Один] => 1 [Два] => 2 [Три] => 3 [Четыре] => 4 )
```

Для слияния двух списков оператор + не подходит. В этом случае используется функция `array_merge()`:

```
$Mass1[] = 'Один';
$Mass1[] = 'Два';
$Mass2[] = 'Три';
$Mass2[] = 'Четыре';
$Mass3 = array_merge($Mass1, $Mass2);
print_r($Mass3); // Выводим массив
```

После этого массив `$Mass3` будет содержать все элементы массивов `$Mass1` и `$Mass2`:

```
Array ( [0] => Один [1] => Два [2] => Три [3] => Четыре )
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Если один из параметров в функции `array_merge()` не является массивом, интерпретатор выведет сообщение об ошибке.

5.14.6. Перебор элементов массива

Для перебора массивов применяются три вида циклов: `for`, `foreach` и `while`.

Цикл `for` используется, например, так:

```
$Mass[] = 'Один';
$Mass[] = 'Два';
$Mass[] = 'Три';
$Mass[] = 'Четыре';
$count = count($Mass);
```

```
for ($i=0; $i<$count; $i++) {
   echo $Mass[$i] . '<br>';
}
```

Следует с осторожностью пользоваться циклом `for`, т. к. функция `count()` возвращает число существующих элементов массива. Если элемент не определен, то он не учитывается в подсчете. Например, следующий код выведет не все элементы массива:

```
// Отключаем вывод предупреждающих сообщений
error_reporting(E_ALL & ~E_NOTICE);
$Mass[1] = 'Один';
$Mass[2] = 'Два';
$Mass[3] = 'Три';
echo count($Mass); // Выведет: 3
echo '<br>';
$count = count($Mass);
for ($i=0; $i<$count; $i++) {
   echo $Mass[$i] . '<br>';
} // Выведет: <br><br>Один<br>Два<br>
```

Как видно из примера, мы не получили значение элемента с индексом 3.

Для перебора ассоциативного массива такой способ не подходит, т. к. индексом является не число, а строка. Вместо этого применяются другие конструкции, например:

```
$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass['Четыре'] = 4;
for (reset($Mass); ($key = key($Mass)); next($Mass)) {
   echo $key . ' =&gt; ' . $Mass[$key] . '<br>';
}
```

В этом случае мы воспользовались следующими функциями:

- `reset()` — устанавливает указатель на первый элемент массива;
- `next()` — перемещает указатель на один элемент массива вперед;
- `key()` — возвращает ключ текущего элемента массива.

Для перебора элементов ассоциативного массива в обратном порядке предназначены другие функции:

- `end()` — устанавливает указатель на последний элемент массива;
- `prev()` — перемещает указатель на один элемент массива назад.

Кроме того, для получения текущего значения элемента массива можно использовать функцию `current()`:

```
$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass['Четыре'] = 4;
```

```
for (reset($Mass); ($key = key($Mass)); next($Mass)) {
   echo $key . ' =&gt; ' . current($Mass). '<br>';
}
```

Разумеется, цикл `for` можно употреблять не только для работы с массивами, но и для других целей. А вот цикл `foreach` предназначен исключительно для работы с массивами. Он позволяет работать как с обычными массивами, например:

```
$Mass[] = 'Один';
$Mass[] = 'Два';
$Mass[] = 'Три';
$Mass[] = 'Четыре';
foreach ($Mass as $key) {
   echo $key . '<br>';
}
```

так и с ассоциативными:

```
$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass['Четыре'] = 4;
foreach ($Mass as $key => $value) {
   echo $key . ' =&gt; ' . $value . '<br>';
}
```

Цикл `while` также подходит для работы с массивами. Обычно это делается с использованием сочетания функций `list()` и `each()`:

```
$Mass[] = 'Один';
$Mass[] = 'Два';
$Mass[] = 'Три';
$Mass[] = 'Четыре';
while (list(, $value) = each($Mass)) {
   echo $value . '<br>';
}

$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass['Четыре'] = 4;
while (list($key, $value) = each($Mass)) {
   echo $key . ' =&gt; ' . $value . '<br>';
}
```

Функция `each()` возвращает текущий элемент массива (пару "ключ/значение"), после чего перемещает указатель.

Перебор элементов массива без использования циклов

До сих пор мы выводили содержимое массивов с помощью циклов. Того же эффекта можно достичь при использовании функции `array_walk()`. Она позволяет после-

довательно применять созданную нами функцию ко всем элементам массива. Например, вывод всех элементов массива будет выглядеть так:

```
function f_print($value, $key) {
   echo $key . ' =&gt; ' . $value . '<br>';
}

$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass['Четыре'] = 4;
array_walk($Mass, "f_print");
// Выведет: Один => 1<br>Два => 2<br>Три => 3<br>Четыре => 4<br>
```

Или, например, можно изменить значения всех элементов массива, скажем, прибавив к ним число 10:

```
$var = 10;
function f_change(&$value, $key, $arr) {
   $arr[$key] += $var;
}
function f_print($value, $key) {
   echo $key . ' =&gt; ' . $value . '<br>';
}

$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass['Четыре'] = 4;
array_walk($Mass, "f_change", $Mass);
array_walk($Mass, "f_print");
// Выведет: Один => 11<br>Два => 12<br>Три => 13<br>Четыре => 14<br>
```

В качестве третьего параметра в функцию `array_walk()` можно передать значение, которое будет передано третьим же параметром в функцию, обрабатывающую элементы массива. Мы воспользовались этой особенностью, чтобы передать в эту функцию сам обрабатываемый массив — так мы сможем без проблем изменить любой его элемент. (Значение, переданное в функцию, мы изменить не можем, т. к. по завершению ее работы оно будет потеряно.)

5.14.7. Добавление и удаление элементов массива

Для добавления и удаления элементов массива предусмотрены следующие функции:

- `array_unshift(<Массив>, <Элемент>)` — добавляет элементы в начало массива:

```
$Mass[0] = 'Три';
$Mass[1] = 'Четыре';
```

```
array_unshift($Mass, 'Один', 'Два');
print_r($Mass);
// Array ( [0] => Один [1] => Два [2] => Три [3] => Четыре )
```

- конструкция `<Массив>[]` — добавляет элементы в конец массива:

```
$Mass[0] = 'Один';
$Mass[1] = 'Два';
$Mass[] = 'Три';
print_r($Mass);
// Array ( [0] => Один [1] => Два [2] => Три )
```

- `array_push(<Массив>, <Элемент>)` — добавляет элементы в конец массива:

```
$Mass[0] = 'Один';
$Mass[1] = 'Два';
array_push($Mass, 'Три', 'Четыре');
print_r($Mass);
// Array ( [0] => Один [1] => Два [2] => Три [3] => Четыре )
```

- `array_shift(<Массив>)` — удаляет первый элемент массива и возвращает его:

```
$Mass[0] = 'Один';
$Mass[1] = 'Два';
echo array_shift($Mass) . "<br>\n"; // Выведет: Один<br>
print_r($Mass);
// Array ( [0] => Два )
```

- `array_pop(<Массив>)` — удаляет последний элемент массива и возвращает его:

```
$Mass[0] = 'Один';
$Mass[1] = 'Два';
echo array_pop($Mass) . "<br>\n"; // Выведет: Два<br>
print_r($Mass);
// Array ( [0] => Один )
```

- `array_unique(<Массив>)` — возвращает ассоциативный массив, состоящий из уникальных значений указанного ассоциативного массива:

```
$Mass = array('Один' => 1, 'Два' => 2, 'Один' => 1,
              'Три' => 1, 'Четыре' => 4);
$Mass2 = array_unique($Mass);
print_r($Mass2);
// Array ( [Один] => 1 [Два] => 2 [Четыре] => 4 )
```

5.14.8. Переворачивание и перемешивание массива

Функция `array_reverse()` возвращает массив, элементы которого следуют в обратном порядке относительно исходного массива:

```
$Mass = array('Один', 'Два', 'Три', 'Четыре');
$Mass = array_reverse($Mass);
print_r($Mass);
// Array ( [0] => Четыре [1] => Три [2] => Два [3] => Один )
```

Функция `shuffle()` "перемешивает" массив. Элементы массива будут расположены в случайном порядке:

```
$Mass = array('Один', 'Два', 'Три', 'Четыре');
shuffle($Mass);
print_r($Mass);
// Array ( [0] => Два [1] => Один [2] => Три [3] => Четыре )
```

5.14.9. Сортировка массива. Создание пользовательской сортировки

Функция `sort()` позволяет отсортировать список в алфавитном порядке, а функция `rsort()` — в обратном порядке:

```
$Mass = array('Один', 'Два', 'Три', 'Четыре');
sort($Mass);
print_r($Mass);
// Array ( [0] => Два [1] => Один [2] => Три [3] => Четыре )
rsort($Mass);
print_r($Mass);
// Array ( [0] => Четыре [1] => Три [2] => Один [3] => Два )
```

Для сортировки ассоциативных массивов эти функции не применяются, т. к. они разрывают связь ключа со значением. Отсортировать ассоциативный массив можно или по ключам, или по значениям. Для этого используются следующие функции:

- `asort()` — сортировка по значениям в алфавитном порядке;
- `arsort()` — сортировка по значениям в обратном порядке:

```
$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass['Четыре'] = 4;
arsort($Mass);
print_r($Mass);
// Array ( [Четыре] => 4 [Три] => 3 [Два] => 2 [Один] => 1 )
```

- `ksort()` — сортировка по ключам в алфавитном порядке;
- `krsort()` — сортировка по ключам в обратном порядке:

```
$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass['Четыре'] = 4;
krsort($Mass);
print_r($Mass);
// Array ( [Четыре] => 4 [Три] => 3 [Один] => 1 [Два] => 2 )
```

Если нужно изменить порядок стандартной сортировки, можно задать свою сортировку с помощью следующих функций:

- `usort()` — для пользовательской сортировки списков;
- `uksort()` — для пользовательской сортировки ассоциативных массивов по ключам;
- `uasort()` — для пользовательской сортировки ассоциативных массивов по значениям.

В качестве первого аргумента этим функциям передается массив, а второй аргумент должен содержать имя функции, сравнивающей два элемента. Функция сравнения принимает две переменные и должна возвращать:

- `1` — если первый больше второго;
- `-1` — если второй больше первого;
- `0` — если элементы равны.

Например, стандартная сортировка зависит от регистра символов:

```
$Mass = array('единица1', 'Единый', 'Единица2');
sort($Mass);
print_r($Mass);
// Array ( [0] => Единица2 [1] => Единый [2] => единица1 )
```

В результате мы получим неправильную сортировку, ведь `Единый` и `Единица2` больше `единица1`. Изменим стандартную сортировку на свою собственную, не учитывающую регистр (листинг 5.12).

Листинг 5.12. Сортировка без учета регистра

```
function f_sort($Str1, $Str2) { // Сортировка без учета регистра
   $Str1_1 = strtolower($Str1); // Преобразуем к нижнему регистру
   $Str2_1 = strtolower($Str2); // Преобразуем к нижнему регистру
   if ($Str1_1>$Str2_1) return 1;
   if ($Str1_1<$Str2_1) return -1;
   return 0;
}

setlocale(LC_CTYPE, "ru_RU.CP1251"); // Настройка локали
$Mass = array('единица1', 'Единый', 'Единица2');
usort($Mass, "f_sort");
print_r($Mass);
// Array ( [0] => единица1 [1] => Единица2 [2] => Единый )
```

Для получения правильной сортировки мы приводим две переменные к одному регистру, а затем осуществляем стандартное сравнение. Заметьте, что регистр самих элементов массива не изменяется, т. к. мы работаем с их копиями. Для правильной работы функции `strtolower()` с русским языком необходимо настроить локаль. Это

позволяет сделать функция `setlocale()`. Более подробно мы рассмотрим функцию `setlocale()` при изучении функций обработки строк *(см. разд. 5.15.2)*.

5.14.10. Получение части массива

Получить часть массива позволяет функция `array_slice()`, имеющая следующий формат вызова:

```
array_slice(<Массив>, <Начальная позиция>[, <Число элементов>]);
```

Функции передаются следующие параметры:

- `<Массив>` — исходный массив;
- `<Начальная позиция>` — число элементов от начала массива, которое нужно пропустить;
- `<Число элементов>` — число элементов, которое нужно получить из исходного массива. Если параметр опущен, то элементы выбираются до конца массива.

Пример:

```
$Mass = array('Один', 'Два', 'Три', 'Четыре', 'Пять');
$Mass2 = array_slice($Mass, 2, 3);
print_r($Mass2);
// Array ( [0] => Три [1] => Четыре [2] => Пять )
```

Полученную часть массива можно заменить одним элементом или массивом элементов с помощью функции `array_splice()`. Вызов функции осуществляется так:

```
array_splice(<Массив>, <Начальная позиция>, <Число элементов>,
      <Добавляемый массив>);
```

Первые три параметра имеют такое же значение, как и у функции `array_slice()`. Четвертый параметр `<Добавляемый массив>` — один элемент или массив элементов, добавляемый вместо выбранных элементов:

```
$Mass1 = array('Один', 'Два', 'Три', 'Четыре', 'Пять');
$Mass2 = array('3', '4', '5');
array_splice($Mass1, 2, 3, $Mass2);
print_r($Mass1);
// Array ( [0] => Один [1] => Два [2] => 3 [3] => 4 [4] => 5 )
```

5.14.11. Преобразование переменных в массив

Функция `compact()` позволяет преобразовать переменные в ассоциативный массив. Ключами становятся имена переменных, а значениями — значения переменных:

```
$var1 = 1;
$var2 = 2;
$var3 = 3;
```

```
$Mass = compact('var1', 'var2', 'var3');
print_r($Mass);
// Array ( [var1] => 1 [var2] => 2 [var3] => 3 )
```

5.14.12. Преобразование массива в переменные

Функция `extract()` создает переменные с именами, соответствующими именам ключей, и значениями, соответствующими значениям элемента ассоциативного массива. Формат вызова функции:

```
extract(<Массив>, [<Способ>], [<Префикс>]);
```

Можно указывать следующие параметры:

- `<Массив>` — исходный ассоциативный массив;
- `<Способ>` — способ обработки конфликтных ситуаций. Может принимать следующие значения:
  - `EXTR_OVERWRITE` — если переменная существует, то ее значение перезаписывается (значение по умолчанию);
  - `EXTR_SKIP` — если переменная существует, то элемент массива пропускается;
  - `EXTR_PREFIX_SAME` — если переменная существует, то перед именем переменной будет добавлен префикс, указанный в параметре `<Префикс>`;
  - `EXTR_PREFIX_ALL` — перед именем всех переменных будет добавлен префикс, указанный в параметре `<Префикс>`;
  - `EXTR_IF_EXISTS` — извлекает значения только тех переменных, которые уже существуют;
  - `EXTR_REFS` — извлекает переменные как ссылки.

Пример:

```
$var1 = 'Привет';
$Mass = array('var1' => 'value1', 'var2' => 'value2', 'var3' => 'value3');
extract($Mass, EXTR_PREFIX_SAME, 's');
echo "$var1 $s_var1 $var2 $var3";
// Выведет: Привет value1 value2 value3
```

Так как переменная `$var1` существует, то перед именем создаваемой переменной будет добавлен префикс `s_`. Все остальные ключи будут преобразованы в одноименные переменные.

5.14.13. Заполнение массива числами

Создать массив, содержащий диапазон чисел, можно либо с помощью цикла, либо с помощью функции `range()`. Функция имеет следующий формат:

```
range(<Начало диапазона>, <Конец диапазона>);
```

Предположим, необходимо создать массив, состоящий из диапазона чисел от 1 до 100:

```
$Mass = range(1, 100);
foreach ($Mass as $key) {
  echo $key . '<br>';
} // Выведет числа от 1 до 100, разделенные тегами <br>
```

5.14.14. Преобразование массива в строку

Преобразовать массив в строку можно с помощью нескольких функций:

- `implode()` — преобразует массив в строку. Элементы добавляются через указанный разделитель:

```
$Mass = array('Фамилия', 'Имя', 'Отчество');
$str = implode(' - ', $Mass);
echo $str; // Выведет: Фамилия - Имя - Отчество
```

- `join()` — полностью аналогична функции `implode()`;
- `serialize()` позволяет преобразовать любой массив в строку специального формата:

```
$Mass = array('Фамилия', 'Имя', 'Отчество');
$str = serialize($Mass);
echo $str;
// a:3:{i:0;s:7:"Фамилия";i:1;s:3:"Имя";i:2;s:8:"Отчество";}
```

- `unserialize()` — используется для восстановления массива из строки, преобразованной с помощью функции `serialize()`:

```
$Mass = array('Фамилия', 'Имя', 'Отчество');
$str = serialize($Mass);
$Mass2 = unserialize($str);
print_r($Mass2);
// Array ( [0] => Фамилия [1] => Имя [2] => Отчество )
```

- `print_r()` — позволяет вывести структуру массива:

```
$Mass = array('Один', 'Два', 'Три');
echo '<pre>';
print_r($Mass);
echo '</pre>';
```

Выведет:

```
Array
(
    [0] => Один
    [1] => Два
    [2] => Три
)
```

- `var_dump()` — выводит подробную информацию о структуре массива:

```
$Mass = array('Один', 2, 'Три');
echo '<pre>';
```

```
var_dump($Mass);
echo '</pre>';
```

Выведет:

```
array(3) {
  [0]=>
  string(4) "Один"
  [1]=>
  int(2)
  [2]=>
  string(3) "Три"
}
```

ОБРАТИТЕ ВНИМАНИЕ

Функции `print_r()` и `var_dump()` позволяют выводить не только структуру массивов, но и значения других переменных. Поэтому функции часто применяются на этапе отладки программы.

5.14.15. Проверка наличия значения в массиве

Функция `in_array()` позволяет проверить наличие значения в массиве. Возвращает `true`, если значение присутствует. Формат функции:

```
in_array(<Что ищем>, <Массив>[, <Тип>]);
```

Параметр `<Что ищем>` может быть числом, строкой или массивом. Следует также заметить, что сравнение проводится с учетом регистра символов. Если необязательный параметр `<Тип>` имеет значение `true`, то дополнительно выполняется проверка соответствия типов данных.

Пример:

```
$arr = array('один', '1', 20);
if (in_array('один', $arr)) echo 'Есть';
else echo 'Нет';
// Выведет: Есть
if (in_array('Один', $arr)) echo 'Есть';
else echo 'Нет';
// Выведет Нет, т. к. не совпадает регистр символов
if (in_array('1', $arr)) echo 'Есть';
else echo 'Нет';
// Выведет: Есть
if (in_array(20, $arr, true)) echo 'Есть';
else echo 'Нет';
// Выведет: Есть
if (in_array('20', $arr, true)) echo 'Есть';
else echo 'Нет';
// Выведет Нет, т. к. не совпадают типы данных
```

5.15. Строки

Разрабатывая Web-сайты, часто приходится проводить манипуляции со строками. Поэтому необходимо знать и уметь использовать встроенные функции PHP, предназначенные для обработки строк. Например, перед добавлением сообщения в гостевую киигу можно удалить лишние пробелы и все теги из строки, добавить защитные слэши перед специальными символами или заменить их HTML-эквивалентами и т. д.

5.15.1. Функции для работы со строками

Перечислим основные функции для работы со строками:

- `strlen()` — возвращает число символов в строке:

```
$str = "Строка";
echo strlen($str); // Выведет: 6
```

- `trim()` — удаляет пробельные символы в начале и конце строки. Пробельными символами считаются: пробел, символ перевода строки (`\n`), символ возврата каретки (`\r`), символы горизонтальной (`\t`) и вертикальной (`\v`) табуляции и символ конца строки (`\0`):

```
$str = ' Строка ';
$str = trim($str);
echo "'$str'"; // Выведет: 'Строка'
```

- `ltrim()` — удаляет пробельные символы в начале строки:

```
$str = ' Строка ';
$str = ltrim($str);
echo "'$str'"; // Выведет: 'Строка '
```

- `rtrim()` — удаляет пробельные символы в конце строки:

```
$str = ' Строка ';
$str = rtrim($str);
echo "'$str'"; // Выведет: ' Строка'
```

- `chop()` — удаляет пробельные символы в конце строки:

```
$str = ' Строка ';
$str = chop($str);
echo "'$str'"; // Выведет: ' Строка'
```

- `strip_tags()` — удаляет из строки все HTML-теги, за исключением указанных во втором параметре:

```
$str = '<span style="color: red"><b>Строка</b></span>';
$str1 = strip_tags($str);
$str2 = strip_tags($str, '<b>');
echo $str1 . '<br>'; // Выведет: Строка<br>
echo $str2; // Выведет: <b>Строка</b>
```

Следует заметить, что функция `strip_tags()` работает не совсем корректно. Если в строке встретится открывающая угловая скобка ("<"), а за ней сразу другой символ, то будет удален весь фрагмент от скобки до конца строки:

```
$str = '5<10 Эта строка будет удалена!';
$str1 = strip_tags($str);
echo $str1; // Выведет: 5
```

- `addslashes()` — добавляет обратную косую черту для защиты специальных символов:

```
$str = '"Волга", "Москвич", "Жигули"';
$str = addslashes($str);
echo $str; // Выведет: \"Волга\", \"Москвич\", \"Жигули\"
```

- `stripslashes()` — удаляет символы обратной косой черты:

```
$str = '\\"Волга\\", \\"Москвич\\", \\"Жигули\\"';
$str = stripslashes($str);
echo $str; // Выведет: "Волга", "Москвич", "Жигули"
```

- `htmlspecialchars(<Строка>, [<Режим>], [<Кодировка>])` — заменяет специальные символы их HTML-эквивалентами. Второй необязательный параметр `<Режим>` задает режим преобразования двойных и одинарных кавычек. Может принимать следующие значения:

  - `ENT_COMPAT` — преобразуются только двойные кавычки;
  - `ENT_QUOTES` — преобразуются и двойные, и одинарные кавычки;
  - `ENT_NOQUOTES` — двойные и одинарные кавычки не заменяются;
  - `ENT_IGNORE` — отбрасывать некорректные кодовые последовательности;
  - `ENT_SUBSTITUTE` — заменять некорректные кодовые последовательности символом `U+FFFD` в случае использования UTF-8 или `&#FFFD;` при использовании другой кодировки;
  - `ENT_HTML401` — обрабатывать код по правилам HTML 4.01;
  - `ENT_HTML5` — обрабатывать код по правилам HTML 5.

  Режим обработки строк по умолчанию — `ENT_COMPAT | ENT_HTML401`, кодировка по умолчанию — UTF-8:

```
$str = '"Волга", "Москвич"';
$str = htmlspecialchars($str);
// Для строки в кодировке UTF-8 такое выражение:
// $str = htmlspecialchars($str, ENT_COMPAT | ENT_HTML5,
'UTF-8');
echo $str;
// Выведет: &quot;Волга&quot;, &quot;Москвич&quot;
```

- `explode()` — разделяет строку на подстроки по указанному разделителю и добавляет полученные подстроки в массив:

```
$str = "Фамилия\tИмя\tОтчество\tГод рождения";
$Mass = explode("\t", $str);
foreach ($Mass as $key) {
   echo $key . '<br>';
} // Выведет: Фамилия<br>Имя<br>Отчество<br>Год рождения<br>
```

- `substr()` — возвращает подстроку указанной длины, начиная с заданной позиции. Если длина не указана, то возвращается подстрока, начиная с заданной позиции и до конца строки. Формат функции:

```
substr(<Строка>, <Начальная позиция>, [<Длина>]);
```

Примеры:

```
$str = "Строка";
$str1 = substr($str, 0, 1);
echo $str1; // Выведет: С
$str2 = substr($str, 1);
echo $str2; // Выведет: трока
```

- `wordwrap()` — позволяет разбить длинный текст на строки указанной длины. Формат функции:

```
wordwrap(<Строка>, <Число символов>, <Символ разрыва>);
```

Следующий пример

```
$str = "Очень длинная строка перед выводом";
echo wordwrap($str, 7, "<br>");
```

выведет каждое слово на отдельной строчке:

```
Очень
длинная
строка
перед
выводом
```

- `nl2br()` — добавляет перед всеми символами новой строки (`\n`) тег `<br />` (XML-аналог HTML-тега `<br>`):

```
$str = "Очень\nдлинная\nстрока\nперед\nвыводом";
echo nl2br($str);
```

Исходный HTML-код, выведенный этим кодом PHP, будет выглядеть следующим образом:

```
Очень<br />
длинная<br />
строка<br />
перед<br />
выводом
```

Если во втором параметре указать значение `false`, то будет добавляться HTML-тег `<br>`:

```
$str = "Строка1\nСтрока2";
echo nl2br($str, false);
// Строка1<br>
// Строка2
```

В окне Web-браузера каждое слово будет отображено в отдельной строке;

- `strtoupper()` — преобразует все буквы в строке к верхнему регистру:

```
$str = "очень длинная строка";
setlocale(LC_CTYPE, "ru_RU.CP1251"); // Настройка локали
echo strtoupper($str); // Выведет: ОЧЕНЬ ДЛИННАЯ СТРОКА
```

- `strtolower()` — преобразует все буквы в строке к нижнему регистру:

```
$str = "ОЧЕНЬ длинная строка";
setlocale(LC_CTYPE, "ru_RU.CP1251"); // Настройка локали
echo strtolower($str); // Выведет: очень длинная строка
```

- `ucfirst()` — преобразует первую букву строки к верхнему регистру:

```
$str = "очень длинная строка";
setlocale(LC_CTYPE, "ru_RU.CP1251"); // Настройка локали
echo ucfirst($str); // Выведет: Очень длинная строка
```

- `ucwords()` — преобразует первые буквы всех слов к верхнему регистру:

```
$str = "очень длинная строка";
setlocale(LC_CTYPE, "ru_RU.CP1251"); // Настройка локали
echo ucwords($str); // Выведет: Очень Длинная Строка
```

5.15.2. Настройка локали

При изменении регистра русских букв может возникнуть проблема. Чтобы ее избежать, необходимо правильно настроить локаль. *Локалью* называют совокупность локальных настроек системы.

Для установки локали используется функция `setlocale()`. Формат функции:

```
setlocale(<Категория>, <Локаль>);
```

Параметр `<Категория>` может принимать следующие значения:

- `LC_ALL` — устанавливает локаль для всех режимов;
- `LC_COLLATE` — для сравнения строк;
- `LC_CTYPE` — для перевода символов в нижний или верхний регистр;
- `LC_MONETARY` — для отображения денежных единиц;
- `LC_NUMERIC` — для форматирования дробных чисел;
- `LC_TIME` — для форматирования вывода даты и времени.

Пример:

```
$str = "очень длинная строка";
setlocale(LC_CTYPE, "ru_RU.CP1251"); // Настройка локали
echo strtoupper($str); // Выведет: ОЧЕНЬ ДЛИННАЯ СТРОКА
```

Пример для кодировки UTF-8:

```
setlocale(LC_CTYPE, 'ru_RU.UTF-8'); // Настройка локали в UNIX
setlocale(LC_ALL, 'Russian_Russia.65001'); // Настройка локали в Windows
```

Можно указать сразу несколько локалей через запятую:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
setlocale(LC_ALL, 'ru_RU.UTF-8', 'Russian_Russia.65001');
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> В Windows нельзя настроить категорию `LC_CTYPE` для кодировки UTF-8. Это связано с особенностями самой операционной системы.

5.15.3. Функции для работы с символами

- `chr(<Код символа>)` — возвращает символ по указанному коду:

  ```
  echo chr(81); // Выведет: Q
  ```

- `ord(<Символ>)` — возвращает код указанного символа:

  ```
  echo ord("Q"); // Выведет: 81
  ```

5.15.4. Поиск и замена в строке

- `strpos()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов. Формат функции:

  ```
  strpos(<Строка>, <Подстрока>, [<Начальная позиция поиска>]);
  ```

  Если начальная позиция не указана, то поиск будет проводиться с начала строки:

  ```
  if (strpos("Привет", "При") !== false) echo "Найдено";
  // Выведет: Найдено
  else echo "Не найдено";
  ```

- `str_replace()` — заменяет все вхождения подстроки в строке на другую подстроку и возвращает результат в виде новой строки. Функция не изменяет исходную строку и зависит от регистра символов. Формат функции:

  ```
  str_replace(<Подстрока для замены>, <Новая подстрока>, <Строка>,
              [<Число произведенных замен>]);
  ```

  Если в необязательном четвертом параметре указать переменную, то в ней будет сохранено число проведенных замен.

Пример:

```
$str = 'Привет, Петя';
$count = 0;
$str = str_replace('Петя', 'Вася', $str, $count);
echo $str; // Выведет: Привет, Вася
echo $count; // Выведет: 1
```

В качестве параметра можно также передать массив:

```
$arr = array('!', '@', '#', '$', '%', '^', '&', '*',
             '(', ')', '_', '+', '=', '.');
echo str_replace($arr, '', 'Текст !@#$%^&*()_+=. текст');
// Выведет: Текст текст
```

5.15.5. Функции для сравнения строк

- `strcmp(<Строка1>, <Строка2>)` — сравнивает две строки. Зависит от регистра символов. Возвращает одно из трех значений:
  - `0` — если строки равны;
  - `1` — если `<Строка1>` больше `<Строки2>`;
  - `-1` — если `<Строка1>` меньше `<Строки2>`.

  Пример:

```
$str1 = "Строка1";
$str2 = "Строка2";
echo strcmp($str1, $str2); // Выведет: -1
```

- `strcoll(<Строка1>, <Строка2>)` — сравнивает строки на основе локализации. Зависит от регистра символов. Если локаль не настроена, то эта функция эквивалентна функции `strcmp()`:

```
setlocale(LC_ALL, "ru_RU.CP1251"); // Настройка локали
$str1 = "Строка1";
$str2 = "Строка2";
echo strcoll($str1, $str2); // Выведет: -1
```

- `strcasecmp(<Строка1>, <Строка2>)` — сравнивает две строки без учета регистра:

```
$str1 = "строка";
$str2 = "Строка";
echo strcmp($str1, $str2); // Выведет: 1
echo strcasecmp($str1, $str2); // Выведет: 0
```

5.15.6. Кодирование строк

- `urlencode()` — выполняет URL-кодирование строки. Это необходимо, например, для передачи русского текста в строке URL-адреса в качестве параметра сценария:

```
$str = "Текст на русском языке";
echo urlencode($str);
// %D2%E5%EA%F1%F2+%ED%E0+%F0%F3%F1%F1%EA%EE%EC+%FF%E7%FB%EA%E5
```

- `urldecode()` — раскодирует строку, закодированную с помощью функции `urlencode()`:

```
$str = "%D2%E5%EA%F1%F2+%ED%E0+%F0%F3%F1%F1%EA%EE%EC+↵
%FF%E7%FB%EA%E5";
$str = urlencode($str);
echo urldecode($str);
// Выведет: Текст на русском языке
```

Кроме этих функций можно использовать функции `rawurlencode()` и `rawurldecode()`:

```
$str = "Текст с пробелами";
$str = rawurlencode($str);
echo $str;
// Выведет:
// %D2%E5%EA%F1%F2%20%F1%20%EF%F0%EE%E1%E5%EB%E0%EC%E8
echo rawurldecode($str);
// Выведет: Текст с пробелами
```

ОБРАТИТЕ ВНИМАНИЕ

Символ пробела заменяется не знаком `+`, а символами `%20`.

- `md5()` — кодирует строку по алгоритму MD5. Используется для кодирования паролей, т. к. не существует алгоритма для дешифровки. Для сравнения введенного пользователем пароля с сохраненным в базе необходимо зашифровать введенный пароль, а затем выполнить сравнение:

```
$pass = "password";
$pass = md5($pass); // Пароль, сохраненный в базе
echo $pass; // Выведет: 5f4dcc3b5aa765d61d8327deb882cf99
$pass2 = "password"; // Пароль, введенный пользователем
if ($pass === md5($pass2)) echo "Пароль правильный";
```

- `crc32()` — кодирует строку, используя алгоритм DES:

```
$pass = "password";
$pass = crc32($pass);
echo $pass; // Выведет: 901924565
```

5.15.7. Преобразование кодировок

С помощью функции `convert_cyr_string()` можно преобразовать строку из одной кодировки в другую.

Формат функции:

```
convert_cyr_string(<Исходная строка>, <Исходная кодировка>,
      <Нужная кодировка>);
```

Значения параметров <Исходная кодировка> и <Нужная кодировка>:

- a или d — кодировка x-cp866;
- i — кодировка iso8859-5;
- k — кодировка KOI8-R;
- m — кодировка x-mac-cyrillic;
- w — кодировка windows-1251 (cp1251).

Пример вызова функции:

```
$str = "уФТПЛБ";
echo convert_cyr_string($str, "k", "w");
// Выведет: Строка
```

Функция iconv() также преобразовывает символы строки из одной кодировки в другую. Формат функции:

```
iconv(<Исходная кодировка>, <Нужная кодировка>[<Флаг>], <Исходная строка>);
```

Пример преобразования строки из кодировки windows-1251 в UTF-8:

```
$str = iconv("windows-1251", "UTF-8", "Строка");
```

Необязательный параметр <Флаг> может принимать следующие значения:

- //TRANSLIT — если символа нет в нужной кодировке, он заменяется одним или несколькими аналогами;
- //IGNORE — символы, которых нет в нужной кодировке, будут опущены.

Зачем нужен этот параметр? Если мы преобразовываем кодировку windows-1251 в UTF-8, то в этом параметре нет необходимости. А вот если наоборот, то может возникнуть ситуация, что символа нет в нужной кодировке, т. к. кодировка UTF-8 позволяет хранить несколько тысяч символов, а кодировка windows-1251 только 256 символов. Если не указать этот параметр, то строка будет обрезана до первого недонустимого символа. Пример:

```
$str = iconv("UTF-8", "windows-1251//IGNORE", "Строка");
```

Вместо iconv() можно использовать функцию mb_convert_encoding() со следующим форматом вызова:

```
mb_convert_encoding(<Исходная строка>, <Нужная кодировка>,
                    <Исходная кодировка>);
```

Пример преобразования строки из кодировки UTF-8 в windows-1251:

```
$str = mb_convert_encoding("Строка", "windows-1251", "UTF-8");
```

Обратите внимание

Порядок следования параметров в функции mb_convert_encoding() отличается от порядка в функции iconv().

5.15.8. Регулярные выражения. Разбираем адрес электронной почты на составные части. Проверяем правильность введенной даты

Регулярные выражения позволяют осуществить сложный поиск или замену в строке. В языке PHP существуют два формата регулярных выражений: POSIX и PCRE. Синтаксис их схож, но скорость и внутренний механизм работы сильно различаются. В PHP 5.3 функции, которые позволяют использовать регулярные выражения формата POSIX, признаны устаревшими. Все они выводят сообщение:

```
Deprecated: Function <Название функции> is deprecated
```

Тем не менее, функции с префиксом "mb_ereg" такого сообщения не выводят. Эти функции позволяют работать со строками в различных кодировках — не только в однобайтовых (например, windows-1251), но и многобайтовых (таких, как UTF-8). Далее мы рассмотрим именно эти функции.

Прежде чем использовать функции для работы с регулярными выражениями формата POSIX, необходимо настроить кодировку с помощью функции `mb_regex_encoding()`:

```
mb_regex_encoding('windows-1251'); // Установка кодировки 1251
```

или

```
mb_regex_encoding('UTF-8'); // Установка кодировки UTF-8
```

Функции, поддерживающие регулярные выражения формата POSIX:

- `mb_ereg()` — выполняет поиск в строке с помощью регулярного выражения. Зависит от регистра символов. Формат функции:

  ```
  mb_ereg(<Регулярное выражение>, <Строка>, [<Массив>]);
  ```

  В параметре `<Массив>` сохраняются соответствия подвыражений с шаблоном:

  ```
  mb_regex_encoding('UTF-8'); // Установка кодировки
  $str = 'unicross@mail.ru';
  $Mass = array();
  mb_ereg("^([a-z0-9_.-]+)@(([a-z0-9\-]+\.)+[a-z]{2,6})$", $str,
       $Mass);
  foreach($Mass as $var) {
     echo $var . '<br>';
  }
  ```

  Этот пример выведет HTML-код, который в Web-браузере отображается так:

  ```
  unicross@mail.ru
  unicross
  mail.ru
  mail.
  ```

Первый элемент массива соответствует найденной строке, второй — строке в первых круглых скобках, третий — во вторых круглых скобках и т. д.;

- `mb_eregi()` — выполняет поиск в строке с помощью регулярного выражения без учета регистра символов. Имеет такой же формат, как и функция `mb_ereg()`.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> В некоторых случаях функция `mb_eregi()` некорректно работает с буквами русского языка. Поэтому русские буквы лучше указывать и в верхнем и в нижнем регистрах (например, `[а-яА-яеЕ]`).

С помощью функций `mb_ereg()` и `mb_eregi()` обычно проверяются входные данные. Например, правильность ввода E-mail можно проконтролировать следующим образом:

```
mb_regex_encoding('UTF-8'); // Установка кодировки
$str = 'unicross@mail.ru';
$pattern = "^[a-z0-9_.-]+@([a-z0-9-]+\.)+[a-z]{2,6}$";
if (mb_eregi($pattern, $str)) echo "Нормально";
else echo "Нет";
// Выведет: Нормально
```

- `mb_ereg_replace()` — возвращает строку, которая является результатом поиска и замены в исходной строке с использованием регулярного выражения. Функция по умолчанию зависит от регистра символов и имеет следующий формат:

```
mb_ereg_replace(<Регулярное выражение>, <Новый фрагмент>,
                <Исходная строка>, [<Модификатор>]);
```

Необязательный параметр `<Модификатор>` может содержать комбинацию следующих флагов:

- `i` — поиск без учета регистра. С русскими буквами возможны проблемы;
- `m` — поиск в строке, состоящей из нескольких строк, разделенных символом новой строки. Метасимвол "точка" соответствует любому символу, кроме символа перевода строки (`\n`);
- `s` — однострочный режим. Метасимвол "точка" соответствует любому символу, в том числе и символу перевода строки;
- `x` — разрешает использовать в регулярном выражении пробельные символы и однострочные комментарии;
- `e` — указывает, что в строке для замены указано выражение языка PHP, которое необходимо предварительно вычислить.

Пример:

```
mb_regex_encoding('UTF-8'); // Установка кодировки
$str = '2001, 2002, 2003, 2004, 2005';
$pattern = '200[14]';
echo mb_ereg_replace($pattern, '2007', $str, 's');
// Выведет: 2007, 2002, 2003, 2007, 2005
```

- `mb_eregi_replace()` — выполняет поиск и замену без учета регистра символов. Формат такой же, как и у функции `mb_ereg_replace()`;
- `mb_split()` — разделяет строку на подстроки по указанному разделителю, в качестве которого используется регулярное выражение, и добавляет их в массив. Например, код

```
mb_regex_encoding('UTF-8'); // Установка кодировки
$str = 'unicross@mail.ru';
$Mass = array();
$Mass = mb_split('[@.]', $str);
foreach ($Mass as $key) {
   echo $key . '<br>';
}
```

выведет HTML-код, который отображается так:

```
unicross
mail
ru
```

Метасимволы, используемые в регулярных выражениях

Два метасимвола позволяют осуществить привязку:

- `^` — привязка к началу строки;
- `$` — привязка к концу строки.

Например, привязка необходима для проверки, содержит ли строка число:

```
mb_regex_encoding('UTF-8'); // Установка кодировки
$str = '2';
if (mb_ereg('^[0-9]+$', $str)) echo 'Число'; // Выведет: Число
else echo 'Не число';
$str = 'Строка2';
if (mb_ereg('^[0-9]+$', $str)) echo 'Число';
else echo 'Не число'; // Выведет: Не число
```

Если убрать привязку к началу и концу строки, то любая строка, содержащая хотя бы одну цифру, будет распознана как "Число":

```
mb_regex_encoding('UTF-8'); // Установка кодировки
$str = 'Строка2';
if (mb_ereg('[0-9]+', $str)) echo 'Число'; // Выведет: Число
else echo 'Не число';
```

Можно указать привязку только к началу или только к концу строки:

```
mb_regex_encoding('UTF-8'); // Установка кодировки
$str = 'Строка2';
if (mb_ereg('[0-9]+$', $str)) echo 'Есть число в конце строки';
else echo 'Нет числа в конце строки';
// Выведет: Есть число в конце строки
```

```
if (mb_ereg('^[0-9]+', $str)) echo 'Есть число в начале строки';
else echo 'Нет числа в начале строки';
// Выведет: Нет числа в начале строки
```

В квадратных скобках `[]` можно указать символы, которые могут встречаться на этом месте в строке. Символы можно перечислять подряд или указать диапазон через дефис:

- `[09]` — соответствует числу 0 или 9;
- `[0-9]` — соответствует любому числу от 0 до 9;
- `[абв]` — соответствует буквам "а", "б" и "в";
- `[а-г]` — соответствует буквам "а", "б", "в" и "г";
- `[а-яе]` — соответствует любой букве от "а" до "я";
- `[АБС]` — соответствует буквам "А", "Б" и "С";
- `[А-ЯЕ]` — соответствует любой русской букве от "А" до "Я";
- `[а-яА-ЯеЕ]` — соответствует любой русской букве в любом регистре;
- `[0-9а-яА-ЯеЕa-zA-Z]` — любая цифра и любая буква независимо от регистра и языка.

> ***Обратите внимание***
>
> Буква "ё" не входит в диапазон `[а-я]`. Кроме того, для русских букв лучше учитывать регистр символов.

Значение можно инвертировать, если после первой скобки указать символ `^`. Таким образом можно указать символы, которых не должно быть на этом месте в строке:

- `[^09]` — не цифра 0 или 9;
- `[^0-9]` — не цифра от 0 до 9;
- `[^а-яА-ЯеЕa-zA-Z]` — не буква.

Вместо перечисления символов можно использовать стандартные классы:

- `[[:alnum:]]` — алфавитно-цифровые символы;
- `[[:alpha:]]` — буквенные символы;
- `[[:lower:]]` — строчные буквы;
- `[[:upper:]]` — прописные буквы;
- `[[:digit:]]` — десятичные цифры;
- `[[:xdigit:]]` — шестнадцатеричные цифры;
- `[[:punct:]]` — знаки нунктуации;
- `[[:blank:]]` — символы табуляции и пробелов;
- `[[:space:]]` — пробельные символы;
- `[[:cntrl:]]` — управляющие символы;
- `[[:print:]]` — печатные символы;

- `[[:graph:]]` — печатные символы, за исключением пробельных;
- . (точка) — любой символ, кроме символа новой строки (`\n`).

> ***Внимание!***
> Стандартные классы работают только с буквами латинского алфавита, а с буквами русского языка не работают.

Что же делать, если нужно найти точку, ведь символ "точка" соответствует любому символу, кроме символа перевода строки? Для этого перед специальным символом необходимо указать символ "\" или разместить точку внутри квадратных скобок (`[.]`). Продемонстрируем это на примере (листинг 5.13).

Листинг 5.13. Проверка правильности введенной даты

```
mb_regex_encoding('UTF-8'); // Установка кодировки
$str = '29,04.2007'; // Неправильная дата (вместо точки указана запятая)

$pattern = '^[0-3][0-9].[01][0-9].[12][09][0-9][0-9]$';
// Символ "\" не указан перед точкой
if (mb_ereg($pattern , $str)) echo 'Дата введена правильно';
else echo 'Дата введена неправильно';
// Так как точка означает любой символ, выведет: Дата введена правильно

$pattern = '^[0-3][0-9]\.[01][0-9]\.[12][09][0-9][0-9]$';
// Символ "\" указан перед точкой
if (mb_ereg($pattern , $str)) echo 'Дата введена правильно';
else echo 'Дата введена неправильно';
// Так как перед точкой указан символ "\",
// выведет: Дата введена неправильно

$pattern = '^[0-3][0-9][.][01][0-9][.][12][09][0-9][0-9]$';
// Точка внутри квадратных скобок
if (mb_ereg($pattern , $str)) echo 'Дата введена правильно';
else echo 'Дата введена неправильно';
// Выведет: Дата введена неправильно
```

Точка теряет свое специальное значение, если ее заключить в квадратные скобки. Символ "^" теряет свое специальное значение, если он не расположен сразу после открывающей квадратной скобки. Чтобы отменить специальное значение символа "-", его необходимо указать после перечисления всех символов, перед закрывающей квадратной скобкой. Все специальные символы можно сделать обычными, если перед ними указать символ "\".

Число вхождений символа в строку задается с помощью *квантификаторов*:

- `{n}` — в точности `n` вхождений символа в строку:

  `[[:digit:]]{2}` — соответствует двум вхождениям любой цифры;

- `{n,}` — `n` или более вхождений символа в строку:

  `[[:digit:]]{2,}` — соответствует двум и более вхождениям любой цифры;

- `{n,m}` — не менее `n` и не более `m` вхождений символа в строку. Цифры указываются через запятую без пробела:

  `[[:digit:]]{2,5}` — соответствует числу вхождений от двух до няти любой цифры;

- `*` — произвольное число вхождений символа в строку (в том числе ни одного вхождения):

  `[[:digit:]]*` — цифры могут не встретиться в строке или встретиться много раз;

- `+` — одно или большее число вхождений символа в строку:

  `[[:digit:]]+` — цифра может встретиться один или много раз;

- `?` — ноль или одно число вхождений символа в строку:

  `[[:digit:]]?` — цифра может встретиться один раз или не встретиться совсем.

Логическое ИЛИ

`n|m` — соответствует одному из символов или выражений `n` или `m`:

`красн(ая)|(ое)` — `красная` или `красное`, но не `красный`.

5.15.9. Perl-совместимые регулярные выражения

В предыдущем разделе мы рассмотрели регулярные выражения POSIX. Кроме формата POSIX, в языке PHP существует поддержка Perl-совместимых регулярных выражений (PCRE, Perl-compatible Regular Expression). Именно формат PCRE следует использовать при работе с однобайтовыми кодировками, а также с кодировкой UTF-8.

Шаблон PCRE представляет собой строку, заключенную в кавычки или апострофы, внутри которой между двумя ограничителями указывается регулярное выражение. За последним ограничителем может быть указан модификатор. В качестве ограничителя могут выступать одинаковые символы или парные скобки:

```
'/<Регулярное выражение>/[<Модификатор>]'
'#<Регулярное выражение>#[<Модификатор>]'
'"<Регулярное выражение>"[<Модификатор>]'
'{<Регулярное выражение>}[<Модификатор>]'
'(<Регулярное выражение>)[<Модификатор>]'
```

В параметре `<Модификатор>` могут быть указаны следующие флаги (или их комбинация):

- `i` — поиск без учета регистра;
- `m` — поиск в строке, состоящей из нескольких строк, разделенных символом новой строки. Символ `^` соответствует привязке к началу каждой подстроки,

а символ $ соответствует позиции перед символом перевода строки. Метасимвол "точка" соответствует любому символу, кроме символа перевода строки (\n);

- s — однострочный режим. Символ ^ соответствует привязке к началу строки, а символ $ соответствует концу строки. Метасимвол "точка" соответствует любому символу, в том числе и символу перевода строки;
- x — разрешает использовать в регулярном выражении пробельные символы и однострочные комментарии, начинающиеся с символа #;
- e — указывает, что в строке для замены в функции preg_replace() указано выражение языка PHP, которое необходимо предварительно вычислить. Вместо этого флага лучше вызывать функцию preg_replace_callback();
- u — служит для обработки строк в кодировке UTF-8.

Метасимволы, применяемые в регулярных выражениях PCRE:

- ^ — привязка к началу строки (назначение зависит от модификатора);
- $ — привязка к концу строки (назначение зависит от модификатора);
- \A — привязка к началу строки (не зависит от модификатора);
- \z — привязка к концу строки (не зависит от модификатора);
- [] — позволяет указать символы, которые могут встречаться на этом месте в строке. Можно перечислять символы подряд или указать диапазон через тире;
- [^] — позволяет указать символы, которые не могут встречаться на этом месте в строке. Можно перечислять символы подряд или указать диапазон через тире;
- n|m — соответствует одному из символов n или m;
- . (точка) — любой символ, кроме символа перевода строки (\n). Если указан модификатор s, то метасимвол "точка" соответствует всем символам, включая символ перевода строки. Внутри квадратных скобок точка не имеет специального значения.

Кроме того, в регулярных выражениях PCRE допустимы следующие стандартные классы:

- \d — соответствует любой цифре;
- \w — соответствует любой букве или цифре;
- \s — любой пробельный символ (пробел, табуляция, перевод страницы, новая строка или перевод каретки);
- \D — не цифра;
- \W — не буква и не цифра;
- \S — не пробельный символ.

Также можно использовать стандартные классы регулярных выражений формата POSIX.

Квантификаторы, указанные в регулярных выражениях PCRE, позволяют задать число повторов предшествующего символа или выражения:

- `{n}` — `n` вхождений символа в строку;
- `{n,}` — `n` или более вхождений символа в строку;
- `{n,m}` — не менее `n` и не более `m` вхождений символа в строку. Числа записывают через запятую без пробела;
- `*` — ноль или большее число вхождений символа в строку;
- `+` — одно или большее число вхождений символа в строку;
- `?` — ни одного или одно вхождение символа в строку.

Регулярные выражения PCRE можно использовать в нескольких функциях.

- `preg_grep(<Шаблон>, <Массив>, [PREG_GREP_INVERT])` — возвращает новый массив, состоящий из элементов `<Массива>`, которые соответствуют `<Шаблону>`. Индексы массива сохраняются. Если указан флаг `PREG_GREP_INVERT`, то возвращается массив значений, не соответствующих шаблону. В качестве примера получим все элементы массива, состоящие только из цифр, и наоборот:

```
$arr = array(20, 54, "Текст", 457);
$pattern = '/^[0-9]+$/s';
$arr2 = preg_grep($pattern, $arr);
echo implode(" - ", $arr2);
// Выведет: 20 - 54 - 457
echo "<br>";
$arr3 = preg_grep($pattern, $arr, PREG_GREP_INVERT);
echo implode(" - ", $arr3);
// Выведет: Текст
```

- `preg_match()` — ищет первое совпадение с шаблоном в заданной строке. Формат функции:

```
preg_match(<Шаблон>, <Строка>, [<Массив совпадений>], [<Флаг>],
      [<Смещение от начала строки>]);
```

Функция возвращает `0`, если совпадение не найдено, и `1` при соответствии шаблону. Если указан параметр `<Массив совпадений>`, то первый элемент массива будет содержать фрагмент, полностью соответствующий шаблону, а остальные элементы массива — это фрагменты, заключенные в шаблоне в круглые скобки.

В качестве примера проверим E-mail на соответствие шаблопу:

```
$emails = 'unicross@mail.ru';
$pattern = '/^([a-z0-9_.-]+)@(([a-z0-9-]+\.)+[a-z]{2,6})$/is';
$arr = array();
if (preg_match($pattern, $emails, $arr)) {
   echo 'E-mail ' . $arr[0] . ' соответствует шаблону';
   echo '<br>ящик: ', $arr[1], ' домен: ', $arr[2];
}
```

```
else {
   echo 'E-mail не соответствует шаблону';
}
```

Этот пример выведет HTML-код, который отображается так:

```
E-mail unicross@mail.ru соответствует шаблону
ящик: unicross домен: mail.ru
```

Элемент массива `$arr[0]` — это полный текст, соответствующий шаблону. Элемент `$arr[1]` соответствует `([a-z0-9_.-]+)`, а элемент `$arr[2]` соответствует шаблону во вторых круглых скобках `(([a-z0-9-]+\.)+[a-z]{2,6})`.

Если какой-либо фрагмент заносить в массив не надо, то после открывающей круглой скобки следует указать комбинацию символов `?:`.

Например, результат выполнения кода

```
$emails = 'unicross@mail.ru';
$pattern = '/^(?:[a-z0-9_.-]+)@(([a-z0-9-]+\.)+[a-z]{2,6})$/is';
$arr = array();
if (preg_match($pattern, $emails, $arr)) {
   echo 'E-mail ' . $arr[0] . ' соответствует шаблону';
   echo '<br>домен: ', $arr[1];
}
else {
   echo 'E-mail не соответствует шаблону';
}
```

отобразится в Web-браузере так:

```
E-mail unicross@mail.ru соответствует шаблону
домен: mail.ru
```

В этом примере элемент массива `$arr[1]` соответствует фрагменту уже не в первых круглых скобках, а во вторых.

Если в параметре `<Флаг>` задано значение `PREG_OFFSET_CAPTURE`, то для каждого найденного фрагмента будет указана его позиция в исходной строке. Для примера получим текст между одинаковыми парными тегами и выведем смещение относительно начала строки:

```
$str = "<b>Текст</b>";
$pattern = '#<([^>])>(.*)</\1>#s';
$arr = array();
if (preg_match($pattern, $str, $arr, PREG_OFFSET_CAPTURE)) {
   echo 'Фрагмент: ', $arr[2][0], '<br>';
   echo 'Смещение: ', $arr[2][1];
}
```

Этот пример иллюстрирует механизм *обратных ссылок*. К найденному фрагменту в круглых скобках внутри шаблона можно обратиться, указав его порядковый номер после слэша, например, `\1` соответствует `([^>])`.

Обратите внимание

При использовании флага `PREG_OFFSET_CAPTURE` **изменился формат массива** `$arr`:

```
$arr[0][0] => "<b>Текст</b>"
$arr[0][1] => 0
$arr[1][0] => "b"
$arr[1][1] => 1
$arr[2][0] => "Текст"
$arr[2][1] => 3
```

- `preg_match_all()` — ищет все совпадения с шаблоном в заданной строке. Формат функции:

```
preg_match_all(<Шаблон>, <Строка>, [<Массив совпадений>],
      [<Флаг>], [<Смещение от начала строки>]);
```

Функция возвращает число найденных совпадений с шаблоном (или `0`, если совпадения не найдены).

Если в параметре `<Флаг>` указано значение `PREG_PATTERN_ORDER`, то массив совпадений будет содержать элементы, упорядоченные по порядковому номеру фрагмента, заключенного в шаблоне в круглые скобки. Нулевой элемент массива будет содержать список полных совпадений с шаблоном. В качестве примера получим все значения между тегами `<b>` и `</b>`:

```
$str = "<b>Значение1</b>Лишнее значение<b>Значение2</b>";
$pattern = '#<b>(.*)</b>#is';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_PATTERN_ORDER);
$count = count($arr[1]);
for ($i=0; $i<$count; $i++) {
   echo $arr[1][$i] . "<br>";
}
```

Вместо желаемого результата мы получим

```
Значение1</b>Лишнее значение<b>Значение2<br>
```

Такое поведение квантификаторов называется "жадностью". При поиске соответствия ищется самая длинная подстрока, соответствующая шаблону, и не учитываются более короткие соответствия. В нашем случае самой длинной подстрокой является вся строка. Чтобы ограничить эту "жадность", необходимо после символа `*` указать символ `?`.

Если в параметре `<Флаг>` указано значение `PREG_SET_ORDER`, то массив совпадений будет содержать элементы, упорядоченные по номеру совпадения. Каждый элемент массива будет содержать список совпадений фрагментов, заключенных в шаблоне в круглые скобки. Нулевой элемент массива будет содержать полное совпадение с шаблоном. В качестве примера получим все значения между тегами `<b>` и `</b>` с учетом "жадности" квантификаторов:

```
$str = "<b>Значение1</b>Лишнее значение<b>Значение2</b>";
$pattern = '#<b>(.*?)</b>#is';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
$count = count($arr);
for ($i=0; $i<$count; $i++) {
   echo $arr[$i][1] . "<br>";
}
```

Этот код выведет то, что мы искали:

```
Значение1<br>Значение2<br>
```

Ограничить "жадность" всех квантификаторов в шаблоне позволяет модификатор `U`. Обратите внимание на регистр модификатора. Буква должна быть прописной:

```
$pattern = '#<b>(.*)</b>#isU';
```

Если к флагам `PREG_PATTERN_ORDER` и `PREG_SET_ORDER` добавить значение `PREG_OFFSET_CAPTURE`, то для каждого найденного фрагмента будет указана его позиция в исходной строке:

```
$str = "<b>Значение1</b>Лишнее значение<b>Значение2</b>";
$pattern = '#<b>(.*?)</b>#is';
$arr = array();
preg_match_all($pattern, $str, $arr,
               PREG_SET_ORDER | PREG_OFFSET_CAPTURE);
echo "<pre>";
print_r($arr);
echo "</pre>";
```

Код HTML, выведенный в этом примере, будет отображен в Web-браузере так:

```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => Значение1
                    [1] => 0
                )
            [1] => Array
                (
                    [0] => Значение1
                    [1] => 3
                )
        )
    [1] => Array
        (
            [0] => Array
```

```
                (
                    [0] => Значение2
                    [1] => 31
                )
            [1] => Array
                (
                    [0] => Значение2
                    [1] => 34
                )
        )
)
```

Здесь основной вывод был осуществлен функцией `print_r()`, предназначенной для вывода массивов, в том числе вложенных.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Функция `preg_match_all()` показывает смещение в байтах, а не в символах. При использовании кодировки UTF-8 это будет иметь значение.

- `preg_replace()` — ищет все совпадения с шаблоном и заменяет их указанным значением. Формат функции:

```
preg_replace(<Шаблон>, <Новый фрагмент>, <Исходная строка>,
      [<Лимит>]);
```

Функция возвращает измененную строку. Если совпадения не найдены, то функция вернет исходную строку. Первые три параметра могут быть одномерными массивами. Если указан параметр `<Лимит>`, то функция заменит только указанное число первых совпадений с шаблоном.

В качестве примера возьмем два тега и поменяем имена тегов местами:

```
$str = "<br><td>";
$pattern = '#<(\w+)><(\w+)>#is';
$repl = '<$2><$1>';
$str2 = preg_replace($pattern, $repl, $str);
echo htmlspecialchars($str2);
// Выведет в окне Web-браузера: <td><br>
```

Обратиться к найденному фрагменту в круглых скобках можно не только с помощью синтаксиса `$n`, но и указав номер скобок, перед которым стоят два слэша (`\\n`):

```
$str = "<br><td>";
$pattern = '#<(\w+)><(\w+)>#is';
$repl = '<\\2><\\1>';
$str2 = preg_replace($pattern, $repl, $str);
echo htmlspecialchars($str2);
// Выведет в окне Web-браузера: <td><br>
```

Чтобы отделить номер скобки от последующего текста, необходимо заключить номер в фигурные скобки (`\${2}`).

Если в шаблоне указан флаг e, то внутри выражения для замены можно использовать конструкции языка PHP. В качестве примера поменяем теги местами и выведем названия тегов строчными буквами:

```
$str = "<BR><TD>";
$pattern = '#<(\w+)><(\w+)>#ise';
$repl = "'<' . strtolower('$2') . '><' . strtolower('$1') . '>'";
$str2 = preg_replace($pattern, $repl, $str);
echo htmlspecialchars($str2);
// Выведет в окне Web-браузера: <td><br>
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Вместо этого способа замены лучше воспользоваться функцией `preg_replace_callback()`. Все дело в том, что переменные `$1`, `$2` и т. д. перед вставкой в строку автоматически обрабатываются функцией `addslashes()`. Если в этих переменных содержатся кавычки и апострофы, то в итоговой строке обязательно будут лишние слэши. Поэтому лучше отказаться от использования флага e.

- `preg_replace_callback()` — выполняет поиск по шаблону и замену с использованием функции обратного вызова. Формат функции:

```
preg_replace_callback(<Шаблон>, <Имя функции>,
      <Строка для замены>, [<Лимит>]);
```

В отличие от `preg_replace()` функция `preg_replace_callback()` передает функции, указанной в параметре `<Имя функции>`, найденные совпадения. Результат, возвращаемый этой функцией, служит фрагментом для замены.

Переделаем наш предыдущий пример и добавим функцию обратного вызова:

```
$str = "<BR><TD>";
$pattern = '#<(\w+)><(\w+)>#is';
$str2 = preg_replace_callback($pattern, "f_replace", $str);
echo htmlspecialchars($str2);
// Выведет в окне Web-браузера: <td><br>
function f_replace($arr) {
   $repl = '<' . strtolower($arr[2]);
   $repl .= '><' . strtolower($arr[1]) . '>';
   return $repl;
}
```

Нулевой элемент массива `$arr` будет содержать полное соответствие шаблону, а последующие элементы соответствуют фрагментам, заключенным в шаблоне в круглые скобки.

- `preg_split()` — разбивает строку по шаблону и возвращает массив подстрок. Формат функции:

```
preg_split(<Шаблон>, <Исходная строка>, [<Лимит>], [<Флаг>]);
```

В параметре `<Флаг>` могут быть указаны следующие значения (или комбинация значений, соединенных оператором `|`):

- PREG_SPLIT_NO_EMPTY — функция вернет только непустые подстроки;
- PREG_SPLIT_DELIM_CAPTURE — фрагмент, заключенный в шаблоне в круглые скобки, также будет возвращаться;
- PREG_SPLIT_OFFSET_CAPTURE — для каждой найденной подстроки будет указана ее позиция в исходной строке.

Например, разбить E-mail на составные части можно так:

```
$str = 'unicross@mail.ru';
$arr = preg_split('/[@.]/', $str);
$count = count($arr);
for ($i=0; $i<$count; $i++) {
   echo $arr[$i] . "<br>";
} // Выведет unicross<br>mail<br>ru<br>
```

Если не требуется указания шаблона, то вместо функции preg_split() лучше использовать функцию explode().

5.15.10. Функции для работы со строками в кодировке UTF-8

В однобайтовых кодировках символ кодируется одним байтом. Первые 7 бит позволяют закодировать 128 символов, соответствующих кодировке ASCII. Символы, имеющие код меньше 33, являются специальными, например, нулевой символ, символ переноса строки, табуляция и т. д. Получить остальные символы позволяет следующий код:

```
for ($i=33; $i<128; $i++) {
   echo $i . " =&gt; " . chr($i) . "<br>";
}
```

Коды этих символов одинаковы практически во всех однобайтовых кодировках. Восьмой бит предназначен для кодирования символов национальных алфавитов. Таким образом, однобайтовые кодировки позволяют закодировать всего 256 символов.

К любому символу строки в однобайтовой кодировке (например, windows-1251 или KOI8-R) можно обратиться как к элементу массива. Достаточно указать его индекс в квадратных скобках. Нумерация начинается с нуля:

```
$X = 'Привет'; // Кодировка windows-1251 или KOI8-R
echo $X[0];
```

В кодировке UTF-8 один символ может кодироваться несколькими байтами. Первые 128 символов соответствуют кодировке ASCII и кодируются всего одним байтом. Остальные символы кодируются переменным числом байтов от двух до шести (на практике только до четырех). Буквы русского алфавита и некоторых других европейских языков кодируются двумя байтами. По этой причине использовать обычные строковые функции нельзя. В данном разделе мы рассмотрим функции, которые подойдут при работе с кодировкой UTF-8.

Так как в кодировке UTF-8 один символ может кодироваться несколькими байтами, то обратиться к символу как к элементу массива можно только после перекодировки. Тем не менее, к символам кодировки ASCII мы можем обратиться как к элементам массива, т. к. они кодируются одним байтом:

```
$X = 'String'; // Кодировка UTF-8
echo $X[0]; // Выведет: S
```

Если необходимо обращаться к любым символам как к элементам массива, то можно воспользоваться следующим кодом:

```
<?php
header('Content-Type: text/html; charset=utf-8');
$str = 'Строка';
$count = mb_strlen($str, 'UTF-8');
$arr = array();
for ($i=0; $i<$count; $i++) {
   $arr[] = mb_substr($str, $i, 1, 'UTF-8');
}
echo '<pre>';
print_r($arr);
echo '</pre>';
?>
```

> ***Обратите внимание***
>
> Для работы PHP с кодировкой UTF-8 необходимо, чтобы в конфигурационном файле была подключена библиотека php_mbstring.dll.

Не забываем, что настроив сервер на кодировку windows-1251, при работе с UTF-8 необходимо указывать кодировку явным образом. Шаблон программы будет выглядеть так:

```
<?php
header('Content-Type: text/html; charset=utf-8');
// Сюда вставляем примеры из этого раздела
?>
```

Кроме того, сам файл следует сохранить в кодировке UTF-8. Как это сделать, было рассказано в начале этой главы.

Для работы со строками в кодировке UTF-8 (а также с другими кодировками) предназначены следующие функции:

- `mb_strlen(<Строка>[, <Кодировка>])` — возвращает число символов в строке:

```
$str = 'Строка';
echo mb_strlen($str, 'UTF-8'); // Выведет: 6
```

- `iconv_strlen(<Строка>[, <Кодировка>])` — возвращает число символов в строке:

```
$str = 'Строка';
echo iconv_strlen($str, 'UTF-8'); // Выведет: 6
```

- `strlen(<Строка>)` — возвращает число байтов в строке. Так как в однобайтовых кодировках один символ описывается одним байтом, функция `strlen()` возвращает число символов. Для многобайтовых кодировок функция возвращает именно число байтов:

```
$str = 'Строка UTF-8';
echo strlen($str); // Выведет: 18
$str = iconv('UTF-8', 'windows-1251', $str);
echo strlen($str); // Выведет: 12
```

Почему же мы получили 18 байтов, а не 24? Все дело в том, что в кодировке UTF-8 первые 128 символов кодируются одним байтом, а все последующие символы кодируются несколькими байтами. Каждый символ в слове "Строка" занимает по два байта, а в последующей части строки ("UTF-8") каждый символ занимает один байт. Итого 6 умножить на 2 плюс 6 равно 18 байтов.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Если в конфигурационном файле php.ini директива `mbstring.func_overload` равна 2 или 7, то функция `strlen()` полностью эквивалентна функции `mb_strlen()`. Это означает, что функция `strlen()` будет возвращать число символов, а не байтов.

- `mb_substr()` — возвращает подстроку указанной длины, начиная с заданной позиции. Если длина не указана, то возвращается подстрока, начиная с заданной позиции и до конца строки. Формат функции:

```
mb_substr(<Строка>, <Начальная позиция>[, <Длина>[, <Кодировка>]]);
```

Пример 1:

```
$str = 'Строка';
$str1 = mb_substr($str, 0, 1, 'UTF-8');
echo $str1; // Выведет: С
```

Пример 2:

```
mb_internal_encoding('UTF-8'); // Установка кодировки
$str = 'Строка';
$str2 = mb_substr($str, 1);
echo $str2; // Выведет: трока
```

Для настройки кодировки необходимо указать ее в четвертом параметре функции `mb_substr()` или отдельно в функции `mb_internal_encoding()`;

- `iconv_substr()` — возвращает подстроку указанной длины, начиная с заданной позиции. Если длина не указана, то возвращается подстрока, начиная с заданной позиции и до конца строки. Формат функции:

```
iconv_substr(<Строка>, <Начальная позиция>[, <Длина>[, <Кодировка>]]);
```

Пример 1:

```
$str = 'Строка';
$str1 = iconv_substr($str, 0, 1, 'UTF-8');
echo $str1; // Выведет: С
```

Пример 2:

```
iconv_set_encoding('internal_encoding', 'UTF-8');
$str = 'Строка';
$str2 = iconv_substr($str, 1);
echo $str2; // Выведет: трока
```

Для настройки кодировки необходимо указать ее в четвертом параметре функции `iconv_substr()` или отдельно в функции `iconv_set_encoding()`;

- `mb_encode_mimeheader()` — позволяет закодировать текст с помощью методов base64 или Quoted-Printable. Формат функции:

```
mb_encode_mimeheader(<Строка>, [<Кодировка>[,
   <Метод кодирования>[, <Символ переноса строк>]]]);
```

Если параметр `<Кодировка>` не указан, то берется значение, указанное в функции `mb_internal_encoding()`. Как показывает практика, указывать кодировку в функции `mb_internal_encoding()` нужно обязательно. Параметр `<Метод кодирования>` может принимать значения `"B"` (base64) или `"Q"` (Quoted-Printable). Если параметр не указан, то используется значение `"B"`. Параметр `<Символ переноса строк>` задает символ для разделения строк. По умолчанию предполагается комбинация `"\r\n"`. Пример:

```
mb_internal_encoding('UTF-8');
$tema = 'Сообщение';
echo mb_encode_mimeheader($tema);
// Выведет: =?UTF-8?B?0KHQvtC+0LHRidC10L3QuNC1?=
```

Для изменения регистра символов предназначены следующие функции:

- `mb_strtoupper(<Строка>[, <Кодировка>])` — приводит все символы строки к верхнему регистру:

```
$str = 'очень длинная строка';
echo mb_strtoupper($str, 'UTF-8');
// Выведет: ОЧЕНЬ ДЛИННАЯ СТРОКА
```

- `mb_strtolower(<Строка>[, <Кодировка>])` — приводит все символы строки к нижнему регистру:

```
$str = 'ОЧЕНЬ длинная строка';
echo mb_strtolower($str, 'UTF-8');
// Выведет: очень длинная строка
```

- `mb_convert_case(<Строка>, <Режим>[, <Кодировка>])` — преобразует регистр символов в зависимости от значения второго параметра. Параметр `<Режим>` может принимать следующие значения:
  - `MB_CASE_UPPER` — приводит все символы строки к верхнему регистру;
  - `MB_CASE_LOWER` — приводит все символы строки к нижнему регистру;
  - `MB_CASE_TITLE` — приводит первые символы всех слов к верхнему регистру.

Примеры:

```
$str = 'ОЧЕНЬ длинная строка';
echo mb_convert_case($str, MB_CASE_UPPER, 'UTF-8');
// Выведет: ОЧЕНЬ ДЛИННАЯ СТРОКА
echo '<br>';
echo mb_convert_case($str, MB_CASE_LOWER, 'UTF-8');
// Выведет: очень длинная строка
echo '<br>';
echo mb_convert_case($str, MB_CASE_TITLE, 'UTF-8');
// Выведет: Очень Длинная Строка
echo '<br>';
mb_internal_encoding('UTF-8'); // Установка кодировки
echo mb_convert_case($str, MB_CASE_UPPER);
// Выведет: ОЧЕНЬ ДЛИННАЯ СТРОКА
```

Для поиска в строке предусмотрены следующие функции:

- `mb_strpos()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов и имеет следующий формат:

```
mb_strpos(<Строка>, <Подстрока>[, <Начальная позиция поиска>[,
          <Кодировка>]]);
```

Если начальная позиция не указана, то поиск будет осуществляться с начала строки:

```
echo mb_strpos('Привет', 'ри', 0, 'UTF-8'); // Выведет: 1
mb_internal_encoding('UTF-8'); // Установка кодировки
if (mb_strpos('Привет', 'При') !== false) echo 'Найдено';
// Выведет: Найдено
else echo 'Не найдено';
```

- `iconv_strpos()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов. Если начальная позиция не указана, то поиск будет производиться с начала строки. Формат функции:

```
iconv_strpos(<Строка>, <Подстрока>[,
          <Начальная позиция поиска>[, <Кодировка>]]);
```

Примеры:

```
echo iconv_strpos('Привет', 'ри', 0, 'UTF-8'); // Выведет: 1
if (iconv_strpos('Привет', 'При', 0, 'UTF-8') !== false)
echo 'Найдено';
// Выведет: Найдено
else echo 'Не найдено';
```

- `mb_stripos()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. В отличие от функции `mb_strpos()` не зависит от регистра символов. Формат функции:

```
mb_stripos(<Строка>, <Подстрока>[, <Начальная позиция поиска>[,
          <Кодировка>]]);
```

Пример:

```
echo mb_stripos('Привет', 'РИ', 0, 'UTF-8'); // Выведет: 1
```

Если начальная позиция не указана, то поиск будет производиться с начала строки;

- `mb_strrpos()` — ищет подстроку в строке, начиная с ее конца. Возвращает позицию последнего вхождения подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов. Формат функции:

```
mb_strrpos(<Строка>, <Подстрока>[, <Начальная позиция поиска>[,
          <Кодировка>]]);
```

Если начальная позиция не указана, то поиск начнется с конца строки:

```
echo mb_strrpos('ерпарверпр', 'ер', 0, 'UTF-8'); // Выведет: 6
```

- `iconv_strrpos()` — ищет подстроку в строке, начиная с ее конца. Возвращает позицию последнего вхождения подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов. Формат функции:

```
iconv_strrpos(<Строка>, <Подстрока>[, <Кодировка>]);
```

Пример:

```
echo iconv_strrpos('ерпарверпр', 'ер', 'UTF-8'); // Выведет: 6
```

- `mb_strripos()` — ищет подстроку в строке, начиная с ее конца. Возвращает позицию последнего вхождения подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. В отличие от функции `mb_strrpos()` не зависит от регистра символов. Формат функции:

```
mb_strripos(<Строка>, <Подстрока>[, <Начальная позиция поиска>[,
         <Кодировка>]]);
```

Если начальная позиция не указана, то поиск начнется с конца строки:

```
echo mb_strripos('ерпарверпр', 'ЕР', 0, 'UTF-8'); // Выведет: 6
```

- `mb_substr_count()` — возвращает число вхождений подстроки в строку. Функция зависит от регистра символов. Формат функции:

```
mb_substr_count(<Строка>, <Подстрока>[, <Кодировка>]);
```

Пример:

```
echo mb_substr_count('ерпаерпр', 'ер', 'UTF-8'); // Выведет: 2
```

Как вы уже наверняка заметили, параметр <Кодировка> во всех этих функциях необязательный.

Если параметр не указан, то:

- при вызове функций, начинающихся с префикса "mb_", используется значение директивы mbstring.internal_encoding или значение, указанное в функции mb_internal_encoding();
- при вызове функций, начинающихся с префикса "iconv_", используется значение директивы iconv.internal_encoding или значение, указанное в функции iconv_set_encoding().

Функции iconv() и mb_convert_encoding() *(см. разд. 5.15.7)* позволяют выполнить преобразование кодировок.

Некоторые обычные строковые функции также подойдут при работе с кодировкой UTF-8:

- str_replace() — для замены в строке;
- htmlspecialchars() — для замены специальных символов их HTML-эквивалентами. Кодировка указывается в третьем параметре;
- trim(), ltrim() и rtrim() — для удаления пробельных символов в начале и (или) конце строки. Если во втором параметре указать список символов (например, русских букв), то функции будут работать некорректно;
- addslashes() — для добавления защитных слэшей перед специальными символами;
- stripslashes() — для удаления защитных слэшей.

Функции trim(), addslashes() и stripslashes() работают корректно, т. к. они удаляют (или добавляют) символы, которые в UTF-8 кодируются одним байтом. Все эти функции мы уже рассматривали в *разд. 5.15.1*. Кроме перечисленных функций, для кодирования и шифрования строк можно применять функции, рассмотренные в *разд. 5.15.6*.

Если для поиска или замены в строке требуются регулярные выражения, то следует применять Perl-совместимые регулярные выражения (PCRE). Так как мы работаем с кодировкой UTF-8, то в параметре <Модификатор> обязательно должен присутствовать модификатор u. В качестве примера удалим все русские буквы из строки:

```
$str = 'строка1строка2строка3';
echo preg_replace('#[а-яе]#isu', '', $str); // 123
```

> ***Обратите внимание***
>
> Регистр модификатора u имеет значение.

Если в этом примере модификатор u не указать, то будет удален один байт из каждого двухбайтового символа и в итоге в строке появятся квадратики или знаки вопроса.

5.15.11. Перегрузка строковых функций

Некоторые функции, предназначенные для работы с однобайтовыми кодировками, можно перегрузить в файле конфигурации php.ini или с помощью файла .htaccess. После этого функции могут корректно работать с многобайтовыми кодировками. Перегрузка функций осуществляется с помощью директивы `mbstring.func_overload`, которая может принимать следующие значения:

- `0` — без перегрузки (значение по умолчанию);
- `1` — функция для отправки писем `mail()` будет эквивалентна функции `mb_send_mail()`;
- `2` — будут перегружены строковые функции. Список функций приведен в табл. 5.1;
- `4` — перегрузка функций, предназначенных для работы с регулярными выражениями формата POSIX. Список функций приведен в табл. 5.2. Эти функции лучше заменять функциями, предназначенными для работы с Perl-совместимыми регулярными выражениями;
- `7` — все указанные ранее функции будут перегружены.

Таблица 5.1. *Перегрузка строковых функций*

Функция	Перегружается в
`strlen()`	`mb_strlen()`
`substr()`	`mb_substr()`
`strtoupper()`	`mb_strtoupper()`
`strtolower()`	`mb_strtolower()`
`strpos()`	`mb_strpos()`
`strrpos()`	`mb_strrpos()`
`substr_count()`	`mb_substr_count()`

Таблица 5.2. *Перегрузка функций, предназначенных для работы с регулярными выражениями формата POSIX*

Функция	Перегружается в
`ereg()`	`mb_ereg()`
`eregi()`	`mb_eregi()`
`ereg_replace()`	`mb_ereg_replace()`
`eregi_replace()`	`mb_eregi_replace()`
`split()`	`mb_split()`

Для корректной работы функций после перегрузки необходимо указать кодировку в директиве `mbstring.internal_encoding`.

5.16. Функции для работы с числами

Перечислим основные функции для работы с числами:

- `sin()`, `cos()`, `tan()` — стандартные тригонометрические функции (синус, косинус, тангенс). Значение указывается в радианах;
- `asin()`, `acos()`, `atan()` — обратные тригонометрические функции (арксинус, арккосинус, арктангенс). Значение возвращается в радианах;
- `exp()` — экспонента;
- `log()` — натуральный логарифм;
- `pow(<Число>, <Степень>)` — возведение `<Числа>` в `<Степень>`;
- `sqrt()` — квадратный корень;
- `pi()` — возвращает число π;
- `abs()` — абсолютное значение;
- `ceil()` — значение, округленное до ближайшего большего целого;
- `floor()` — значение, округленное до ближайшего меньшего целого;
- `max(<Список чисел через запятую>)` — максимальное значение из списка;
- `min(<Список чисел через запятую>)` — минимальное значение из списка;
- `mt_rand(<Начало диапазона>, <Конец диапазона>)` — случайное число от `<Начало диапазона>` до `<Конец диапазона>` включительно:

```
echo mt_rand(10, 100);
```

Для примера создадим генератор паролей произвольной длины (листинг 5.14). Для этого добавляем в массив `$mass` все разрешенные символы, а далее в цикле получаем содержимое массива по случайному индексу. По умолчанию будет выдаваться пароль из восьми символов.

Листинг 5.14. Генератор паролей

```
function f_passw_generator($count_char=8) {
   $mass =
   array('a','b','c','d','e','f','g','h','i','j','k','l',
   'm','n','o','p','q','r','s','t','u','v','w','x','y','z',
   'A','B','C','D','E','F','G','H','I','J','K','L',
   'M','N','O','P','Q','R','S','T','U','V', 'W',
   'X','Y','Z','1','2','3','4','5','6','7','8','9','0');
   $passw = '';
   $count = count($mass)-1;
   for ($i=0; $i<$count_char; $i++) {
      $passw .= $mass[mt_rand(0, $count)];
   }
```

```
    return $passw;
}
echo f_passw_generator(10); // Выведет что-то вроде JNtX7DvSsE
```

- `mt_srand(<Параметр>)` — настраивает генератор случайных чисел на новую последовательность. Параметром обычно служит функция `time()`, возвращающая число секунд, прошедшее с 1 января 1970 г.:

```
mt_srand(time());
echo mt_rand(10, 100);
```

- `base_convert()` — позволяет преобразовать число, записанное в одной системе счисления, в другую. Имеет следующий формат:

```
base_convert(<Содержащая число строка>,
      <Исходная система счисления>, <Нужная система счисления>);
```

Пример:

```
$var = base_convert(9, 10, 2);
echo $var; // Выведет 1001
$var = base_convert("A", 16, 10);
echo $var; // Выведет 10
```

- `bindec()` — преобразует двоичное число в десятичное:

```
echo bindec("1001"); // Выведет 9
```

- `decbin()` — преобразует десятичное число в двоичное:

```
echo decbin("9"); // Выведет 1001
```

- `hexdec()` — преобразует шестнадцатеричное число в десятичное:

```
echo hexdec("1b"); // Выведет 27
```

- `dechex()` — преобразует десятичное число в шестнадцатеричное:

```
echo dechex(27); // Выведет 1b
```

- `octdec()` — преобразует восьмеричное число в десятичное;
- `decoct()` — преобразует десятичное число в восьмеричное;
- `number_format()` — позволяет преобразовать число в отформатированную строку. Имеет следующий синтаксис:

```
number_format(<Число>[, <Число знаков после запятой>[,
              <Десятичный разделитель>, <Разделитель тысяч>]])
```

Пример:

```
$x = 1234567.126;
echo number_format($x) . '<br>'; // Выведет: "1,234,567"
echo number_format($x, 2) . '<br>'; // Выведет: "1,234,567.13"
echo number_format($x, 2, ',', ' ');
// Выведет: "1 234 567,13"
```

5.17. Функции для работы с датой и временем. Получение текущей даты, даты создания файла и проверка корректности введенной даты

Для работы с датами в PHP чаще всего применяются следующие функции:

- `date_default_timezone_set(<Зона>)` — настраивает зону местного времени:

```
date_default_timezone_set('Europe/Moscow');
```

- `time()` — возвращает число секунд, прошедшее с 1 января 1970 г.:

```
echo time();
```

- `date(<Строка формата даты/времени>, [<Исходная дата>])` — возвращает дату и время.

  В параметре `<Строка формата даты/времени>` могут быть использованы следующие служебные символы:

  - `U` — число секунд, прошедшее с 1 января 1970 г.;
  - `Y` — год из 4 цифр;
  - `y` — год из 2 цифр;
  - `z` — день с начала года (от 0 до 365);
  - `F` — название месяца по-английски;
  - `m` — номер месяца с предваряющим нулем (от "01" до "12");
  - `n` — номер месяца без предваряющего нуля (от "1" до "12");
  - `M` — аббревиатура месяца из 3-х букв по-английски;
  - `d` — номер дня с предваряющим нулем (от "01" до "31");
  - `j` — номер дня без предваряющего нуля (от "1" до "31");
  - `l` — название дня недели по-английски;
  - `w` — номер дня недели (0 — для воскресенья и 6 — для субботы);
  - `D` — аббревиатура дня недели из трех букв по-английски;
  - `A` — "AM" (до полудня) или "PM" (после полудня);
  - `a` — "am" (до полудня) или "pm" (после полудня);
  - `H` — часы в 24-часовом формате (от "00" до "23");
  - `h` — часы в 12-часовом формате (от "01" до "12");
  - `i` — минуты (от "00" до "59");
  - `s` — секунды (от "00" до "59").

```
echo date("U - Y - y - z - F - m - n - M - d - j") . "<br>";
echo date("l - w - D - A - a - H - h - i - s");
// Выведет:
// 1226784874 - 2008 - 08 - 320 - November - 11 - 11 - Nov - 16 - 16
// Sunday - 0 - Sun - AM - am - 00 - 12 - 34 - 34
```

Выведем текущую дату и время таким образом, чтобы день недели и месяц были написаны по-русски (листинг 5.15).

Листинг 5.15. Вывод текущей даты

```
date_default_timezone_set('Europe/Moscow');
$day = array('воскресенье', 'понедельник', 'вторник', 'среда', 'четверг',
             'пятница', 'суббота');
$month = array('', 'января', 'февраля', 'марта', 'апреля', 'мая', 'июня',
              'июля', 'августа', 'сентября', 'октября', 'ноября',
              'декабря');
$date = 'Сегодня<br>';
$date .= $day[(int)date('w')];
$date .= date(' d ');
$date .= $month[(int)date('n')];
$date .= date(' Y');
$date .= date(' H:i:s') . '<br>' . date('d.m.y');
echo $date;
```

Код, приведенный в листинге 5.15, выведет

```
Сегодня
понедельник 16 ноября 2009 00:37:08
16.11.09
```

Если в функции `date()` указан второй параметр, то дата будет не текущая, а соответствующая значению из второго параметра. Например, в функцию можно передать число секунд, прошедших с 1 января 1970 г., и получить любое другое форматирование даты.

```
$date = date("Дата d-m-Y", 1180986651);
echo $date;
// Выведет Дата 04-06-2007
```

Или можно получить дату создания файла:

```
$date = date("Дата d-m-Y", filectime("index.php"));
echo $date;
// Выведет Дата 20-06-2008
```

Функция `strftime()` позволяет отформатировать дату в зависимости от настроек локали. Формат вызова:

```
strftime(<Строка формата даты/времени>[,
         <Число секунд, прошедшее с 1 января 1970 г.>])
```

Если второй параметр не указан, то используется значение, возвращаемое функцией `time()`. В параметре `<Строка формата даты/времени>` могут быть указаны следующие служебные символы:

- `%y` — год из двух цифр;
- `%Y` — год из четырех цифр;
- `%j` — день с начала года (от "001" до "366");
- `%m` — номер месяца с предваряющим нулем (от "01" до "12");
- `%b` — сокращенное название месяца в текущей локали:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
echo strftime("%b"); // ноя
```

- `%B` — полное название месяца в текущей локали:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
echo strftime("%B"); // Ноябрь
```

- `%W` — номер недели в году. Первый понедельник года считается первым днем первой недели;
- `%d` — номер дня с предваряющим нулем (от "01" до "31");
- `%a` — сокращенное название дня недели в текущей локали:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
echo strftime("%a"); // Вт
```

- `%A` — полное название дня недели в текущей локали:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
echo strftime("%A"); // вторник
```

- `%H` — часы в 24-часовом формате (от "00" до "23");
- `%I` — часы в 12-часовом формате (от "01" до "12");
- `%M` — минуты (от "00" до "59");
- `%S` — секунды (от "00" до "59");
- `%Z` — временная зона:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
echo strftime("%Z"); // Московское время (зима)
```

- `%c` — формат даты и времени по умолчанию в текущей локали:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
echo strftime("%c"); // 17.11.2009 18:14:25
```

- `%x` — представление даты в текущей локали:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
echo strftime("%x"); // 17.11.2009
```

- `%X` — представление времени в текущей локали:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
echo strftime("%X"); // 18:14:25
```

- `%%` — символ "%".

Функция `checkdate()` позволяет проверить корректность введенной даты. Формат функции:

```
checkdate(<Месяц>, <День>, <Год>);
```

Функция возвращает `true`, если дата, заданная аргументами, является правильной. Дата считается правильной, если:

- год в диапазоне от 1 до 32 767 включительно;
- месяц в диапазоне от 1 до 12 включительно;
- день является допустимым номером дня для месяца, заданного аргументом `<Месяц>`:

```
if (checkdate(5, 32, 2008)) echo "Дата правильная";
else echo "Нет"; // Т. к. даты 32.05.2008 нет, выведет: Нет
```

5.18. Функции. Разделение программы на фрагменты

Функция — это фрагмент кода PHP, который можно вызвать из любого места программы. Функция описывается с помощью ключевого слова `function` по следующей схеме:

```
function <Имя функции> ([<Параметры>]) {
   <Тело функции>
   [return <Значение>]
}
```

5.18.1. Основные понятия

Имя функции должно быть уникальным и может содержать только буквы, цифры, символ подчеркивания и не может начинаться с цифры. Основное различие между именами функций и переменных заключается в значении регистра символов. Имена переменных зависят от регистра, а названия функций не зависят.

Например, следующие имена функций одинаковы:

```
StripSlashes()
stripslashes()
```

После имени функции в круглых скобках можно указать один или несколько параметров через запятую. Параметров может вообще не быть. В этом случае указываются только круглые скобки.

Между фигурными скобками располагаются выражения PHP. Кроме того, функция может возвращать значение при ее вызове. Возвращаемое значение задается с помощью оператора возврата `return`.

Пример функции без параметров:

```
function f_print_OK() {
   echo "Сообщение при удачно выполненной операции";
}
```

Пример функции с параметром:

```
function f_print($msg) {
   echo $msg;
}
```

Пример функции, возвращающей сумму двух переменных:

```
function f_Sum($x, $y) {
   $z = $x + $y;
   return $z;
}
```

В качестве возвращаемого значения в операторе возврата `return` можно указывать не только имя переменной, но и выражение:

```
function f_Sum($x, $y) {
   return ($x + $y);
}
```

В программе функции можно вызвать следующим образом:

```
f_print_OK();
f_print("Сообщение");
$var = f_Sum(5, 2); // Переменной $var будет присвоено значение 7
```

Выражения, указанные после оператора `return`, никогда не будут выполнены:

```
function f_Sum($x, $y) {
   return ($x + $y);
   echo "Сообщение"; // Это выражение никогда не будет выполнено
}
```

Имя переменной, передающей значение функции, может не совпадать с именем переменной внутри функции:

```
function f_Sum($x, $y) {
   return ($x + $y);
}
$var1 = 5;
$var2 = 2;
$var3 = f_Sum($var1, $var2);
```

Некоторые параметры функции могут быть необязательными. Для этого при определении функции такому параметру необходимо присвоить начальное значение.

Например, переделаем наш предыдущий пример и сделаем второй параметр необязательным:

```
function f_Sum($x, $y=2) {
   return ($x + $y);
}
$var1 = 5;
$var3 = f_Sum($var1); // Переменной $var3 будет присвоено значение 7
$var4 = f_Sum($var1, 5); // Переменной $var4 будет присвоено значение 10
```

Таким образом, если второй параметр не задан, его значение будет равно 2.

5.18.2. Расположение описаний функций

Обычно описания функций принято располагать в начале файла или в отдельном файле. Хотя функции могут находиться в любом месте файла, даже после места вызова функции:

```
$var1 = 5;
$var3 = f_Sum($var1); // Переменной $var3 будет присвоено значение 7
function f_Sum($x, $y=2) {
   return ($x + $y);
}
```

5.18.3. Операторы *require* и *include*. Выносим функции в отдельный файл. Создаем шаблоны для множества страниц

Если функции вынесены в отдельный файл, то подключить его позволяют два оператора: `require` и `include`. Операторы имеют следующий формат:

```
require(<Имя файла>);
require <Имя файла>;
include(<Имя файла>);
include <Имя файла>;
```

Вынесем функцию `f_Sum()` в отдельный файл (листинг 5.17) и подключим его с помощью оператора `require` (листинг 5.16).

Листинг 5.16. Использование оператора `require`

```
<?php
require("script.inc");
$var1 = 5;
$var2 = f_Sum($var1);
echo $var2;
?>
```

Листинг 5.17. Содержимое файла script.inc

```
<?php
function f_Sum($x, $y=2) {
   return ($x + $y);
}
?>
```

Создать файл script.inc можно, например, с помощью Notepad++. Следует отметить, что подключаемый файл может иметь любое расширение, хотя общепринято давать подключаемым файлам расширение inc (от "include").

Попробуем открыть файл script.inc с помощью Web-браузера. В итоге в окне Web-браузера отобразится исходный код:

```
<?php
function f_Sum($x, $y=2) {
   return ($x + $y);
}
?>
```

По этой причине необходимо размещать включаемые файлы в каталоге, доступном только для сценария, но недоступном через Интернет. При установке и настройке PHP мы указали местонахождение включаемых файлов в директиве `include_path` файла php.ini:

```
include_path = ".;C:\php5\includes"
```

Здесь через точку с запятой указано два места для поиска включаемых файлов:

- . (точка) — в той же папке, что и исполняемый файл;
- `C:\php5\includes` — в папке includes (c:\php5\includes).

Иными словами, не найдя включаемого файла в той папке, где расположен исполняемый файл, интерпретатор выполнит поиск в папке includes (C:\php5\includes).

Можно также сохранить включаемый файл с расширением php. В этом случае исходный код не будет отображаться в окне Web-браузера.

Если включаемый файл содержит исполняемый код, то указывать PHP-дескрипторы нужно обязательно. Иначе PHP-код будет выведен как обычный текст, а при вызове определенных в нем функций отобразится сообщение об ошибке:

```
function f_Sum($x, $y=2) { return ($x + $y); }
Fatal error: Call to undefined function f_Sum() in
C:\Apache2\htdocs\index.php on line 9
```

Иными словами, во включаемом файле может и не быть кода PHP. Для примера вынесем верхний колонтитул и функцию `f_Sum()` в файл header.inc (листинг 5.19), а нижний колонтитул — в файл footer.inc (листинг 5.20). Затем подключим эти файлы к основному сценарию (листинг 5.18).

Листинг 5.18. Размещение HTML-кода во включаемом файле

```
<?php
require("header.inc");
$var1 = 5;
$var2 = f_Sum($var1);
echo $var2;
require("footer.inc");
?>
```

Листинг 5.19. Содержимое файла header.inc

```
<html>
<head>
<title>Функции</title>
</head>
<body>
<?php
function f_Sum($x, $y=2) {
   return ($x + $y);
}
?>
```

Листинг 5.20. Содержимое файла footer.inc

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

В листинге 5.21 приведен результирующий HTML-код, сформированный после выполнения предыдущей программы.

Листинг 5.21. Результирующий HTML-код

```
<html>
<head>
<title>Функции</title>
</head>
<body>
7</body>
</html>
```

Таким образом можно сформировать шаблоны для множества страниц. Интерпретатор, встретив оператор `require`, сделает содержимое включаемого файла частью страницы. Если файл не может быть загружен, то оператор генерирует неисправимую ошибку, и сценарий прекращает работу.

Вместо оператора `require` можно использовать оператор `include`. Если включаемый файл не найден, оператор выведет сообщение об ошибке, но сценарий будет выполняться далее. Если данный файл содержит функции, то каждый вызов функции из этого файла также будет генерировать ошибку.

Оператор `include` возвращает `true`, если файл загружен, и `false` в случае ошибки. Подавить сообщение об ошибке можно с помощью оператора `@` (листинг 5.22).

Листинг 5.22. Подавление сообщения об ошибке

```
<?php
if (@include("header.inc")){
   $var1=5;
   $var2 = f_Sum($var1);
   echo $var2;
}
require("footer.inc");
?>
```

5.18.4. Операторы *require_once* и *include_once*

Операторы `require_once` и `include_once` работают точно так же, как `require` и `include`. Однако перед включением файла интерпретатор проверяет, включался ли уже этот файл или нет. Если да, то файл не будет подключен. Операторы имеют следующий формат:

```
require_once(<Имя файла>);
require_once <Имя файла>;
include_once(<Имя файла>);
include_once <Имя файла>;
```

Пример:

```
require_once("script.inc");
$var1 = 5;
$var2 = f_Sum($var1);
echo $var2;
include_once("footer.inc");
```

Применять операторы `require_once` и `include_once` удобно при разработке больших проектов, т. к. в одном из файлов включаемый файл, возможно, уже был подключен. Подключение одного файла дважды может привести к ошибкам, которые очень трудно найти.

5.18.5. Рекурсия. Вычисляем факториал

Рекурсия — это возможность функции вызывать саму себя. С одной стороны, это удобно, с другой стороны, если не предусмотреть условие выхода, то возникнет бесконечный цикл.

Для примера приведем вычисление факториала (листинг 5.23).

Листинг 5.23. Вычисление факториала

```
<b>Вычисление факториала</b><br><br>
Введите число<br>
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="text" name="fact">
<input type="submit" value="OK">
</form><br>
<?php
function f_Factorial($x) {
   if ($x == 0 || $x == 1) return 1;
   else return ($x * f_Factorial($x - 1));
}
if (isset($_GET['fact'])) {
   $fact = $_GET['fact'];
   if (!preg_match('/^[0-9]+$/s', $fact)) {
      echo 'Необходимо ввести целое число!';
   }
   else {
      echo 'Факториал числа ' . $fact . ' = ' . f_Factorial((int)$fact);
   }
}
?>
```

5.18.6. Глобальные и локальные переменные. Использование глобальных переменных внутри функций

Глобальными называют переменные, объявленные вне функции. В PHP глобальные переменные видны в любой части программы, кроме функций.

Локальные переменные объявлены внутри функции и видны только в теле функции. Если имена локальной и глобальной переменной совпадают, то все операции внутри функции осуществляются с локальной переменной, а значение глобальной не изменяется.

Листинг 5.24 демонстрирует область видимости переменных.

Листинг 5.24. Глобальные и локальные переменные

```
<?php
function f_Sum() {
   $var1 = 5;
   $number = 1;
   echo 'Локальная переменная $var1 = ' . $var1 . '<br>';
```

```
   echo 'Локальная переменная $number = ' . $number . '<br>';
   echo 'Глобальная переменная $var2 = ' . gettype($var2);
   echo ', т. е. не видна внутри функции<br>';
   return ($var1 + $number);
}
$var1 = 10;
echo 'Глобальная переменная $var1 = ' . $var1 . '<br>';
$var2 = 7;
$var3 = f_Sum(); // Вызов функции
echo 'Глобальная переменная $var1 осталась = ' . $var1 . '<br>';
echo 'Локальная переменная $number = ' . gettype($number);
echo ', т. е. не видна вне тела функции';
?>
```

В окне Web-браузера получим следующий результат:

```
Глобальная переменная $var1 = 10
Локальная переменная $var1 = 5
Локальная переменная $number = 1

Notice: Undefined variable: var2 in C:\Apache2\htdocs\test.php on line 7
Глобальная переменная $var2 = NULL, т. е. не видна внутри функции
Глобальная переменная $var1 осталась = 10
Notice: Undefined variable: number in C:\Apache2\htdocs\
test.php on line 16
Локальная переменная $number = NULL, т. е. не видна вне тела функции
```

Как видно из примера, переменная `$number`, объявленная внутри функции `f_Sum()`, не доступна вне функции. Объявление внутри функции локальной переменной `$var1` не изменило значения одноименной глобальной переменной. А глобальная переменная `$var2` не видна внутри функции `f_Sum()`.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Две строки в результатах вывода начинаются со слова "Notice". Таким образом интерпретатор предупреждает нас о неопределенной переменной.

Для того чтобы глобальная переменная была видна внутри функции, необходимо перед именем переменной в теле функции указать ключевое слово `global`. Продемонстрируем это на примере (листинг 5.25).

Листинг 5.25. Использование глобальных переменных внутри функции

```
<?php
function f_Sum() {
   global $var1;
   $number = 2;
   echo 'Глобальная переменная $var1 внутри функции = ';
   echo $var1 . '<br>';
```

```
   $var1 += $number;
}
$var1 = 10;
echo 'Глобальная переменная $var1 вне функции = ' . $var1 . '<br>';
f_Sum(); // Вызов функции
echo 'Значение переменной $var1 после функции = ' . $var1 . '<br>';
?>
```

В окне Web-браузера получим следующий результат:

```
Глобальная переменная $var1 вне функции = 10
Глобальная переменная $var1 внутри функции = 10
Значение переменной $var1 после функции = 12
```

Внутри функции к глобальной переменной можно обратиться через суперглобальный массив `$GLOBALS` (листинг 5.26).

Листинг 5.26. Использование суперглобального массива `$GLOBALS`

```
<?php
function f_Sum() {
   $number = 2;
   echo 'Глобальная переменная $var1 внутри функции = ';
   echo $GLOBALS['var1'] . '<br>';
   $GLOBALS['var1'] += $number;
}
$var1 = 10;
echo 'Глобальная переменная $var1 вне функции = ' . $var1 . '<br>';
f_Sum();
echo 'Значение переменной $var1 после функции = ' . $var1 . '<br>';
?>
```

В итоге, в окне Web-браузера получим такой же результат:

```
Глобальная переменная $var1 вне функции = 10
Глобальная переменная $var1 внутри функции = 10
Значение переменной $var1 после функции = 12
```

Если создать новый элемент в массиве `$GLOBALS`, то автоматически будет создана глобальная переменная с таким же названием:

```
if (!isset($var1)) echo 'Переменная $var1 не определена<br>';
$GLOBALS['var1'] = 10;
echo 'Значение переменной $var1 = ' . $var1;
```

В итоге, в окне Web-браузера получим результат:

```
Переменная $var1 не определена
Значение переменной $var1 = 10
```

5.18.7. Статические переменные

Если внутри функции объявлена статическая переменная, то после завершения работы функции она не будет удалена и сохранит свое значение. Объявляется статическая переменная с помощью ключевого слова `static`, которое указывается перед именем переменной.

Выведем все четные числа от 1 до 100 (листинг 5.27).

Листинг 5.27. Использование статических переменных

```
<?php
function f_Sum() {
   static $var;
   $number = 2;
   $var += $number;
   echo $var . "<br>\n";
}
for ($i=0; $i<50; $i++) f_Sum();
?>
```

5.18.8. Переменное число параметров в функции. Сумма произвольного количества чисел

Функции `func_get_args()` и `func_get_arg()` позволяют получить доступ ко всем параметрам, переданным функции (листинг 5.28). Функция `func_num_args()` дает возможность определить общее число параметров, переданных функции.

Листинг 5.28. Использование функции `func_get_arg()`

```
<?php
function f_Sum($var1, $var2) {
   return func_get_arg(0)+func_get_arg(1);
}
echo f_Sum(5, 6); // Выведет: 11
?>
```

Какой в этом смысл? Дело в том, что при использовании этих функций можно передать нашей функции больше аргументов, чем первоначально объявлено. Можно, например, просуммировать сразу несколько чисел, а не только два (листинг 5.29).

Листинг 5.29. Переменное число параметров в функции

```
<?php
function f_Sum($var1, $var2) {
   $sum = 0;
   $count = func_num_args();
```

```
   for ($i=0; $i<$count; $i++) {
      $sum += func_get_arg($i);
   }
   return $sum;
}
echo f_Sum(5, 6, 7, 20); // Выведет 38
?>
```

Такой же результат можно получить, используя функцию `func_get_args()` (листинг 5.30).

Листинг 5.30. Использование функции `func_get_args()`

```
<?php
function f_Sum($var1, $var2) {
   $sum = 0;
   foreach (func_get_args() as $val) {
      $sum += $val;
   }
   return $sum;
}
echo f_Sum(5, 6, 7, 20); // Выведет 38
?>
```

5.19. Условные операторы. Выполнение блоков кода только при соответствии условию

Условные операторы позволяют в зависимости от значения логического выражения выполнить отдельный участок программы или, наоборот, не выполнять его. Логические выражения возвращают только два значення: `true` или `false`.

5.19.1. Операторы сравнения

Операторы сравнения входят в состав логических выражений. Перечислим их:

- `==` — равно;
- `===` — строго равно;
- `!=` — не равно;
- `<>` — не равно;
- `!==` — строго не равно;
- `<` — меньше;
- `>` — больше;

- `<=` — меньше или равно;
- `>=` — больше или равно.

В чем отличие оператора == (равно) от оператора === (строго равно)? Если используется оператор ==, интерпретатор пытается преобразовать разные типы данных к одному и лишь затем сравнивает их. Оператор ===, встретив данные разных типов, сразу возвращает `false`.

Значение логического выражения можно инвертировать с помощью оператора `!`:

```
!($var1 == $var2)
```

Если переменные `$var1` и `$var2` равны, то возвращается значение `true`, но так как перед выражением стоит оператор `!`, выражение вернет `false`.

Можно несколько логических выражений объединить в одно большое с помощью следующих операторов:

- `&&` — логическое И;
- `||` — логическое ИЛИ.

Это выражение вернет `true` только в случае, если оба выражения вернут `true`:

```
($var1 == $var2) && ($var2 != $var3)
```

А это выражение вернет `true`, если хотя бы одно из выражений вернет `true`:

```
($var1 == $var2) || ($var3 == $var4)
```

Оператор `&&` можно заменить логической операцией `AND`, а `||` — логической операцией `OR`:

- `AND` — логическое И, например:

```
($var1 == $var2) AND ($var2 != $var3)
```

- `OR` — логическое ИЛИ, например:

```
($var1 == $var2) OR ($var3 == $var4)
```

5.19.2. Оператор ветвления *if...else*. Проверка выбранного элемента из списка

Оператор ветвления уже встречался ранее в наших примерах, в частности, так мы определяли факт существования переменной. Так как функция `isset()` при существовании переменной возвращает значение `true`, то это условие можно проверить, используя оператор ветвления `if...else`:

```
if (isset($_GET['name'])) {
   echo 'Hello, ' . $_GET['name'];
}
else {
   echo 'Введите ваше имя<br>';
   echo '<form action="' . $_SERVER['SCRIPT_NAME'] . '">';
```

```
   echo '<input type="text" name="name">';
   echo '<input type="submit" value="OK">';
   echo '</form>';
}
```

Обратите внимание, что логическое выражение не содержит операторов сравнения:

```
if (isset($_GET['name'])) {
```

Такая запись эквивалентна следующей:

```
if (isset($_GET['name'])==true) {
```

Проверка на равенство выражения значению `true` выполняется по умолчанию.

Оператор ветвления `if...else` имеет следующий формат:

```
if (<Логическое выражение>) {
   <Блок, выполняемый, если условие истинно>
}
[elseif (<Логическое выражение>) {
   <Блок, выполняемый, если условие истинно>
}]
[else {
   <Блок, выполняемый, если все условия ложны>
}]
```

Для примера напишем программу, которая проверяет, является ли введенное пользователем число четным или нет (листинг 5.31). После проверки выводится соответствующее сообщение.

Листинг 5.31. Проверка числа на четность

```
<b>Проверка числа на четность</b><br><br>
Введите число<br>
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="text" name="var">
<input type="submit" value="OK">
</form><br>
<?php
if (isset($_GET['var'])) {
   $var = $_GET['var'];
   if (preg_match('/^[0-9]+$/s', $var)) {
      // Преобразуем тип string (строка) в integer (число)
      $var = intval($var);
      if ($var%2 == 0) {
         echo $var . ' — Четное число';
      }
      else {
         echo $var . ' — Нечетное число';
      }
   }
```

```
   else echo 'Необходимо ввести число';
}
?>
```

Как видно из примера, один условный оператор можно вложить в другой. Кроме того, если блок состоит из одного выражения, фигурные скобки можно не указывать:

```
if ($var%2 == 0) echo $var . ' — Четное число';
else echo $var . ' — Нечетное число';
```

Блок `else` может отсутствовать:

```
if ($var%2 == 0) echo $var . ' — Четное число';
```

Кроме того, оператор `if...else` позволяет проверить сразу несколько условий. Рассмотрим это на примере (листинг 5.32).

Листинг 5.32. Проверка введенного значения

```
<b>Какой операционной системой вы пользуетесь?</b><br><br>
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<select name="os">
<option value="0" selected>Не выбрано</option>
<option value="1">Windows XP</option>
<option value="2">Windows Vista</option>
<option value="3">Windows 7</option>
<option value="4">Windows 8</option>
<option value="5">Другая</option>
</select>
<input type="submit" value="Выбрал">
</form>
<?php
if (isset($_GET['os'])) {
   $os = $_GET['os'];
   if ($os == '1') echo 'Вы выбрали — Windows XP';
   elseif ($os == '2') echo 'Вы выбрали — Windows Vista';
   elseif ($os == '3') echo 'Вы выбрали — Windows 7';
   elseif ($os == '4') echo 'Вы выбрали — Windows 8';
   elseif ($os == '5') echo 'Вы выбрали - Другая';
   elseif ($os == '0') echo 'Вы не выбрали операционную систему';
   else echo 'Мы не смогли определить вашу операционную систему';
}
?>
```

С помощью оператора `elseif` мы можем определить выбранное в списке значение и вывести соответствующее сообщение.

5.19.3. Оператор ? Проверка числа на четность

Оператор ? имеет следующий формат:

```
<Переменная> = (<Логическое выражение>) ? <Выражение если Истина> :
<Выражение если Ложь>;
```

Перепишем нашу программу (листинг 5.32) и используем оператор ? вместо if...else (листинг 5.33).

Листинг 5.33. Использование оператора ?

```
<b>Проверка числа на четность</b><br><br>
Введите число<br>
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="text" name="var">
<input type="submit" value="OK">
</form><br>
<?php
if (isset($_GET['var'])) {
   $var = $_GET['var'];
   if (preg_match('/^[0-9]+$/s', $var)) {
      // Преобразуем тип string (строка) в integer (число)
      $var = intval($var);
      echo ($var%2 == 0) ? $var . ' — Четное число'
                         : $var . ' — Нечетное число';
   }
   else echo 'Необходимо ввести число';
}
?>
```

Рассмотрим еще один пример. Предположим, необходимо вывести сообщение при возникновении определенного условия. Если попробовать вывести так

```
$var = 5;
($var == 5) ? echo 'Равно' : echo 'Не равно'; // Ошибка
```

то возникнет ошибка. Обойти эту ошибку можно способом, который мы рассмотрели в листинге 5.33, или заменой оператора echo на print:

```
$var = 5;
($var == 5) ? print 'Равно' : print 'Не равно';
```

Средний параметр можно не указывать:

```
$var = $_GET['var'] ? : 'Значение по умолчанию';
echo $var;
```

Если переменная $_GET['var'] не существует, то переменная $var будет иметь значение "Значение по умолчанию", а если определена, то значение переменной $var

будет равно значению переменной `$_GET['var']`. Такая короткая запись эквивалентна следующему коду:

```
$var = $_GET['var'] ? $_GET['var'] : 'Значение по умолчанию';
echo $var;
```

При использовании короткой записи следует учитывать, что для несуществующей переменной будет выведено предупреждающее сообщение:

```
Notice: Undefined index: var
```

5.19.4. Оператор выбора *switch*. Использование оператора *switch* вместо *if...else*

Оператор выбора `switch` имеет следующий формат:

```
switch (<Переменная или выражение>) {
   case <Значение 1>:
      <Выражение 1>;
      break;
   case <Значение 2>:
      <Выражение 2>;
      break;
...
   default:
      <Выражение>;
}
```

Перепишем нашу программу определения операционной системы (листинг 5.32), заменив оператор `if...else` на `switch` (листинг 5.34).

Листинг 5.34. Использование оператора `switch`

```
<b>Какой операционной системой вы пользуетесь?</b><br><br>
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<select name="os">
<option value="0" selected>Не выбрано</option>
<option value="1">Windows XP</option>
<option value="2">Windows Vista</option>
<option value="3">Windows 7</option>
<option value="4">Windows 8</option>
<option value="5">Другая</option>
</select>
<input type="submit" value="Выбрал">
</form>
<?php
if (isset($_GET['os'])) {
   switch($_GET['os']) {
```

```
    case '1':
       echo 'Вы выбрали – Windows XP'; break;
    case '2':
       echo 'Вы выбрали – Windows Vista'; break;
    case '3':
       echo 'Вы выбрали – Windows 7'; break;
    case '4':
       echo 'Вы выбрали – Windows 8'; break;
    case '5':
       echo 'Вы выбрали – Другая'; break;
    case '0':
       echo 'Вы не выбрали операционную систему'; break;
    default:
       echo 'Мы не смогли определить вашу операционную систему';
    }
}
?>
```

Вместо логического выражения оператор `switch` принимает переменную или выражение. В зависимости от значения переменной (или выражения) выполняется один из блоков `case`, в котором указано это значение. Если ни одно из значений не описано в блоках `case`, то выполняется блок `default`. Оператор `break` позволяет досрочно выйти из оператора выбора `switch`. Зачем это нужно? Если не указать оператор `break` в конце блока `case`, то будет выполняться следующий блок `case` независимо от указанного значения. Если убрать все операторы `break` из нашего примера, то в результате (при выборе Windows XP) в окне Web-браузера отобразится следующая надпись:

```
Вы выбрали – Windows XPВы выбрали – Windows VistaВы выбрали – Windows 7
Вы выбрали – Windows 8Вы выбрали – ДругаяВы не выбрали операционную системуМы
не смогли определить вашу операционную систему
```

Иными словами, оператор `break` следует обязательно указывать в конце блока `case`.

5.20. Операторы циклов. Многократное выполнение блока кода

Операторы циклов уже встречались нам при работе с массивами *(см. разд. 5.14.6)*. Опишем теперь их подробнее.

Предположим, нужно вывести все числа от 1 до 100 по одному на строке. Обычным способом пришлось бы писать 100 строк кода:

```
echo "1<br>\n";
echo "2<br>\n";
...
echo "100<br>\n";
```

С помощью циклов то же действие можно выполнить одной строкой кода:

```
for ($i=1; $i<101; $i++) echo $i . "<br>\n";
```

Иными словами, циклы позволяют выполнить одни и те же выражения многократно.

5.20.1. Цикл *for*

Цикл `for` используется для выполнения выражений определенное число раз. Формат оператора:

```
for (<Начальное значение>; <Условие>; <Приращение>) {
   <Выражения>
}
```

Здесь присутствуют следующие конструкции:

- `<Начальное значение>` — присваивает переменной-счетчику начальное значение;
- `<Условие>` — содержит логическое выражение. Пока логическое выражение возвращает значение `true`, выполняются выражения внутри цикла;
- `<Приращение>` — задает изменение переменной-счетчика при каждой итерации.

Последовательность работы цикла `for`:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие; если оно истинно, выполняются выражения внутри цикла, в противном случае выполнение цикла завершается.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращение>`.
4. Переход к п. 2.

Цикл выполняется до тех пор, пока `<Условие>` не вернет `false`. Если это не случится, цикл будет бесконечным.

`<Приращение>` может не только увеличивать значение переменной-счетчика, но и уменьшать. Выведем все числа от 100 до 1:

```
for ($i=100; $i>0; $i--) echo $i . "<br>\n";
```

`<Приращение>` может изменять значение переменной-счетчика не только на единицу. Выведем все четные числа от 1 до 100:

```
for ($i=2; $i<101; $i+=2) echo $i . "<br>\n";
```

Следует заметить, что выражение, указанное в параметре `<Условие>`, вычисляется на каждой итерации. Рассмотрим вывод элементов массива:

```
$arr = array(1, 2, 3);
for ($i=0; $i<count($arr); $i++) {
   if ($i == 0) {
      $arr[] = 4; // Добавляем новые элементы
      $arr[] = 5; // для доказательства
   }
   echo $arr[$i] . " ";
} // Выведет: 1 2 3 4 5
```

В этом примере мы указываем функцию `count()` в параметре `<Условие>`, а внутри цикла (чтобы доказать вычисление на каждой итерации) добавляем новые элементы в массив. В итоге получили все элементы массива, включая новые элементы. Чтобы этого избежать, следует вычисление размера массива указать в первом параметре:

```
$arr = array(1, 2, 3);
for ($i=0, $c=count($arr); $i<$c; $i++) {
   if ($i == 0) {
      $arr[] = 4; // Добавляем новые элементы
      $arr[] = 5; // для доказательства
   }
   echo $arr[$i] . " ";
} // Выведет: 1 2 3
```

5.20.2. Цикл *while*

Выполнение выражений в цикле `while` продолжается до тех пор, пока логическое выражение истинно. Формат оператора:

```
<Начальное значение>;
while (<Условие>) {
   <Выражения>;
   <Приращение>;
}
```

Последовательность работы цикла `while`:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие; если оно истинно, выполняются выражения внутри цикла, иначе выполнение цикла завершается.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращение>`.
4. Переход к п. 2.

Выведем все числа от 1 до 100, используя цикл `while`:

```
$i = 1;
while ($i<101) {
   echo $i . "<br>\n";
   $i++;
}
```

> ***Внимание!***
>
> Если `<Приращение>` не указано, то цикл будет бесконечным.

В качестве `<Приращение>` не обязательно должна быть арифметическая операция. Например, при работе с базами данных в качестве `<Приращение>` будет перемещение к следующей строке, а условием выхода из цикла — последняя строка в базе данных. В этом случае `<Начальное значение>` — получение первой строки базы данных.

5.20.3. Цикл *do...while*

Выражения в теле цикла `do...while` выполняются до тех пор, пока логическое выражение истинно. Но, в отличие от цикла `while`, условие проверяется не в начале цикла, а в конце. Поэтому выражения внутри цикла `do...while` выполнятся минимум один раз.

Формат оператора `do...while`:

```
<Начальное значение>;
do {
   <Выражения>;
   <Приращение>;
} while (<Условие>);
```

Последовательность работы цикла `do...while`.

1. Переменной-счетчику присваивается начальное значение.
2. Выполняются выражения внутри цикла.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращение>`.
4. Проверяется условие; если оно истинно, происходит переход к п. 2, а если нет — выполнение цикла завершается.

Выведем все числа от 1 до 100, используя цикл `do...while`:

```
$i = 1;
do {
   echo $i . "<br>\n";
   $i++;
} while ($i<101);
```

> ***Внимание!***
> Если `<Приращение>` не указано, то цикл будет бесконечным.

5.20.4. Цикл *foreach*

Цикл `foreach` используется для перебора элементов массива:

```
$Mass = array('Один', 'Два', 'Три', 'Четыре');
foreach ($Mass as $key) {
   echo $key . '<br>';
}
```

Перебрать элементы ассоциативного массива можно следующим образом:

```
$Mass['Один'] = 1;
$Mass['Два'] = 2;
$Mass['Три'] = 3;
$Mass['Четыре'] = 4;
foreach ($Mass as $key => $value) {
   echo $key . ' =&gt; ' . $value . '<br>';
}
```

Если параметр в цикле `foreach` не является массивом, интерпретатор выведет сообщение об ошибке:

```
$Mass = '';
foreach ($Mass as $key => $value) {
   echo $key . ' =&gt; ' . $value . '<br>';
}
// Ошибка: Warning: Invalid argument supplied for foreach()
```

По этой причине перед вызовом цикла `foreach` необходимо проверить тип переменной, например, с помощью функции `is_array()`:

```
if (isset($Mass) && is_array($Mass)) {
   // Проверка существования и типа переменной
   foreach ($Mass as $key => $value) {
      echo $key . ' =&gt; ' . $value . '<br>';
   }
}
```

5.20.5. Оператор *continue*. Переход на следующую итерацию цикла

Оператор `continue` позволяет перейти к следующей итерации цикла до завершения выполнения всех выражений внутри цикла.

Выведем все числа от 1 до 100, кроме чисел от 5 до 10 включительно:

```
for ($i=1; $i<101; $i++) {
   if ($i>4 && $i<11) continue;
   echo $i . "<br>\n";
}
```

5.20.6. Оператор *break*. Прерывание цикла

Оператор `break` позволяет прервать выполнение цикла досрочно.

Для примера выведем все числа от 1 до 100 еще одним способом:

```
for ($i=1; ; $i++) {
   if ($i>100) break;
   echo $i . "<br>\n";
}
```

Здесь мы оставили условие цикла пустым, и это значит, что цикл будет продолжаться бесконечно. Однако благодаря наличию оператора `break` выполнение цикла прерывается, как только будет напечатано 100 строк.

Оператор `break` прерывает выполнение цикла, а не программы, т. е. далее будет выполнено выражение, следующее сразу за циклом.

5.21. Завершение выполнения сценария. Навигация при выборе значения из списка

Для досрочного завершения PHP-сценария предусмотрен оператор `exit`:

```
exit;
exit([<Сообщение>]);
```

Предположим, наш сайт содержит четыре страницы: index.php (главная страница), firm.php (о фирме), price.php (продукция) и contact.php (контактная информация). Реализуем механизм навигации по сайту. Переход на другие страницы будет осуществляться не с помощью ссылок, а нутем выбора нужной страницы из списка. Для этого на всех страницах сайта должна присутствовать форма, описанная в листинге 5.35.

Листинг 5.35. Навигация по сайту с помощью списка

```
<form action="go.php">
<select name="page">
<option value="0" selected>На главную</option>
<option value="1">О фирме</option>
<option value="2">Продукция</option>
<option value="3">Контакты</option>
</select>
<input type="submit" value="Go!">
</form>
```

Далее создаем файл go.php с кодом, приведенным в листинге 5.36.

Листинг 5.36. Содержимое файла go.php

```
<?php
if (isset($_GET['page'])) {
   switch($_GET['page']) {
   case '1':
      header('Location: firm.php'); exit();
   case '2':
      header('Location: price.php'); exit();
   case '3':
      header('Location: contact.php'); exit();
   default:
      header('Location: index.php'); exit();
   }
}
else {
   header('Location: index.php'); exit();
}
?>
```

Теперь при выборе страницы из списка и нажатии кнопки **Go!** мы попадем на нужную страницу. Обратите внимание, что вместо оператора `break` мы использовали оператор `exit`, т. к. после перехода на нужную страницу выполнение остального кода просто лишено смысла.

> ***Внимание!***
>
> Так как функция `header()` устанавливает заголовки ответа сервера, то, кроме указанного кода, в файле go.php не должно быть никаких операторов вывода. Даже пустая строка перед открывающим PHP-дескриптором (`<?php`) вызовет ошибку. Пустые строки внутри PHP-дескрипторов ошибку не генерируют, поскольку вывод информации осуществляется только с помощью операторов `echo` или `print`. Кроме того, если используется кодировка UTF-8, то файл должен быть сохранен в кодировке UTF-8 без BOM.

5.22. Ошибки в программе

Существуют три типа ошибок в скриптах: синтаксические, логические и ошибки времени выполнения.

5.22.1. Синтаксические ошибки

Это ошибки в имени оператора или функции, отсутствие закрывающей или открывающей скобок и т. д., т. е. ошибки в синтаксисе языка. Как правило, интерпретатор предупредит о наличии ошибки. А программа не будет выполняться совсем.

Например, если вместо

```
echo $i . "<br>";
```

написать

```
ecgo $i . "<br>";
```

то Web-браузер отобразит нечто подобное

```
Parse error: syntax error, unexpected T_VARIABLE in
C:\Apache2\htdocs\index.php on line 22
```

Итак, интерпретатор предупреждает нас, что в строке 22 файла index.php содержится ошибка. Достаточно отсчитать 22 строки в исходном коде и исправить опечатку с `ecgo` на `echo`.

Перечислим часто встречающиеся синтаксические ошибки:

- отсутствует точка с занятой в конце выражения;
- опечатка в имени оператора или функции;
- буква набрана в русской раскладке клавиатуры вместо латинской;
- отсутствие открывающей или закрывающей скобки (или наоборот лишние скобки);
- в цикле `for` указаны параметры через занятую, а не через точку с занятой.

5.22.2. Логические ошибки

Это ошибки в логике работы программы, которые можно выявить только по результатам работы скрипта. Как правило, интерпретатор не предупреждает о наличии ошибки. А программа будет выполняться, т. к. не содержит синтаксических ошибок. Такие ошибки достаточно трудно выявить и исправить. Например, в логическом выражении вместо оператора == (равно) указан оператор присваивания =. С точки зрения синтаксиса здесь все правильно.

5.22.3. Ошибки времени выполнения

Это ошибки, которые возникают во время работы скрипта. Их причиной являются события, не предусмотренные программистом. Классическим примером служит деление на ноль.

С помощью оператора `@` можно подавить вывод сообщений о любых ошибках в выражении, которому он предшествует. Например, можно подавить вывод сообщения об ошибке деления на ноль:

```
$val = @(2/0);
```

или

```
@$val = 2/0;
```

Однако после этого значение `$val` не будет иметь смысла (в данном случае `$val` получит значение `false` и логический тип данных), т. е. сама ошибка устранена не будет.

5.22.4. Обработка ошибок

Задать степень обработки и протоколирования ошибок позволяет директива `error_reporting` в файле php.ini:

```
error_reporting = E_ALL & ~E_NOTICE
```

Перечислим значения директивы: `E_ALL` (все ошибки), `E_ERROR` (фатальные ошибки), `E_RECOVERABLE_ERROR`, `E_WARNING` (предупреждения времени выполнения), `E_PARSE` (синтаксические ошибки), `E_NOTICE` (замечания, например, о том, что переменная не инициализирована), `E_CORE_ERROR`, `E_CORE_WARNING`, `E_COMPILE_ERROR`, `E_COMPILE_WARNING`, `E_USER_ERROR`, `E_USER_WARNING`, `E_USER_NOTICE`.

Знак `~` (тильда), стоящий перед значением, указывает, что вывод сообщений об ошибке данного типа должен быть выключен.

В директиве допустимы следующие двоичные побитовые операторы:

- `&` — двоичное И;
- `|` — двоичное ИЛИ.

```
error_reporting = E_ALL & ~E_NOTICE | E_WARNING
```

Если доступа к файлу php.ini нет (на виртуальном хостинге доступа точно не будет), то в сценарии можно вызвать функцию `error_reporting()`:

```
error_reporting(E_ALL & ~E_NOTICE);
```

В качестве параметра функции `error_reporting()` можно указать число:

```
31743 = E_ALL & ~E_NOTICE
32767 = E_ALL
32255 = E_ALL & ~E_NOTICE | E_WARNING
```

Предыдущий пример можно заменить на следующий код:

```
error_reporting(31743);
```

На виртуальном хостинге принято не выводить ошибки в Web-браузер, а записывать их в журнал ошибок error.log. В этом случае при возникновении фатальной ошибки пользователь увидит белый экран, а не сообщение об ошибке.

Отключить вывод ошибок в Web-браузер позволяет директива `display_errors` в файле php.ini:

```
display_errors = Off
```

А директива `log_errors` включает вывод сообщений об ошибках в журнал ошибок:

```
log_errors = On
```

Задать путь к файлу, в который будут выводиться ошибки, позволяет директива `error_log`:

```
error_log = 'C:/php5/err.txt'
```

Изменить эти директивы из скрипта можно с помощью функции `ini_set()`:

```
error_reporting(E_ALL);
ini_set('display_errors', 'Off');
ini_set('error_log', 'err.txt');
ini_set('log_errors', 'On');
$file = fopen('file.txt', 'r');
```

5.22.5. Инструкция *or die()*

Для обработки критических для всей программы ошибок можно использовать инструкцию `or die()`. В круглых скобках может быть указано сообщение об ошибке или функция, которая будет вызвана при возникновении ошибки. После вывода сообщения или вызова функции выполнение скрипта прекратится:

```
@$file = fopen("file.txt", "r") or die("Ошибка");
```

или

```
@$file = fopen("file.txt", "r") or die(err_msg());
function err_msg() {
   echo "Ошибка";
}
```

5.23. Переменные окружения

Создадим сценарий, состоящий всего из трех строк:

```
<?php
$var = 10;
?>
```

А теперь вопрос. Сколько переменных доступно сценарию? Думаете, одна `$var`? Давайте перепишем нашу программу и добавим одну строчку:

```
<?php
$var = 10;
echo $_SERVER['DOCUMENT_ROOT'];
?>
```

В результате работы скрипта в окне Web-браузера отобразится следующая строка:

```
C:/Apache2/htdocs
```

Откуда же взялась переменная `$_SERVER['DOCUMENT_ROOT']`? Ведь мы ее не создавали! Ответ на этот вопрос достаточно прост — эта переменная была автоматически создана интерпретатором. Такая переменная называется *переменной окружения*.

5.23.1. Суперглобальные массивы

Рассмотренная нами ранее переменная окружения `$_SERVER['DOCUMENT_ROOT']` представляет собой элемент массива `$_SERVER`. Это весьма примечательный массив — он доступен не только в сценарии, но и внутри функций, из-за чего и носит название *суперглобального*.

Перечислим суперглобальные массивы:

- `$_SERVER` — массив переменных среды сервера;
- `$_FILES` — массив переменных, определяющих отправленные через форму файлы;
- `$_POST` — массив переменных, переданных посредством метода `POST`;
- `$_GET` — массив переменных, переданных посредством метода `GET`;
- `$_COOKIE` — массив cookies-переменных;
- `$_ENV` — массив переменных, определяющих конфигурацию среды;
- `$_REQUEST` — массив всех переменных, вводимых пользователем. Этот массив зависит от значения директивы `request_order`.

5.23.2. Часто используемые переменные окружения

Рассмотрим наиболее часто используемые переменные окружения:

- `$_SERVER['DOCUMENT_ROOT']` — нуть к корневому каталогу сервера;
- `$_SERVER['REMOTE_ADDR']` — IP-адрес клиента, запрашивающего ресурс;

- `$_SERVER['REMOTE_USER']` — имя пользователя, прошедшего аутентификацию;
- `$_SERVER['QUERY_STRING']` — строка переданных серверу параметров;
- `$_SERVER['HTTP_USER_AGENT']` — название и версия Web-браузера клиента;
- `$_SERVER['HTTP_REFERER']` — URL-адрес, с которого пользователь перешел на наш сайт;
- `$_SERVER['REQUEST_METHOD']` — метод передачи информации (`GET` или `POST`).

Предположим, что пользователь заполнил форму с одним текстовым полем, имеющим имя `text1` (`name="text1"`). При передаче данных методом `GET` сервер сформирует следующие переменные:

```
$_GET['text1']
$_REQUEST['text1']
```

Если передача формы осуществлялась методом `POST`, то сервер сформирует другие переменные:

```
$_POST['text1']
$_REQUEST['text1']
```

Мы можем извлечь ее значение и присвоить обычной переменной PHP:

```
if (isset($_GET['text1'])) $text1 = $_GET['text1'];
else $text1 = '';
```

или

```
if (isset($_POST['text1'])) $text1 = $_POST['text1'];
else $text1 = '';
```

Остальные переменные окружения используются реже, а по названиям интуитивно понятно их предназначение. В дальнейшем мы еще не раз будем возвращаться к переменным окружения.

5.24. Заголовки HTTP

Заголовки HTTP предназначены для передачи некоторых дополнительных сведений, например, при запросе файла Web-браузером дополнительно указываются предпочитаемые MIME-типы, поддерживаемые языки и кодировки, информация о самом Web-браузере и т. д. Сервер в свою очередь при выдаче файла указывает MIME-тип файла, дату последней модификации файла, сведения о кодировке, языке и т. д.

Как вы уже знаете, данные формы могут быть отправлены либо методом `GET`, либо методом `POST`. При методе `GET` данные формы пересылаются путем их добавления к URL-адресу после знака "?". При методе `POST` данные передаются после всех HTTP-заголовков. Рассмотрим диалог Web-браузера и Web-сервера более подробно.

Предположим, есть форма:

```
<form action="test.php" method="GET">
<input type="text" name="text1">
<input type="submit" value="Отправить">
</form>
```

При заполнении текстового поля и нажатии кнопки **Отправпть** Web-браузер посылает следующий запрос:

```
GET /test.php?text1=Teskt+v+pole HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:33.0) ↳
Gecko/20100101 Firefox/33.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ru
Accept-Encoding: gzip, deflate
Referer: http://localhost/test.php
Connection: keep-alive
```

Обратите внимание на первую строку запроса. В ней первое слово обозначает метод передачи данных. В нашем случае это метод `GET`. Далее указывается строка запроса:

```
/test.php?text1=Tekst+v+pole
```

Здесь указывается путь от корня сайта к файлу-обработчику (test.php). После знака вопроса передается имя поля и его значение (`text1=Tekst+v+pole`). За строкой запроса следует название протокола (`HTTP/1.1`). Доменное имя Web-сайта передается в заголовке `Host` без указания протокола.

Кроме того, в запросе дополнительно указываются предпочитаемые MIME-типы (заголовок `Accept`), поддерживаемые языки (заголовок `Accept-Language`), методы сжатия (заголовок `Accept-Encoding`), информация о самом Web-браузере (заголовок `User-Agent`) и т. д.

Если изменить метод передачи с `GET` на `POST`, то запрос будет другим:

```
POST /test.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:33.0) ↳
Gecko/20100101 Firefox/33.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ru
Accept-Encoding: gzip, deflate
Referer: http://localhost/test.php
Connection: keep-alive
Content-Length: 18
Content-Type: application/x-www-form-urlencoded

text1=Tekst v pole
```

В первой строке указывается метод передачи (POST), путь к файлу-обработчику (test.php) от корня сервера и название протокола (HTTP/1.1). Сами данные формы передаются после всех заголовков. Обратите внимание: данные формы от заголовков отделяет пустая строка. Длина переданных данных указывается в заголовке Content-Length, а метод их кодирования — в заголовке Content-Type.

На этот запрос Web-сервер посылает следующий ответ:

```
HTTP/1.1 200 OK
Date: Thu, 27 Nov 2014 10:36:12 GMT
Server: Apache/2.4.10 (Win32) OpenSSL/1.0.1h PHP/5.4.35
X-Powered-By: PHP/5.4.35
Content-Length: 348
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
Content-Language: ru
```

В первой строке ответа сервера указывается название протокола (HTTP/1.1), а затем статус ответа (200) и его текстовое описание (OK). Статус 200 указывает, что запрос был успешно обработан. Перечислим основные коды статуса:

- 200 — запрос успешно обработан;
- 301 и 302 — перенаправление на другую страницу;
- 304 — с момента последнего запроса файл не изменялся;
- 401 — пользователь не авторизован;
- 403 — нет доступа. При отсутствии индексного файла в каталоге и отключенной опции Indexes директивы Options генерируется именно этот код;
- 404 — ресурс не найден;
- 500 — внутренняя ошибка сервера.

В заголовке Content-Type указываются MIME-тип (text/html) и кодировка передаваемых данных (utf-8). С помощью заголовка Content-Length указывается длина передаваемых данных. Сами данные передаются после всех заголовков. Данные от заголовков отделяет пустая строка.

Чтобы увидеть диалог Web-браузера с сервером, можно, например, воспользоваться модулем Firebug для Firefox *(см. разд. 3.14.5)*. Для этого на вкладке **Сеть** следует щелкнуть мышью на строке запроса. Результат можно увидеть на рис. 5.1.

5.24.1. Основные заголовки

Перечислим основные заголовки:

- Accept — MIME-типы, поддерживаемые Web-браузером:

```
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png,
image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1
```

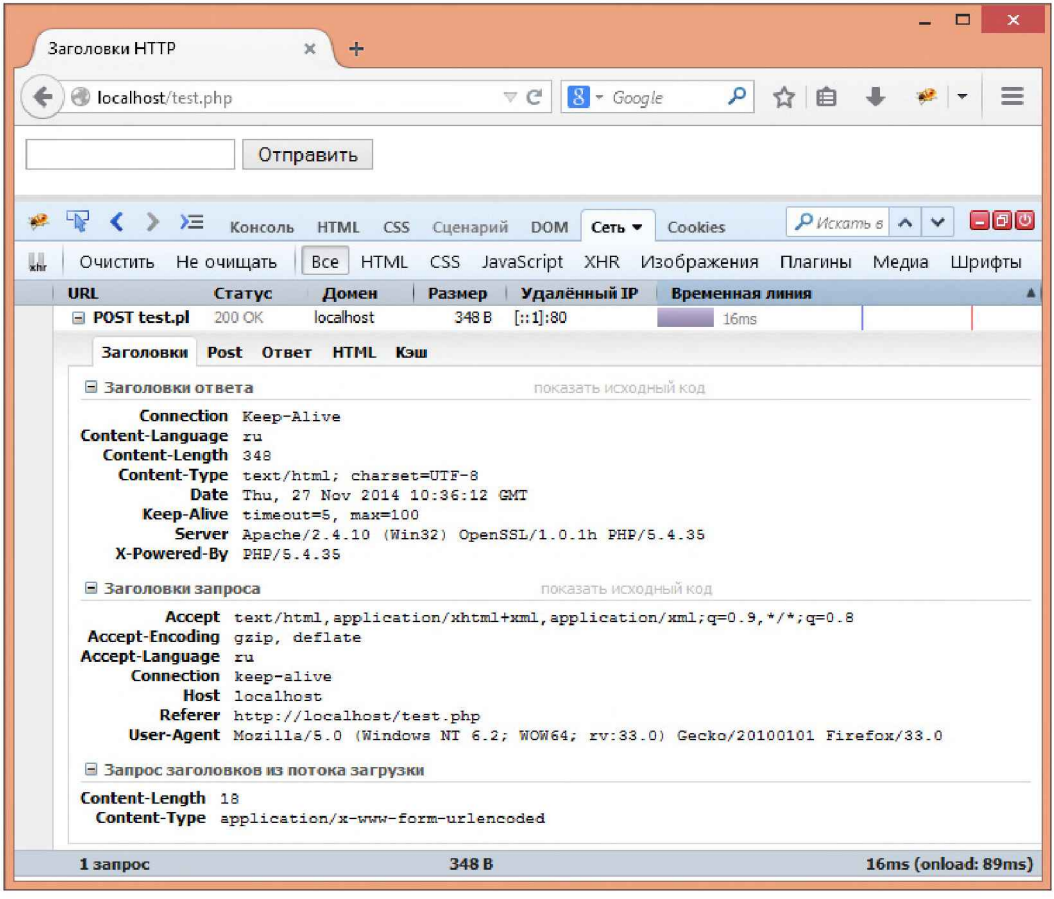


Рис. 5.1. Просмотр заголовков в модуле Firebug

- `Accept-Language` — список поддерживаемых Web-браузером языков в порядке предпочтения:

```
Accept-Language: ru-RU,ru;q=0.9,en;q=0.8
```

- `Accept-Charset` — список поддерживаемых Web-браузером кодировок:

```
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
```

- `Accept-Encoding` — список поддерживаемых Web-браузером методов сжатия:

```
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
```

- `Content-Type` — тип передаваемых данных:

```
Content-Type: text/html; charset=UTF-8
```

- `Content-Length` — длина передаваемых данных при методе `POST` и длина ответа сервера:

```
Content-Length: 12
```

- `Cookie` — информация об установленных cookies;

- `GET` — заголовок запроса при передаче данных методом `GET`;
- `POST` — заголовок запроса при передаче данных методом `POST`;
- `Host` — интернет-адрес хоста;
- `Last-Modified` — дата последней модификации файла:

```
Last-Modified: Thu, 27 Nov 2014 10:36:12 GMT
```

- `Location` — перенаправление: при указании этого заголовка Web-браузер обязан перейти по указанному URL-адресу:

```
Location: firm.php
```

- `Pragma` — заголовок, запрещающий кэширование документа:

```
Pragma: no-cache
```

- `Referer` — содержит URL-адрес, с которого пользователь перешел на наш сайт;
- `Server` — содержит название и версию программного обеспечения сервера и платформы PHP:

```
Server: Apache/2.4.10 (Win32) OpenSSL/1.0.1h PHP/5.4.35
```

- `User-Agent` — содержит информацию об используемом Web-браузере:

```
User-Agent: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:33.0) ↳
Gecko/20100101 Firefox/33.0
```

Некоторые из этих заголовков Web-браузер посылает серверу (например, `Referer` и `User-Agent`), другие используются в ответе сервера (например, `Pragma` или `Server`), третьи могут пересылаться в обоих направлениях (скажем, `Content-Length`).

5.24.2. Функции для работы с заголовками. Перенаправление клиента на другой URL-адрес. Запрет кэширования страниц. Реализация ссылки *Скачать*. Просмотр заголовков, отправляемых сервером

Функция `header()` позволяет добавить заголовок. Формат функции:

```
header(<Заголовок>);
```

Например, чтобы перенаправить клиента на URL **http://www.rambler.ru/**, нужно написать следующий код:

```
header("Location: http://www.rambler.ru/");
exit();
```

Чтобы запретить кэширование документа, нужно послать сразу несколько заголовков:

```
header("Expires: Thu, 01 Jan 2015 10:49:15 GMT");
header("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
```

```
header("Cache-Control: no-store, no-cache, must-revalidate");
header("Pragma: no-cache");
```

> ***Внимание!***
>
> Так как функция `header()` устанавливает заголовки ответа сервера, которые посылаются до отсылки основного содержимого документа, то перед функцией не должно быть никаких операторов вывода. Даже пустая строка перед открывающим PHP-дескриптором (`<?php`) вызовет ошибку. Пустые строки внутри PHP-дескрипторов ошибку не генерируют, поскольку вывод информации осуществляется только с помощью операторов `echo` или `print`. Кроме того, при использовании кодировки UTF-8 файл должен быть в кодировке UTF-8 без BOM.

Очень часто на сайтах можно видеть две ссылки: **Открыть** и **Скачать**. Например, у нас есть файл в формате Word с именем filename.doc. Реализовать первую ссылку достаточно просто:

```
<a href="filename.doc">Открыть</a>
```

При переходе по ссылке файл будет открыт соответствующей программой без запроса сохранения файла. Во всяком случае так поступит Microsoft Internet Explorer.

Реализовать вторую ссылку позволяет установка соответствующих заголовков. Для этого нам понадобится промежуточный файл, например, save.php. В документе размещаем следующую ссылку:

```
<a href="save.php">Скачать</a>
```

А в файле save.php пишем код, приведенный в листинге 5.37.

Листинг 5.37. Реализация ссылки "Скачать"

```
<?php
$path = 'filename.doc';
if (!file_exists($path)) {
   echo 'Файл не найден';
}
else {
   $size = filesize($path);
   header('Content-Type: application/msword');
   header('Content-Length: ' . $size);
   header('Content-Disposition: Attachment; FileName="' . $path . '"');
   readfile($path);
}
?>
```

При переходе по такой ссылке Web-браузер выведет диалоговое окно с запросом "Что делать с файлом?"

С помощью заголовков можно вывести данные, которые будут обработаны Microsoft Excel:

```
<?php
$text = "Артикул\tНазвание\tКоличество\n";
$text .= "001\tДискета\t100\n";
$text .= "002\tМонитор\t5\n";
$text .= "003\tHDD\t12\n";
header('Content-type: application/vnd.ms-excel');
$d = date('d_m_Y');
header('Content-Disposition: Attachment; FileName="price_' . $d . '.xls"');
echo $text;
exit();
?>
```

У этого способа есть недостаток. По умолчанию все ячейки таблицы имеют формат "Общий". В этом формате число `001` будет преобразовано в число `1`, пули в начале будут удалены. Задать формат "Текстовый" для определенной ячейки можно, представив таблицу в формате HTML (листинг 5.38).

Листинг 5.38. Представление данных в виде таблицы

```
<?php
header('Content-type: application/vnd.ms-excel');
$d = date('d_m_Y');
header('Content-Disposition: Attachment; FileName="price_' . $d . '.xls"');
?>
<html>
<head>
<title>Пример</title>
<meta http-equiv="content-type" content="text/html; charset=windows-1251">
<style type="text/css">
td {
    font-size:10.0pt;
    font-family:"Arial Cyr";
    mso-number-format:General;
    text-align:general;
    vertical-align:bottom;
    white-space:nowrap;
}
.txt { mso-number-format:"\@"; }
</style>
</head>
<body>
<table border="1" cellpadding="0" cellspacing="0">
<tr valign="bottom">
   <td class="txt" width="65"><b><center>Артикул</center></b></td>
   <td width="100"><b><center>Название</center></b></td>
   <td width="100"><b><center>Количество</center></b></td>
</tr>
```

```
<tr valign="bottom">
   <td class="txt">001</td>
   <td>Дискета</td>
   <td>100</td>
</tr>
<tr valign="bottom">
   <td class="txt">002</td>
   <td>Монитор</td>
   <td>5</td>
</tr>
<tr valign="bottom">
   <td class="txt">003</td>
   <td>HDD</td>
   <td>12</td>
</tr>
</table>
</body>
</html>
```

Обратите внимание — нули в артикуле остались. Чтобы узнать, какие значения необходимо указать, создайте таблицу в Excel, отформатируйте данные, а затем сохраните таблицу в формате HTML и отобразите исходный код.

Посмотреть заголовки, отправляемые сервером, позволяет функция `get_headers()`. Формат функции:

```
get_headers(<URL-адрес>);
```

В параметре `<URL-адрес>` должен быть указан абсолютный путь к файлу:

```
$url = "http://localhost/index.php";
echo "<pre>";
print_r(get_headers($url));
echo "</pre>";
```

Функция возвращает массив с заголовками, который будет отображен так:

```
Array
(
    [0] => HTTP/1.1 200 OK
    [1] => Date: Thu, 27 Nov 2014 10:50:30 GMT
    [2] => Apache/2.4.10 (Win32) OpenSSL/1.0.1h PHP/5.4.35
    [3] => X-Powered-By: PHP/5.4.35
    [4] => Connection: close
    [5] => Content-Type: text/html; charset=UTF-8
    [6] => Content-Language: ru
)
```

5.24.3. Работа с cookies. Создаем индивидуальный счетчик посещений

Web-браузеры позволяют сохранять небольшой объем информации в специальном текстовом файле на компьютере пользователя. Такая информация называется cookies. Возможность использования cookies можно отключить в настройках Web-браузера.

Для записи cookies предназначена функция `setcookie()`. Формат функции:

```
setcookie(<Имя>, <Значение>, [<Время жизни>], [<Путь>], [<Домен>],
      [<Способ передачи>]);
```

Большинство параметров необязательные. Если не указано `<Время жизни>` cookies, то оно будет удалено сразу после закрытия Web-браузера:

```
setcookie("var1", "12");
setcookie("var2", "15", time() + 86400);
```

Последнее выражение устанавливает cookies на один день.

> ***Внимание!***
>
> Так как функция `setcookie()` устанавливает заголовки ответа сервера, то перед функцией не должно быть никаких операторов вывода. Даже пустая строка перед открывающим PHP-дескриптором (`<?php`) вызовет ошибку. Кроме того, при использовании кодировки UTF-8 файл должен быть в кодировке UTF-8 без BOM.

Считывание cookies осуществляется следующим образом:

```
echo $_COOKIE['var1'];
echo $_COOKIE['var2'];
```

Все установленные cookies доступны через переменную окружения `$_SERVER['HTTP_COOKIE']`:

```
$cookies = $_SERVER['HTTP_COOKIE'];
```

Переменная `$cookies` будет содержать строку, в которой перечислены все установленные пары `имя=значение` через точку с запятой.

```
"var1=12; var2=15"
```

Для удаления cookies следует установить cookies с прошедшей датой.

В качестве примера работы с cookies создадим счетчик посещений (листинг 5.39).

Листинг 5.39. Счетчик посещений

```
<?php
if (!isset($_COOKIE['id_count'])) $id_count = 0;
else $id_count = $_COOKIE['id_count'];
$id_count++;
setcookie('id_count', $id_count, 0x6FFFFFFF);
echo 'Вы посетили ресурс ' . $id_count . ' раз';
?>
```

Если в cookies сохраняется строка, состоящая из русских букв, то ее следует закодировать, например, с помощью функции `urlencode()`. Раскодировать строку можно с помощью функции `urldecode()`.

Сохранить массив в cookies позволяет функция `serialize()`. Чтобы получить обратно массив, следует вызвать функцию `unserialize()`.

5.25. Работа с файлами и каталогами

Очень часто нужно сохранить какие-либо данные. Для этого существуют два способа: сохранение в файл и сохранение в базе данных. Первый способ подходит для сохранения информации небольшого объема. Если объем велик, то лучше (и удобнее) воспользоваться базой данных.

Файлы используются при создании гостевых книг, списков рассылки, ленты новостей, протоколирования различных ситуаций (например, ошибок) и во многих других случаях.

5.25.1. Основные понятия

Для чтения или записи файла нужно выполнить следующие действия:

1. Открыть файл.
2. Блокировать файл.
3. Считать или записать данные.
4. Снять блокировку.
5. Закрыть файл.

Учитывая, что для ускорения работы выполняется буферизация данных, можно опустить явное снятие блокировки файла. Все дело в том, что информация из буфера записывается в файл полностью только в момент закрытия файла. В период между снятием блокировки и закрытием файла другой процесс может успеть что-то записать в файл. При закрытии файла блокировка автоматически снимается.

5.25.2. Функции для работы с файлами. Создание файла, запись в файл, вывод содержимого файла в список

Рассмотрим основные функции для работы с файлами.

- `fopen(<Путь к файлу>, <Режим>[, <Путь поиска>[, <HTTP-заголовки>]])` — открывает файл и возвращает дескриптор (идентификатор). Параметр `<Режим>` может принимать следующие значения:
  - `r` — только чтение. После открытия файла указатель устанавливается на начало файла. Если файл не существует, функция `fopen()` вернет `false`;

- r+ — чтение и запись. После открытия файла указатель устанавливается на начало файла. Если файл не существует, функция fopen() вернет false;
- w — запись. Если файл не существует, то он будет создан. Если файл существует, то он будет перезаписан. После открытия файла указатель устанавливается на начало файла;
- w+ — чтение и запись. Если файл не существует, то он будет создан. Если файл существует, то он будет перезаписан. После открытия файла указатель устанавливается на начало файла;
- a — запись. После открытия файла указатель устанавливается на конец файла. Если файл не существует, функция fopen() вернет false;
- a+ — чтение и запись. После открытия файла указатель устанавливается на конец файла. Если файл не существует, то он будет создан. Содержимое файла не удаляется.

Кроме того, после параметра <Режим> может следовать модификатор:

- b — файл будет открыт в бинарном режиме (по умолчанию);
- t — файл будет открыт в текстовом режиме.

- flock(<Дескриптор>, <Режим>[, <Результат операции>]) — позволяет блокировать файл или снять блокировку. Параметр <Режим> может принимать следующие значения:
  - LOCK_SH или 1 — разделяемый доступ для чтения. Если другой процесс хочет записать что-либо в файл, то ему придется подождать снятия блокировки;
  - LOCK_EX или 2 — монопольный режим для записи. Файл не доступен для совместного использования;
  - LOCK_UN или 3 — снимает блокировку.
- fread(<Дескриптор>, <Длина в байтах>) — позволяет прочитать из файла строку указанной длины. Если функции не удалось прочесть заданное число байтов, она возвратит то, что удалось прочитать.
- fgets(<Дескриптор>[, <Длина в байтах>]) — позволяет считывать из файла по одной строке за раз. Считывание будет выполняться до тех пор, пока не встретится символ новой строки (\n), символ конца файла или из файла не будет прочитано указанное число байтов.
- file(<Путь к файлу>[, <Режим>[, <HTTP-заголовки>]]) — читает весь файл в массив, каждый элемент которого будет равен одной строке, прочитанной из файла. В параметре <Режим> могут быть указаны следующие значения:
  - FILE_USE_INCLUDE_PATH — поиск файла будет осуществлен в каталогах, перечисленных в директиве include_path;
  - FILE_IGNORE_NEW_LINES — не добавлять символ новой строки в конец элемента массива;
  - FILE_SKIP_EMPTY_LINES — игнорировать пустые строки.

- `readfile(<Путь к файлу>[, <true | false>[, <HTTP-заголовки>]])` — открывает файл и выводит все его содержимое в окно Web-браузера. Если во втором параметре указано значение `true`, то поиск файла будет осуществлен в каталогах, перечисленных в директиве `include_path`.
- `file_get_contents(<Путь к файлу>[, <Флаг>[, <HTTP-заголовки>[, <Начальная позиция>[, <Максимальная длина>]]]])` — возвращает содержимое файла в виде строки. В отличие от функции `readfile()` не выводит содержимое файла в окно Web-браузера. Если в параметре `<Флаг>` указано значение `FILE_USE_INCLUDE_PATH`, то поиск файла будет осуществлен в каталогах, перечисленных в директиве `include_path`.
- `fwrite(<Дескриптор>, <Строка>)` — записывает данные в файл.
- `fflush(<Дескриптор>)` — записывает изменения из буфера ввода/вывода на диск. Используется при обработке файлов большого объема, т. к. обычно запись на диск производится только при закрытии файла.
- `fclose(<Дескриптор>)` — закрывает файл.
- `file_put_contents()` — записывает данные в файл. Если файл не существует, то он будет создан. Если файл существует, то по умолчанию он будет перезаписан. Для записи в конец файла следует указать флаг `FILE_APPEND`. Вызов функции эквивалентен последовательности вызовов функций `fopen()`, `fwrite()` и `fclose()`. Формат функции:

  ```
  file_put_contents(<Путь к файлу>, <Данные>[, <Флаг>[, <HTTP-заголовки>]])
  ```

  В параметре `<Флаг>` могут быть указаны следующие значения (или их комбинация):

  - `FILE_USE_INCLUDE_PATH` — поиск файла будет осуществлен в каталогах, перечисленных в директиве `include_path`;
  - `FILE_APPEND` — если файл существует, то данные будут добавлены в конец содержимого файла;
  - `LOCK_EX` — монопольный режим для записи. Файл не доступен для совместного использования.

  Пример:

  ```
  // Перезапись файла
  file_put_contents('test.txt', 'Эти данные будут записаны');
  // Добавление в конец
  file_put_contents('test.txt', "\nА эти данные будут добавлены",
                    FILE_APPEND);
  ```

Для примера создадим файл file.txt и запишем в него нять адресов E-mail по одному на строчке (листинг 5.40).

Листинг 5.40. Создание файла и запись в него

```
<?php
@$file = fopen("file.txt", "a+") or die("Ошибка");
```

```
$mail = "mail1@site.ru\nmail2@site.ru\nmail3@site.ru\n";
$mail .= "mail4@site.ru\nmail5@site.ru";
flock($file, LOCK_EX);
fwrite($file, $mail);
flock($file, LOCK_UN);
fclose($file);
echo "Файл создан";
?>
```

Если в процессе создания файла возникнет ошибка, то она будет подавлена оператором `@`, а в окне Web-браузера будет выведено сообщение "Ошибка". При этом дальнейшая обработка файла будет остановлена.

Теперь добавим новую запись в конец файла (листинг 5.41).

Листинг 5.41. Добавление новой записи в конец файла

```
<?php
@$file = fopen("file.txt", "a+") or die("Ошибка");
$mail = "\nmail6@site.ru";
flock($file, LOCK_EX);
fwrite($file, $mail);
flock($file, LOCK_UN);
fclose($file);
echo "Операция произведена";
?>
```

А теперь выведем содержимое файла в список (листинг 5.42).

Листинг 5.42. Вывод содержимого файла в список

```
<?php
@$file = fopen("file.txt", "r");
if ($file) {
   flock($file, LOCK_SH);
   echo "<select>\n";
   while(!feof($file)) {
      echo '<option>', trim(fgets($file, 200)), '</option>';
   }
   echo "</select>\n";
   flock($file, LOCK_UN);
   fclose($file);
}
else {
   echo "Не удалось открыть файл";
}
?>
```

5.25.3. Перемещение внутри файла

Для каждого открытого файла существует особый указатель, помечающий текущую позицию в нем. Изменить позицию указателя внутри файла можно с помощью следующих функций:

- `rewind(<Дескриптор>)` — устанавливает указатель на начало файла;
- `ftell(<Дескриптор>)` — возвращает позицию указателя относительно начала файла;
- `feof(<Дескриптор>)` — возвращает `true`, если указатель находится в конце файла;
- `fseek(<Дескриптор>, <Смещение>[, <Позиция>])` — устанавливает указатель в позицию, имеющую смещение `<Смещение>` относительно позиции `<Позиция>`. Параметр `<Позиция>` может принимать следующие значения:
  - `SEEK_SET` — начало файла (по умолчанию);
  - `SEEK_CUR` — текущая позиция указателя;
  - `SEEK_END` — конец файла.

Установка указателя на конец файла продемонстрирована в программном коде, приведенном в листинге 5.43.

Листинг 5.43. Добавление E-mail с установкой указателя на конец файла

```
<?php
@$file = fopen("file.txt", "r+");
if ($file) {
   flock($file, LOCK_EX);
   fseek($file, 0, SEEK_END);
   fwrite($file, "\nmail7@site.ru");
   flock($file, LOCK_UN);
   fclose($file);
   echo "Строка записана";
}
else {
   echo "Не удалось открыть файл";
}
?>
```

5.25.4. Создание списка рассылки с возможностью добавления, изменения и удаления E-mail-адресов

В качестве примера рассмотрим создание списков рассылки. Создадим возможность добавления нового E-mail, удаления и переименования, а также выведем содержимое файла в поле `<textarea>`. Для этого создадим два файла: mail_script.php (листинг 5.44) и mail.php (листинг 5.45).

Листинг 5.44. Содержимое файла mail_script.php

```
<?php
// Проверка E-mail на корректность
function f_test_email($email) {
   $pattern = '/^([a-z0-9_.-]+)@([a-z0-9-]+\.)+[a-z]{2,6}$/is';
   return preg_match($pattern, $email);
}
// Проверка наличия E-mail. Возвращает индекс или false
function f_in_array($email, $mass) {
   for ($i=0, $c=count($mass); $i<$c; $i++) {
      if (strtolower($email) === strtolower($mass[$i]))
         return $i;
   }
   return false;
}
// Добавление E-mail
function f_add(&$txt) {
   if (f_test_email($txt)) {
      if (!file_exists('file.txt')) { // Если файл не существует
         file_put_contents('file.txt', $txt) or die('Ошибка');
         $txt = '';
         return '<span style="color: green">E-mail добавлен</span><br>';
      }
      $arr = file('file.txt',
                  FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
      if (count($arr) == 0) { // Если нет ни одного E-mail
         file_put_contents('file.txt', $txt) or die('Ошибка');
         $txt = '';
         return '<span style="color: green">E-mail добавлен</span><br>';
      }
      if (f_in_array($txt, $arr) === false) {
         file_put_contents('file.txt', "\n" . $txt, FILE_APPEND)
            or die('Ошибка');
         $txt = '';
         return '<span style="color: green">E-mail добавлен</span><br>';
      }
      else {
         $msg = '<span style="color: red">E-mail был добавлен ';
         $msg .= "ранее</span><br>\n";
         return $msg;
      }
   }
   else {
      $msg = '<span style="color: red">E-mail не соответствует ';
      $msg .= "шаблону</span><br>\n";
```

```
      return $msg;
   }
}
// Удаление E-mail
function f_delete(&$del) {
   if (!file_exists('file.txt')) { // Если файл не существует
      $msg = '<span style="color: red">Файл не существует';
      $msg .= "</span><br>\n";
      return $msg;
   }
   if (f_test_email($del)) {
      $arr = file('file.txt',
                  FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
      $index = f_in_array($del, $arr);
      if ($index !== false) {
         $arr[$index] = '';
         $str = implode("\n", $arr);
         $str = trim(str_replace("\n\n", "\n", $str));
         if (file_put_contents('file.txt', $str) === false)
            die('Ошибка');
         $del = '';
         return '<span style="color: green">E-mail удален</span><br>';
      }
      else {
         return '<span style="color: red">E-mail не найден</span><br>';
      }
   }
   else {
      $msg = '<span style="color: red">E-mail не соответствует ';
      $msg .= "шаблону</span><br>\n";
      return $msg;
   }
}
// Изменение E-mail
function f_update(&$s, &$na) {
   if (!file_exists('file.txt')) { // Если файл не существует
      $msg = '<span style="color: red">Файл не существует</span><br>';
      return $msg;
   }
   if (f_test_email($s) && f_test_email($na)) {
      $arr = file('file.txt',
                  FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
      $index = f_in_array($s, $arr);
      if ($index !== false) {
         if (f_in_array($na, $arr) === false) {
            $arr[$index] = $na;
            $str = implode("\n", $arr);
```

```
            file_put_contents('file.txt', $str) or die('Ошибка');
            $s = $na = '';
            $msg = '<span style="color: green">E-mail ';
            $msg .= 'изменен</span><br>';
            return $msg;
         }
         else {
            $msg = '<span style="color: red">Добавляемый E-mail ';
            $msg .= 'зарегистрирован ранее</span><br>';
            return $msg;
         }
      }
      else {
         return '<span style="color: red">E-mail не найден</span><br>';
      }
   }
   else {
      $msg = '<span style="color: red">E-mail не соответствует ';
      $msg .= 'шаблону</span><br>';
      return $msg;
   }
}
//  Вывод содержимого файла
function f_print() {
   echo '<textarea cols="25" rows="15">';
   if (file_exists('file.txt')) readfile('file.txt');
   echo '</textarea><br>';
}
?>
```

Листинг 5.45. Содержимое файла mail.php

```
<?php
require_once('mail_script.php');
if (isset($_GET['add'])) {
   $add = $_GET['add'];
   echo f_add($add);
}
else $add = '';
?>
<!-- Выводим форму Добавить -->
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="text" name="add" value="<?php echo $add; ?>">
<input type="submit" value="Добавить">
</form>
<?php
```

```
if (isset($_GET['del'])) {
   $del = $_GET['del'];
   echo f_delete($del);
}
else $del = '';
?>
<!-- Выводим форму Удалить -->
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="text" name="del" value="<?php echo $del; ?>">
<input type="submit" value="Удалить">
</form>
<?php
if (isset($_GET['s']) && isset($_GET['na'])) {
   $s = $_GET['s'];
   $na = $_GET['na'];
   echo f_update($s, $na);
}
else $s = $na = '';
?>
<!-- Выводим форму Изменить -->
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
Старый E-mail<br>
<input type="text" name="s" value="<?php echo $s; ?>"><br>
Новый E-mail<br>
<input type="text" name="na" value="<?php echo $na; ?>">
<input type="submit" value="Изменить">
</form>
<!-- Выводим содержимое файла -->
<?php f_print(); ?>
```

Откроем в Web-браузере файл mail.php. С помощью форм можно добавить новый E-mail, удалить или переименовать существующий. Причем добавить можно только новый E-mail; если будет введен уже существующий E-mail, то в Web-браузере отобразится соответствующее предупреждение. Кроме того, выполняется проверка на корректность введенного E-mail; если он не соответствует шаблону, то также отобразится сообщение. Заменить E-mail можно только на отсутствующий в файле E-mail. Таким образом, в файле будут записаны только уникальные E-mail-адреса.

Как разослать письма по E-mail-адресам из этого файла, мы рассмотрим при изучении отправки писем с сайта *(см. разд. 5.26)*.

5.25.5. Чтение CSV-файлов. Преобразование CSV-файла в HTML-таблицу

При работе с таблицами (например, в Excel) есть возможность сохранения таблицы в формате CSV. В этом формате каждая строка будет содержать значения ряда ячеек таблицы, разделенных точкой с запятой.

Например, таблица

1	2	3	4
5	6	7	8
9	10	11	12

при сохранении в формате CSV будет выглядеть следующим образом:

```
1;2;3;4
5;6;7;8
9;10;11;12
```

Для чтения CSV-файлов предусмотрена функция `fgetcsv()`. Формат функции:

```
fgetcsv(<Дескриптор>, [<Длина в байтах>], [<Разделитель>],
      [<Ограничитель>]);
```

Если `<Разделитель>` не указан, то по умолчанию используется , (запятая). А если не указан `<Ограничитель>`, то по умолчанию используется символ " (кавычка).

Функция `fgetcsv()` считывает из файла одну строку при каждом вызове. Считывание будет выполняться до тех пор, пока не встретится символ новой строки (`\n`), символ конца файла или из файла не будет прочитано указанное число байтов. Строка будет разбита по разделителю `<Разделитель>` и помещена в возвращаемый массив.

Если какая-либо ячейка содержит символ разделителя, то все содержимое ячейки обычно заключается в кавычки. Если используется другой символ, то он должен быть указан в параметре `<Ограничитель>`.

При сохранении в формате CSV таблица

1	2	3	4
5	6	7	8
9	10	11	12;15

будет выглядеть так:

```
1;2;3;4
5;6;7;8
9;10;11;"12;15"
```

Чтобы преобразовать CSV-файл в HTML-таблицу, можно воспользоваться кодом, приведенным в листинге 5.46.

Листинг 5.46. Преобразование CSV-файла в HTML-таблицу

```
<?php
@$file = fopen('filecsv.csv', 'r') or die('Ошибка');
flock($file, 1);
```

```
echo '<table cellspacing="0" cellpadding="5" border="1" width="200">';
echo "\n";
while(!feof($file)) {
   $Mass = fgetcsv($file, 1024, ';');
   $j = count($Mass);
   if ($j != 1) {
      echo '<tr align="center">' . "\n";
      for ($k=0; $k<$j; $k++) {
         echo '<td width="25%">';
         echo $Mass[$k];
         echo "</td>\n";
      }
      echo "</tr>\n";
   }
}
echo '</table>';
flock($file, 3); // 3 == LOCK_UN
fclose($file);
?>
```

5.25.6. Права доступа в операционной системе UNIX

Большинство хостинговых площадок используют операционную систему семейства UNIX. В этой ОС для каждого объекта (файла или каталога) назначаются права доступа для каждой разновидности пользователей — владельца, группы и прочих. Рассмотрим эту важную тему подробнее.

Для файла или каталога могут быть назначены следующие права доступа:

- чтение;
- запись;
- выполнение.

Права доступа обозначаются буквами:

- `r` — файл можно читать, а содержимое каталога можно просматривать;
- `w` — файл можно модифицировать, удалять и переименовывать, а в каталоге можно создавать или удалять файлы. Каталог можно переименовать или удалить;
- `x` — файл можно выполиять, а в каталоге можно выполнять операции над файлами, в том числе искать в нем файлы.

Права доступа к файлу определяются записью типа:

```
-rw-r--r--
```

Первый символ означает, что это файл, и не задает никаких прав доступа. Далее три символа (`rw-`) задают права доступа для владельца (чтение и запись). Символ `-` означает, что права доступа на выполнение нет. Следующие три символа задают

права доступа для группы (r--) — только чтение. Последние три символа (r--) задают права для всех остальных пользователей (только чтение).

Права доступа к каталогу определяются такой строкой:

```
drwxr-xr-x
```

Первая буква (d) означает, что это каталог. Владелец может выполнять в каталоге любые действия (rwx), а группа и все остальные пользователи — только читать и выполнять поиск (r-x). Для того чтобы каталог можно было просматривать, должны быть установлены права на выполнение (x).

Кроме того, права доступа могут обозначаться числом. Такие числа называются *маской прав доступа*. Число состоит из трех цифр от 0 до 7. Первая цифра задает права для владельца, вторая — для группы, а третья — для всех остальных пользователей. Например, права доступа -rw-r--r-- соответствуют числу 644.

Сопоставим числам, входящим в маску прав доступа, двоичную и буквенную записи (табл. 5.3).

Таблица 5.3. *Права доступа в разных записях*

Восьмеричная цифра	Двоичная запись	Буквенная запись
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Например, права доступа rw-r--r-- можно записать так: 110 100 100, что переводится в число 644. Таким образом, если право предоставлено, то в соответствующей позиции стоит 1, а если нет — то 0.

Для файлов, не являющихся cgi-программами (таких как html, shtml или php), права доступа могут быть установлены равными 644 (-rw-r--r--) (запись-чтение для владельца и только чтение для всех остальных).

Для файлов, являющихся cgi-программами (Perl-скрипты, скомпилированные программы на языке C и прочие), права доступа должны быть установлены в 755 (rwxr-xr-x) (исполнение-запись-чтение для владельца и чтение-исполнение для всех остальных).

Если Web-сервер запускает php-скрипты от имени владельца, то для записи данных в файл вполне достаточно поставить на этот файл права доступа 600 (rw-------). Если, конечно, файл не предназначен для чтения всеми пользователями.

Права доступа на каталоги рекомендуется устанавливать в 755 (`rwxr-xr-x`).

Чтобы изменить права доступа из скрипта, необходимо воспользоваться функцией `chmod()`. Формат функции:

```
chmod (<Путь к файлу>, <Права доступа>);
```

Права доступа задаются в виде числа, перед которым следует указать `0` (это соответствует восьмеричной записи числа):

```
chmod ($path, 0644);
```

Определить права доступа можно с помощью следующих функций:

- `is_readable(<Путь к файлу>)` — возвращает `true`, если файл доступен для чтения;
- `is_writable(<Путь к файлу>)` — возвращает `true`, если файл доступен для записи;
- `is_executable(<Путь к файлу>)` — возвращает `true`, если файл является выполняемым.

5.25.7. Функции для манипулирования файлами

- `copy(<Копируемый файл>, <Куда копируем>)` — позволяет скопировать файл. Если файл существует, то он будет перезаписан. Функция возвращает `true`, если файл удачно скопирован:

  ```
  if (@copy("file.csv", "file2.csv")) echo "Файл скопирован";
  ```

- `rename(<Старое имя>, <Новое имя>)` — переименовывает файл. Если новое имя файла уже существует, то функция вернет `false`. Если файл переименован, то функция вернет `true`:

  ```
  if (@rename("file.csv", "file3.csv")) echo "Файл переименован";
  ```

- `unlink(<Путь к файлу>)` — позволяет удалить файл. Функция вернет `true`, если файл был удален:

  ```
  if (@unlink("file3.csv")) echo "Файл удален";
  ```

- `file_exists(<Путь к файлу>)` — проверяет наличие файла. Значением функции будет `true`, если файл найден:

  ```
  if (file_exists("file2.csv")) echo "Файл существует";
  ```

- `basename(<Путь к файлу>)` — возвращает имя файла без нути к нему:

  ```
  echo basename("C:/Apache2/htdocs/file2.csv");
  // Выведет: file2.csv
  ```

- `dirname(<Путь к файлу>)` — возвращает нуть к каталогу:

  ```
  echo dirname("C:/Apache2/htdocs/file2.csv");
  // Выведет: C:/Apache2/htdocs
  ```

- `realpath(<Относительный путь к файлу>)` — преобразует относительный путь к файлу в абсолютный:

```
echo realpath("file2.csv");
// Выведет: C:\Apache2\htdocs\file2.csv
```

- `filesize(<Путь к файлу>)` — возвращает размер файла:

```
echo filesize("file2.csv");
```

- `fileatime(<Путь к файлу>)` — служит для определения времени последнего доступа к файлу:

```
$date = date("Дата d-m-Y", fileatime("index.php"));
echo $date;
// Выведет: Дата 18-06-2008
```

- `filectime(<Путь к файлу>)` — позволяет узнать время создания файла:

```
$date = date("Дата d-m-Y", filectime("index.php"));
echo $date;
// Выведет: Дата 04-06-2008
```

- `filemtime(<Путь к файлу>)` — возвращает время последнего изменения файла:

```
$date = date("Дата d-m-Y", filemtime("index.php"));
echo $date;
// Выведет: Дата 20-06-2008
```

- `touch(<Путь к файлу>, [<Время>])` — устанавливает для файла время последнего изменения:

```
touch("index.php");
```

Если параметр `<Время>` не указан, то используется текущее время. Если файла нет, то он будет создан.

5.25.8. Загрузка файлов на сервер

Загрузка файлов на сервер осуществляется с помощью формы, у которой параметр `enctype` равен `multipart/form-data`. Создадим файл file_load.html с содержимым, приведенным в листинге 5.47.

Листинг 5.47. Содержимое файла file_load.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Загрузка файлов</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
 <h1>Загрузка файлов</h1>
 <form action="file.php" method="POST" enctype="multipart/form-data">
  <div>
```

```
    <input type="file" name="file_name" size="20">
    <input type="submit" value="Загрузить">
  </div>
 </form>
</body>
</html>
```

Далее создаем файл file.php и добавляем в него код, приведенный в листинге 5.48.

Листинг 5.48. Содержимое файла file.php

```
<?php
if (isset($_FILES['file_name'])) {
   if ($_FILES['file_name']['error'] == 0 &&
      $_FILES['file_name']['size'] > 0) {
      $path = "C:\\Apache2\\htdocs\\";
      $path .= basename($_FILES['file_name']['name']);
      if (@move_uploaded_file($_FILES['file_name']['tmp_name'], $path)) {
         echo 'Файл загружен';
      }
      else {
         echo 'Ошибка при загрузке';
      }
   }
   else echo 'Ошибка при загрузке';
}
?>
```

При выборе файла с помощью кнопки **Обзор** и нажатии кнопки **Загрузить** файл будет отправлен серверу. Например, отправляем файл banner.gif. Получив файл, сервер сохраняет его в каталоге для временных файлов и создает переменные окружения со следующими значениями:

```
$_FILES["file_name"]["name"] => banner.gif
$_FILES["file_name"]["type"] => image/gif
$_FILES["file_name"]["tmp_name"] => C:\php5\tmp\phpDB.tmp
$_FILES["file_name"]["error"] => 0
$_FILES["file_name"]["size"] => 14987
```

Значение `file_name` здесь может изменяться — это название поля выбора файла в HTML-форме, а остальные параметры неизменны, и соответствующие им элементы ассоциативного массива содержат следующие данные:

- `name` — первоначальное название файла;
- `type` — MIME-тип файла;
- `tmp_name` — нуть и название временного файла;
- `size` — размер файла;

- error — код ошибки. Может принимать следующие значения:
  - 0 — UPLOAD_ERR_OK — ошибок нет, файл был успешно загружен на сервер;
  - 1 — UPLOAD_ERR_INI_SIZE — размер принятого файла превысил максимально допустимую величину, которая задана директивой upload_max_filesize конфигурационного файла php.ini;
  - 2 — UPLOAD_ERR_FORM_SIZE — размер загружаемого файла превысил значение MAX_FILE_SIZE, указанное в HTML-форме;
  - 3 — UPLOAD_ERR_PARTIAL — загружаемый файл был получен только частично;
  - 4 – UPLOAD_ERR_NO_FILE — файл не загружен.

Итак, файл загружен в каталог временных файлов. Теперь необходимо проверить, не возникло ли проблем с загрузкой. Если все в порядке, то переменная окружения $_FILES["file_name"]["error"] будет содержать значение 0. Затем нужно скопировать файл из каталога временных файлов в нужный каталог. Если файл не скопировать из каталога временных файлов, то по завершению работы сценария он будет удален. Скопировать файл можно с помощью двух функций: copy() и move_uploaded_file().

С функцией copy() мы уже знакомы, она просто копирует файлы. Функция move_uploaded_file() имеет следующий формат:

```
move_uploaded_file(<Загруженный файл>, <Куда копируем>);
```

Она перемещает загруженный файл в новое место, первоначально проверяя, является ли файл загруженным на сервер (переданным по протоколу POST). Если файл действительно загружен на сервер, он будет перемещен в место, указанное во втором параметре. Если он не является загруженным файлом, никаких действий не предпринимается, и функция возвращает false. Если файл, указанный во втором параметре, уже существует, он будет перезаписан. Если файл был успешно перемещен, то функция возвратит true.

5.25.9. Функции для работы с каталогами. Создаем программу для просмотра всех доступных каталогов и файлов на диске

Для работы с каталогами предусмотрены следующие функции:

- mkdir(<Имя каталога>, <Права доступа>) — создает новый каталог с правами доступа, указанными во втором параметре. Права доступа указываются в виде трехзначного числа, перед которым указывается 0, например 0755;
- rmdir(<Имя каталога>) — удаляет пустой каталог. Если в каталоге есть файлы, то каталог удален не будет;
- getcwd() — возвращает текущий каталог;
- chdir(<Имя каталога>) — делает указанный каталог текущим;

- `opendir(<Имя каталога>)` — открывает каталог для чтения. Функция возвращает дескриптор, который указывается в других функциях;
- `readdir(<Дескриптор>)` — считывает следующее имя объекта (файла или подкаталога);
- `closedir(<Дескриптор>)` — закрывает каталог;
- `rewinddir(<Дескриптор>)` — перемещает указатель в начало каталога;
- `is_dir(<Объект>)` — возвращает `true`, если объект является каталогом;
- `is_file(<Объект>)` — возвращает `true`, если объект является файлом.

Прочитаем содержимое каталога C:\Apache2\htdocs и выведем содержимое каталога в окно Web-браузера. Каталоги и файлы выведем отдельно, а для файлов укажем размер, дату создания и дату изменения файла. Кроме того, добавим возможность перемещения по файловой системе с помощью гиперссылок и предусмотрим возможность использования русских букв в названиях каталогов и файлов. Для этого создадим два файла: dir_script.php (листинг 5.49) и dir.php (листинг 5.50).

Листинг 5.49. Содержимое файла dir_script.php

```
<?php
function f_url_new($path) {
   $Mass = explode('/', $path);
   if (count($Mass)>1) {
      array_pop($Mass); // Удаляем последний элемент
      return implode('/', $Mass);
   }
   else return '';
}
function f_read_dir($path, &$d, &$f) {
   $descr = @opendir($path); // Открываем каталог
   if ($descr) {
      chdir($path); // Делаем каталог текущим
      while ($obj = readdir($descr)) {
         if (is_dir($obj)) { // Если это каталог
            if ($obj != '.') {
               $d[] = $obj;
            }
         }
         if (is_file($obj)) { // Если это файл
            $size = filesize($obj);
            $cdate = date('d-m-Y', filectime($obj));
            $mdate = date('d-m-Y', filemtime($obj));
            $f[] = array($obj, $size, $cdate, $mdate);
         }
      }
```

```
      closedir($descr); // Закрываем каталог
   }
   else exit('Не удалось открыть каталог');
}
?>
```

Листинг 5.50. Содержимое файла dir.php

```
<?php
require_once('dir_script.php');
$dir = array();
$files = array();
// Задаем путь по умолчанию
if (!isset($_GET['path'])) $path = 'C:/Apache2/htdocs';
else $path = $_GET['path'];
if (strlen($path)==0) exit('Не задан путь');
// Получаем файлы и папки текущего каталога
f_read_dir($path, $dir, $files);
$path2 = f_url_new($path);
// Кодируем все спецсимволы
$path = urlencode($path);
$path2 = urlencode($path2);
// Выводим содержимое каталога
?>
<table cellspacing="0" cellpadding="5" border="0" width="100%">
<tr><td width="25%">
<h2 align="center">Каталоги</h2>
</td><td>
<h2 align="center">Файлы</h2>
</td></tr>
<tr><td valign="top">
<?php
for ($i=0, $c=count($dir); $i<$c; $i++) {
   if ($dir[$i] == '..') {
      echo '<a href="?path=' . $path2 . '">На уровень выше</a><br><br>';
   }
   else {
      echo '<a href="?path=' . $path . urlencode('/' . $dir[$i]);
      echo '">' . $dir[$i] . "</a><br>\n";
   }
}
?>
</td><td valign="top">
<table cellspacing="0" cellpadding="5" border="1" width="100%">
<tr align="center">
<td width="25%">Название файла</td>
<td width="25%">Размер файла</td>
```

```
<td width="25%">Дата создания файла</td>
<td width="25%">Дата последнего изменения</td>
</tr>
<?php
// Выводим названия файлов
for ($k=0, $c=count($files); $k<$c; $k++) {
   echo '<tr align="center">';
   echo '<td>' . $files[$k][0] . "</td>\n";
   echo '<td>' . $files[$k][1] . "</td>\n";
   echo '<td>' . $files[$k][2] . "</td>\n";
   echo '<td>' . $files[$k][3] . "</td>\n";
   echo "</tr>\n";
}
echo "</table>\n";
if (count($files)==0) {
   echo '<div style="text-align: center"><b>Нет файлов</b></div>';
}
?>
</td></tr></table>
```

Откроем в Web-браузере файл dir.php. В результате отобразится содержимое каталога C:\Apache2\htdocs. С помощью гиперссылок можно перемещаться между каталогами, отображая их содержимое, почти как в программе Проводник в Windows.

5.25.10. Получение информации из сети Интернет

Открыть для чтения можно не только локально сохраненный файл, но и файл, находящийся на другом сервере в Интернете. С помощью функций `fopen()`, `file()` и `file_get_contents()` файл можно получить как по протоколу HTTP, так и по протоколу FTP. Для этого достаточно указать соответствующий протокол в URL-адресе, переданном в качестве параметра `<Путь к файлу>` этим функциям:

```
http://www.site.ru/file.txt
ftp://www.site.ru/file.txt
```

Как вам уже известно, в логах сервера отображается программное обеспечение клиента, сделавшего запрос. С помощью директивы `user_agent` в файле php.ini можно задать свое название. Для этого вместо строки

```
; user_agent="PHP"
```

нужно написать другую, например:

```
user_agent="MySpider/1.0"
```

При обработке больших файлов может потребоваться больше времени, чем задано по умолчанию (30 секунд). Увеличить время работы сценария можно с помощью директивы `max_execution_time`:

```
max_execution_time = 120
```

Кроме того, следует учитывать, что с помощью директивы `allow_url_fopen` можно запретить открытие внешних файлов. Для получения информации из сети Интернет значение директивы должно быть равно `On`:

```
allow_url_fopen = On
```

Получить документ с помощью функции `fopen()` можно следующим образом:

```
$host = 'http://wwwadmin.ru/testrobots.php?var1=10&var2=15';
$content = '';
$header = "User-Agent: MySpider/1.0\r\n";
$header .= "Cookie: test=5\r\n";
$option = array(
   'http' => array('method' => 'GET', 'header' => $header)
);
$context = stream_context_create($option);
@$file = fopen($host, 'r', false, $context);
if ($file) {
   while(!feof($file)) {
      $content .= fgets($file, 1024);
   }
   fclose($file);
   echo '<b>Содержимое страницы:</b><br><br>';
   echo '<pre>' . htmlspecialchars($content) . '</pre>';
}
else {
   echo 'Не удалось открыть файл';
}
```

Если необходимо получить документ в виде массива или строки, то можно воспользоваться функциями `file()` и `file_get_contents()`:

```
$host = 'http://wwwadmin.ru/testrobots.php?var1=10&var2=15';
$header = "User-Agent: MySpider/1.0\r\n";
$header .= "Cookie: test=5\r\n";
$option = array(
   'http' => array('method' => 'GET', 'header'=> $header)
);
$context = stream_context_create($option);
$file1 = implode('', file($host, 0, $context));
if ($file1) {
   echo '<b>Содержимое страницы (file()):</b><br><br>';
   echo '<pre>' . htmlspecialchars($file1) . '</pre>';
}
$file2 = file_get_contents($host, 0, $context);
if ($file2) {
   echo '<b>Содержимое страницы (file_get_contents()):</b><br><br>';
   echo '<pre>' . htmlspecialchars($file2) . '</pre>';
}
```

Функция `fsockopen()` дает возможность получить не только содержимое документа, но и все заголовки ответа сервера. Эта функция очень универсальна и позволяет открыть соединение не только с портом 80, но и с любым другим. Например, можно передать сообщение почтовому серверу на 25-й порт. Формат функции:

```
fsockopen(<Хост>, <Порт>, [<Номер ошибки>], [<Сообщение об ошибке>],
      [<Тайм-аут>]);
```

Функция устанавливает сетевое соединение и возвращает его дескриптор. Если соединение не установлено, то функция возвращает `false`. Получить номер и сообщение об ошибке можно с помощью необязательных параметров `<Номер ошибки>` и `<Сообщение об ошибке>`. В необязательном параметре `<Тайм-аут>` можно указать максимальное время, в течение которого производится попытка соединения (в секундах). Если параметр не указан, то значение будет взято из директивы `default_socket_timeout` файла php.ini:

```
default_socket_timeout = 60
```

После установки соединения необходимо передать заголовки запроса. Между собой заголовки должны разделяться с помощью комбинации символов `\r\n`. Заголовки должны отделяться от тела запроса с помощью комбинации `\r\n\r\n`.

Открытым соединением можно манипулировать как обычным файлом с помощью функций `fgets()`, `fwrite()`, `feof()` и др. Закрыть соединение позволяет функция `fclose()`.

Функция `stream_set_blocking()` дает возможность установить режим блокировки соединения. Формат функции:

```
stream_set_blocking(<Дескриптор соединения>, <Режим блокировки>);
```

Если режим равен `1`, то функции чтения будут ожидать полного завершения передачи данных. Если указать `0`, то блокировка снимается.

В качестве примера получим документ методом `GET` и выведем отдельно заголовки ответа сервера и содержимое документа (листинг 5.51).

Листинг 5.51. Чтение документа методом `GET`

```
<?php
$page = "/testrobots.php?var1=10&var2=15";
$port = "80";
$host = "wwwadmin.ru";
$header = "GET $page HTTP/1.1\r\n";
$header .= "Host: $host\r\n";
$header .= "User-Agent: MySpider/1.0\r\n";
$header .= "Accept: text/html, text/plain, application/xml\r\n";
$header .= "Accept-Language: ru, ru-RU\r\n";
$header .= "Accept-Charset: windows-1251\r\n";
$header .= "Accept-Encoding: identity\r\n";
$header .= "Connection: close\r\n";
$header .= "Cookie: test=5\r\n";
$header .= "\r\n";
```

```
@$fsock = fsockopen($host, $port, $err, $err_text, 30);
if ($fsock) {
   stream_set_blocking($fsock, 0);
   fwrite($fsock, $header);
   $s = 5;
   $content = "";
   $headers = "";
   $buffer = "";
   while(!feof($fsock)) {
      $buffer = fgets($fsock, 1024);
      if ($buffer == "\r\n") { $s = 10; }
      if ($s == 5) { $headers .= $buffer; }
      else { $content .= $buffer; }
   }
   fclose($fsock);
}
else {
   echo "Произошла ошибка " . $err . ": " . $err_text;
}
echo "<b>Заголовки ответа сервера:</b><br><br>";
echo "<pre>" . htmlspecialchars($headers) . "</pre>";
echo "<br><br><b>Содержимое страницы:</b><br><br>";
echo "<pre>" . htmlspecialchars($content) . "</pre>";
?>
```

Обратите внимание на строку

```
$host = "wwwadmin.ru";
```

Мы не указали протокол соединения, т. к. протокол `http://` по умолчанию закреплен за 80-м портом.

Если необходимо получить только заголовки ответа сервера, то вместо метода `GET` следует указать метод `HEAD`. Для этого строку

```
$header = "GET $page HTTP/1.1\r\n";
```

нужно заменить на

```
$header = "HEAD $page HTTP/1.1\r\n";
```

Данные можно передать не только методами `GET` и `HEAD`, но и методом `POST`, имитируя таким образом передачу данных формы. Рассмотрим это на примере (листинг 5.52).

Листинг 5.52. Имитация передачи данных методом `POST`

```
<?php
$page = "/testrobots.php";
$port = "80";
$host = "wwwadmin.ru";
```

```
$header = "POST $page HTTP/1.1\r\n";
$header .= "Host: $host\r\n";
$header .= "User-Agent: MySpider/1.0\r\n";
$header .= "Accept: text/html, text/plain, application/xml\r\n";
$header .= "Accept-Language: ru, ru-RU\r\n";
$header .= "Accept-Charset: windows-1251\r\n";
$header .= "Accept-Encoding: identity\r\n";
$header .= "Connection: close\r\n";
$header .= "Content-Type: application/x-www-form-urlencoded\r\n";
$header .= "Content-Length: 15\r\n";
$header .= "Cookie: test=5\r\n";
$header .= "\r\n";
@$fsock = fsockopen($host, $port, $err, $err_text, 30);
if ($fsock) {
   stream_set_blocking($fsock, 0);
   fwrite($fsock, $header);
   fwrite($fsock, "var1=10&var2=15");
   $s = 5;
   $content = "";
   $headers = "";
   $buffer = "";
   while(!feof($fsock)) {
      $buffer = fgets($fsock, 1024);
      if ($buffer == "\r\n") { $s = 10; }
      if ($s == 5) { $headers .= $buffer; }
      else { $content .= $buffer; }
   }
   fclose($fsock);
}
else {
   echo "Произошла ошибка " . $err . ": " . $err_text;
}
echo "<b>Заголовки ответа сервера:</b><br><br>";
echo "<pre>" . htmlspecialchars($headers) . "</pre>";
echo "<br><br><b>Содержимое страницы:</b><br><br>";
echo "<pre>" . htmlspecialchars($content) . "</pre>";
?>
```

В этом примере мы заменили метод передачи данных с `GET` на `POST` и добавили два заголовка:

```
$header .= "Content-Type: application/x-www-form-urlencoded\r\n";
$header .= "Content-Length: 15\r\n";
```

Первый заголовок имитирует передачу данных формы, а второй указывает длину данных, переданных методом `POST`. Сами данные мы передаем после всех заголовков:

```
fwrite($fsock, "var1=10&var2=15");
```

Использование библиотеки CURL

Вместо перечисленных функций для получения информации из сети Интернет можно воспользоваться библиотекой *CURL (Client URL Library)*. Чтобы использовать библиотеку CURL, необходимо в файле php.ini убрать символ комментария (`;`) перед строкой

```
;extension=php_curl.dll
```

В библиотеку CURL входят следующие функции:

- `curl_init([<URL-адрес>])` — создает новый сеанс и возвращает идентификатор;
- `curl_close(<Идентификатор>)` — завершает сеанс;
- `curl_setopt(<Идентификатор>, <Параметр>, <Значение>)` — устанавливает параметры сеанса.

  `<Параметр>` может равняться одной из следующих констант, определяющих смысл параметра `<Значение>`:

  - `CURLOPT_URL` — URL-адрес;
  - `CURLOPT_PORT` — номер порта;
  - `CURLOPT_USERAGENT` — значение HTTP-заголовка `User-Agent`;
  - `CURLOPT_RETURNTRANSFER` — если `true`, то функция `curl_exec()` будет возвращать результат, а не выводить его сразу в Web-браузер;
  - `CURLOPT_TIMEOUT` — максимальное время выполнения операции (в секундах);
  - `CURLOPT_CONNECTTIMEOUT` — максимальное время ожидания установки соединения (в секундах);
  - `CURLOPT_HEADER` — если `true`, то результат будет включать не только содержимое документа, но и заголовки ответа сервера;
  - `CURLOPT_NOBODY` — если `true`, то результат будет включать только заголовки ответа сервера;
  - `CURLOPT_POST` — если `true`, то запрос будет отправлен методом `POST`. Кроме того, будет добавлен HTTP-заголовок `Content-Type` со значением `application/x-www-form-urlencoded`. Этот заголовок обычно используется при передаче данных формы;
  - `CURLOPT_POSTFIELDS` — строка, содержащая данные для передачи методом `POST`;
  - `CURLOPT_REFERER` — значение HTTP-заголовка `Referer`;
  - `CURLOPT_COOKIE` — значение HTTP-заголовка `Cookie`;
  - `CURLOPT_FOLLOWLOCATION` — если `true`, то перенаправления будут обрабатываться автоматически;
  - `CURLOPT_HTTPHEADER` — массив с дополнительными HTTP-заголовками;

- CURLOPT_FILE — дескриптор файла, в который будет выведен результат операции;
- CURLOPT_WRITEHEADER — дескриптор файла, в который будут выведены полученные заголовки;
- CURLOPT_STDERR — дескриптор файла, в который будут выводиться сообщения об ошибках;
- CURLOPT_CRLF — если true, окончания строк в стиле UNIX будет автоматически преобразовываться в таковые стандарта Windows (CRLF);

- curl_exec(<Идентификатор>) — выполняет запрос;
- curl_getinfo(<Идентификатор>, [<Параметр>]) — возвращает информацию о последней операции. Если <Параметр> не указан, то функция возвращает ассоциативный массив. Если <Параметр> равен CURLINFO_HTTP_CODE, то функция возвращает последний полученный код HTTP. Если <Параметр> равен CURLINFO_CONTENT_TYPE, то функция возвращает содержимое полученного заголовка Content-Type или значение NULL, если заголовка нет в ответе сервера;
- curl_errno(<Идентификатор>) — возвращает код последней ошибки;
- curl_error(<Идентификатор>) — позволяет получить строку с описанием последней ошибки.

Пример запроса методом GET с получением содержимого документа и всех заголовков ответа сервера приведен в листинге 5.53.

Листинг 5.53. Получение документа методом GET с использованием библиотеки CURL

```
<?php
$url = "http://wwwadmin.ru/testrobots.php?var1=1&var2=5&var3=45";
@$curl=curl_init();
if (!curl_errno($curl)) {
   curl_setopt($curl, CURLOPT_URL, $url);
   curl_setopt($curl, CURLOPT_USERAGENT, "MySpider/1.0");
   curl_setopt($curl, CURLOPT_HEADER, 1);
   $headers = array(
      "Accept: text/html, text/plain, application/xml",
      "Accept-Language: ru, ru-RU",
      "Accept-Charset: windows-1251",
      "Accept-Encoding: identity",
      "Connection: close",
      "Cookie: test=5"
   );
   curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
   curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
   $content = curl_exec($curl);
```

```
   if (!curl_errno($curl)) {
      echo "<b>Код возврата:</b> ";
      echo curl_getinfo($curl, CURLINFO_HTTP_CODE);
      echo "<br><b>Заголовок Content-Type:</b> ";
      echo curl_getinfo($curl, CURLINFO_CONTENT_TYPE);
      echo "<br><br>";
      echo "<b>Содержимое страницы:</b><br><br>";
      echo "<pre>" . htmlspecialchars($content) . "</pre>";
   }
   else {
      echo "Произошла ошибка " . curl_errno($curl) . ": ";
      echo curl_error($curl);
   }
   curl_close($curl);
}
else {
   echo "Произошла ошибка " . curl_errno($curl) . ": ";
   echo curl_error($curl);
}
?>
```

В ряде случаев достаточно получить только заголовки ответа сервера (листинг 5.54).

Листинг 5.54. Получение заголовков ответа с использованием библиотеки CURL

```
<?php
$url = "http://wwwadmin.ru/testrobots.php?var1=1&var2=5&var3=45";
@$curl=curl_init();
if (!curl_errno($curl)) {
   curl_setopt($curl, CURLOPT_URL, $url);
   curl_setopt($curl, CURLOPT_USERAGENT, "MySpider/1.0");
   curl_setopt($curl, CURLOPT_HEADER, 1);
   curl_setopt($curl, CURLOPT_NOBODY, 1);
   $headers = array(
      "Accept: text/html, text/plain, application/xml",
      "Accept-Language: ru, ru-RU",
      "Accept-Charset: windows-1251",
      "Accept-Encoding: identity",
      "Connection: close",
      "Cookie: test=5"
   );
   curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
   curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
   $content = curl_exec($curl);
   if (!curl_errno($curl)) {
      echo "<b>Код возврата:</b> ";
      echo curl_getinfo($curl, CURLINFO_HTTP_CODE);
```

```
      echo "<br><b>Заголовок Content-Type:</b> ";
      echo curl_getinfo($curl, CURLINFO_CONTENT_TYPE);
      echo "<br><br>";
      echo "<b>Заголовки ответа сервера:</b><br><br>";
      echo "<pre>" . htmlspecialchars($content) . "</pre>";
   }
   else {
      echo "Произошла ошибка " . curl_errno($curl) . ": ";
      echo curl_error($curl);
   }
   curl_close($curl);
}
else {
   echo "Произошла ошибка " . curl_errno($curl) . ": ";
   echo curl_error($curl);
}
?>
```

Приведем также пример запроса методом `POST`, в котором мы не получаем заголовки ответа сервера, а только выводим содержимое страницы (листинг 5.55).

Листинг 5.55. Передача данных методом `POST` с использованием библиотеки CURL

```
<?php
$url = "http://wwwadmin.ru/testrobots.php";
@$curl=curl_init();
if (!curl_errno($curl)) {
   curl_setopt($curl, CURLOPT_URL, $url);
   curl_setopt($curl, CURLOPT_USERAGENT, "MySpider/1.0");
   curl_setopt($curl, CURLOPT_HEADER, 0);
   $headers = array(
      "Accept: text/html, text/plain, application/xml",
      "Accept-Language: ru, ru-RU",
      "Accept-Charset: windows-1251",
      "Accept-Encoding: identity",
      "Connection: close",
      "Cookie: test=5"
   );
   curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
   curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
   curl_setopt($curl, CURLOPT_POST, 1);
   curl_setopt($curl, CURLOPT_POSTFIELDS, "var1=1&var2=5&var3=45");
   $content = curl_exec($curl);
   if (!curl_errno($curl)) {
      echo "<b>Код возврата:</b> ";
      echo curl_getinfo($curl, CURLINFO_HTTP_CODE);
      echo "<br><b>Заголовок Content-Type:</b> ";
      echo curl_getinfo($curl, CURLINFO_CONTENT_TYPE);
```

```
      echo "<br><br>";
      echo "<b>Содержимое страницы:</b><br><br>";
      echo "<pre>" . htmlspecialchars($content) . "</pre>";
   }
   else {
      echo "Произошла ошибка " . curl_errno($curl) . ": ";
      echo curl_error($curl);
   }
   curl_close($curl);
}
else {
   echo "Произошла ошибка " . curl_errno($curl) . ": ";
   echo curl_error($curl);
}
?>
```

5.26. Отправка писем с сайта. Рассылка писем по E-mail-адресам из файла

Отправить письма с сайта позволяет функция `mail()`. Формат функции:

```
mail(<E-mail получателя>, <Тема>, <Сообщение>, [<Заголовки>]);
```

Функция возвращает `true`, если письмо отправлено.

В параметре `<Заголовки>` обычно указываются следующие заголовки:

- `From` — имя и обратный адрес отправителя:

  ```
  From: Nik <unicross@mail.ru>
  ```

- `Content-Type` — MIME-тип и кодовая таблица:

  ```
  Content-Type: text/html; charset=windows-1251
  ```

Заголовки должны быть разделены комбинацией символов `\r\n`. Если в тексте заголовков присутствуют русские буквы, то текст следует зашифровать с помощью метода base64 следующим образом:

```
=?<Кодировка>?B?<Зашифрованный текст>?=
```

Зашифровать текст с помощью метода base64 позволяет функция `base64_encode()`:

```
$tema = "Сообщение";
$tema = "=?windows-1251?B?" . base64_encode($tema) . "?=";
```

Зашифровать текст в кодировке UTF-8 позволяет функция `mb_encode_mimeheader()`. Формат функции:

```
mb_encode_mimeheader(<Строка>, [<Кодировка>], [<Метод кодирования>],
                     [<Символ переноса строк>]);
```

Если параметр `<Кодировка>` не указан, то берется значение, указанное в функции `mb_internal_encoding()`. Как показывает практика, указывать кодировку в функции

mb_internal_encoding() нужно обязательно. Параметр <Метод кодирования> может принимать значения "B" (Base64) или "Q" (Quoted-Printable). Если параметр не указан, то используется значение "B". Параметр <Символ переноса строк> задает символ для разделения строк. По умолчанию предполагается комбинация "\r\n".

Пример:

```
mb_internal_encoding('UTF-8');
$tema = 'Сообщение';
echo mb_encode_mimeheader($tema);
// Выведет: =?UTF-8?B?0KHQvtC+0LHRidC10L3QuNC1?=
```

Текст сообщения шифровать необязательно. А вот кодовую таблицу символов следует указать в заголовке Content-Type. Также следует учитывать, что длина одной строки не должна превышать 70 символов. Строки отделяются друг от друга символом перевода строки (\n).

Любое письмо может быть отправлено в виде обычного текста (листинг 5.56), а также в формате HTML (листинг 5.57). В первом случае указывается MIME-тип text/plain, а во втором — text/html.

В качестве примера отправим письмо с подтверждением регистрации.

Листинг 5.56. Пример отправки письма в виде обычного текста

```
<?php
$msg = "Добрый день!\n\n";
$msg .= "Вы успешно зарегистрированы.\n\n";
$msg .= "http://www.site.ru/\n";
$msg .= "support@site.ru";
$Ot = "Суппорт";
$Ot = "=?windows-1251?B?" . base64_encode($Ot) . "?=";
$header = "Content-Type: text/plain; charset=windows-1251\r\n";
$header .= "From: " . $Ot . " <support@site.ru>";
$tema = "Сообщение";
$tema = "=?windows-1251?B?" . base64_encode($tema) . "?=";
mail("vasya@mail.ru", $tema, $msg, $header);
?>
```

Отправленное письмо (с заголовками) будет выглядеть следующим образом:

```
To: vasya@mail.ru
Subject: =?windows-1251?B?0e7u4fnl7ejl?=
Content-Type: text/plain; charset=windows-1251
From: =?windows-1251?B?0fPv7+7w8g==?= <support@site.ru>

Добрый день!

Вы успешно зарегистрированы.

http://www.site.ru/
support@site.ru
```

Листинг 5.57. Пример отправки письма в формате HTML

```
<?php
$msg = "Добрый день!<br><br>\n";
$msg .= "Вы успешно зарегистрированы.<br><br>\n";
$msg .= "<a href=\"http://www.site.ru/\">http://www.site.ru/</a><br>\n";
$msg .= "support@site.ru";
$Ot = "Суппорт";
$Ot = "=?windows-1251?B?" . base64_encode($Ot) . "?=";
$header = "Content-Type: text/html; charset=windows-1251\r\n";
$header .= "From: " . $Ot . " <support@site.ru>";
$tema = "Сообщение";
$tema = "=?windows-1251?B?" . base64_encode($tema) . "?=";
mail("vasya@mail.ru", $tema, $msg, $header);
?>
```

При изучении работы с файлами мы создали файл file.txt со списком рассылки и механизм работы с ним *(см. разд. 5.25.4)*. Теперь рассмотрим возможность рассылки писем по E-mail-адресам из этого файла (листинг 5.58).

Листинг 5.58. Рассылка писем

```
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="hidden" name="send" value="1">
<input type="submit" value="Разослать">
</form>
<?php
if (isset($_GET["send"])) {
   $email = file('file.txt') or die('Файл не найден');
   $msg = "Добрый день!\n\n";
   $msg .= "Новости нашего сайта.\n\n";
   $msg .= "http://www.site.ru/\n";
   $msg .= "mail@site.ru";
   $header = "Content-Type: text/plain; charset=windows-1251\r\n";
   $header.="From: news <mail@site.ru>";
   $tema = "Новости сайта";
   $tema = "=?windows-1251?B?" . base64_encode($tema) . "?=";
   for ($i=0, $c=count($email); $i<$c; $i++) {
      $email[$i] = trim($email[$i]);
      if (preg_match('/^([a-z0-9_.-]+)@([a-z0-9-]+\.)+[a-z]{2,6}$/is',
         $email[$i])) {
         @mail($email[$i], $tema, $msg, $header);
      }
   }
   echo "Сообщения разосланы";
}
?>
```

При нажатии кнопки **Разослать** на все E-mail из файла будет отправлено письмо.

> ***Обратите внимание***
>
> На локальной машине письма отправлены не будут, т. к. мы не устанавливали программу отправки писем — sendmail. На сервере хостинг-провайдера данная программа практически всегда установлена и настроена. Правда, число одновременно отправленных писем часто ограничено.

При рассылке в письме обязательно должна быть предусмотрена возможность отписаться от рассылки. И запомните — рассылка спама в Интернете запрещена. Под спамом понимаются письма, не запрошенные явным образом получателем.

5.27. Аутентификация с помощью PHP. Создание Личного кабинета

В *разд 4.4.14* мы уже рассматривали аутентификацию посетителей при помощи файла конфигурации сервера Apache .htaccess. В PHP существует свой способ аутентификации посетителей на основе механизма сессий.

За механизм сессий в файле php.ini отвечают следующие директивы:

- `session.save_handler` — определяет место хранения данных сеанса:

```
session.save_handler = files
```

- `session.save_path` — задает путь к месту хранения данных сеанса. При установке мы изменили значение "/tmp" на "c:/php5/tmp" и создали папку tmp в каталоге c:/php5:

```
session.save_path = "c:/php5/tmp"
```

- `session.use_cookies` — включает возможность использования cookies для механизма сессий. Если указано значение 1, то использование разрешено, если 0 — то запрещено:

```
session.use_cookies = 1
```

- `session.name` — определяет имя сеанса:

```
session.name = PHPSESSID
```

- `session.auto_start` — задает возможность автоматического запуска механизма работы с сеансами. Значение 0 запрещает запуск:

```
session.auto_start = 0
```

- `session.cookie_path` — определяет путь для установки в cookies сеанса:

```
session.cookie_path = /
```

- `session.cache_expire` — устанавливает время жизни для кэшированных страниц сеанса в минутах:

```
session.cache_expire = 180
```

- `session.cookie_lifetime` — определяет время жизни cookies на машине пользователя. Значение 0 указывает на то, что cookies будет удалено сразу после закрытия окна Web-браузера:

```
session.cookie_lifetime = 0
```

- `session.use_trans_sid` — задает возможность присоединения `PHPSESSID` к URL-адресам. Если указано значение 1, то использование разрешено, если 0 — то запрещено:

```
session.use_trans_sid = 1
```

Работа с механизмом сессий осуществляется следующим образом:

1. Запускается сеанс.
2. Регистрируются переменные сеанса.
3. Используются переменные сеанса.
4. Удаляются переменные сеанса.
5. Сеанс закрывается.

Запустить сессию позволяет функция `session_start()`:

```
session_start();
```

Если запустить сессию удалось, она возвращает `true`.

При запуске сессии на компьютер пользователя устанавливается cookies с именем `PHPSESSID` и значением вида `db711b560810e7f90d67a4c8e6a873af`. А в папке c:/php5/tmp создается временный файл с именем `sess_db711b560810e7f90d67a4c8e6a873af`.

Если прием cookies отключен в настройках Web-браузера, то к любой ссылке будет добавлена следующая строка:

```
?PHPSESSID=db711b560810e7f90d67a4c8e6a873af
```

Если на странице имеется форма, то внутрь будет автоматически добавлено скрытое поле:

```
<input type="hidden" name="PHPSESSID"
value="db711b560810e7f90d67a4c8e6a873af" />
```

При первом запуске будут установлены cookies, а имя сессии будет добавлено в URL-адреса для всех внутренних ссылок. Если cookies разрешено использовать, то в дальнейшем добавление к URL будет прекращено.

Поменять имя сеанса можно с помощью функции `session_name()`:

```
session_name("Ivan");
session_start();
```

Теперь вместо имени `PHPSESSID` будет использоваться имя `Ivan`.

Зарегистрировать переменную внутри сеанса можно следующим образом:

```
$_SESSION["var1"] = 1;
```

Переменная сохраняется не в cookies пользователя, а в специальном файле на сервере. В cookies пользователя сохраняется только переменная с именем `PHPSESSID` и значением вида `db711b560810e7f90d67a4c8e6a873af`.

Проверить существование переменной можно с помощью функции `isset()`:

```
if (isset($_SESSION["var1"])) {
   echo "Переменная зарегистрирована";
}
```

Удалить переменную сеанса позволяет функция `unset()`:

```
unset($_SESSION["var1"]);
```

Удалить сразу все переменные сеанса позволяет функция `session_unset()`:

```
session_start(); // Запускаем сессию
session_unset(); // Удаляем все переменные
session_destroy(); // Удаляем идентификатор
```

После завершения сеанса сначала нужно удалить все переменные сеанса, а затем вызвать функцию `session_destroy()` для удаления идентификатора сеанса:

```
session_destroy();
```

Для примера создадим папку secure в папке C:\Apache2\htdocs. В этой папке создадим следующие файлы:

- index.php — содержит форму для ввода логина и пароля (листинг 5.59);
- secure.php — файл с информацией только для прошедших аутентификацию пользователей (листинг 5.60);
- exit.php — для завершения сеанса (листинг 5.61);
- data.php — для хранения логина и пароля (листинг 5.62).

Листинг 5.59. Содержимое файла index.php

```
<?php
require_once('data.php');
$err = '';
if (isset($_POST['login']) && isset($_POST['passw'])) {
   $_POST['passw'] = md5($_POST['passw']);
   if ($_POST['login']===$enter_login &&
      $_POST['passw']===$enter_passw) {
      session_start();
      $_SESSION['sess_login'] = $_POST['login'];
      $_SESSION['sess_pass'] = $_POST['passw'];
      header('Location: secure.php');
      exit();
   }
   else {
      $err = '<span style="color: red"><b>';
```

```
      $err .= 'Логин или пароль введены неправильно!';
      $err .= '</b></span><br>';
   }
}
?>
<form action="index.php" method="POST">
 <div align="center" style="padding: 250px 0 0 0">
  <table border="0" cellspacing="0" width="200">
   <caption><b>Вход в систему</b></caption>
   <tr><td align="right"><b>Логин:</b></td>
   <td><input type="text" name="login"></td></tr>
   <tr><td align="right"><b>Пароль:</b></td>
   <td><input type="password" name="passw"></td></tr>
   <tr>
   <td align="center" colspan="2">
    <input type="submit" value="Войти">
  </td></tr></table>
  <?php echo $err; ?>
 </div>
</form>
```

Листинг 5.60. Содержимое файла secure.php

```
<?php
session_start();
require_once('data.php');
if (isset($_SESSION['sess_login']) && isset($_SESSION['sess_pass'])) {
   if ($_SESSION['sess_login']===$enter_login &&
         $_SESSION['sess_pass']===$enter_passw) {
      echo "Информация для прошедших аутентификацию<br><br>\n";
      echo "<a href=\"exit.php\">Выйти из системы</a>\n";
   }
   else {
      header('Location: index.php');
      exit();
   }
}
else {
   header('Location: index.php');
   exit();
}
?>
```

Листинг 5.61. Содержимое файла exit.php

```
<?php
session_start();
```

```
session_unset(); // Удаляем все переменные
session_destroy();
header("Location: index.php");
?>
```

Листинг 5.62. Содержимое файла data.php

```
<?php
$enter_login = "login";
$enter_passw = "202cb962ac59075b964b07152d234b70";
?>
```

В данном примере только один логин (`login`) и пароль (`123`). В реальной практике для каждого пользователя создается свой логин и пароль. Учетные записи хранятся в файле или чаще всего в базе данных. Если используется обычный файл (как в нашем случае), то он должен содержать пароль в зашифрованном виде, а сам файл должен быть недоступен через Интернет. В нашем примере файл data.php должен быть расположен в папке C:\php5\includes, а не в C:\Apache2\htdocs\secure.

> ***Внимание!***
> Так как мы устанавливаем заголовки ответа сервера, то перед функцией `session_start()` не должно быть никаких операторов вывода. Даже пустая строка перед открывающим PHP-дескриптором (`<?php`) вызовет ошибку.

5.28. Работа с графикой

Библиотека GD позволяет работать со следующими форматами изображений:

- JPEG (Joint Photographic Experts Group);
- PNG (Portable Network Graphics);
- GIF (Graphics Interchange Format);
- WBMP (Wireless Bitmap).

5.28.1. Информация об установленной библиотеке GD

Чтобы использовать библиотеку GD, необходимо в файле php.ini убрать символ комментария (`;`) перед строкой

```
;extension=php_gd2.dll
```

Если библиотека установлена правильно, функция `gd_info()` возвращает информацию об установленной библиотеке GD в виде ассоциативного массива:

```
$Mass = gd_info();
foreach ($Mass as $key=>$val) {
   if ($val===true) {
      $val = 'Да';
   }
```

```
    if ($val===false) {
       $val = '<span style="color: red">Нет</span>';
    }
    echo "$key: $val <br>\n";
}
```

Этот PHP-код выведет текст, отображающийся в Web-браузере так:

```
GD Version: bundled (2.1.0 compatible)
FreeType Support: Да
FreeType Linkage: with freetype
T1Lib Support: Нет
GIF Read Support: Да
GIF Create Support: Да
JPEG Support: Да
PNG Support: Да
WBMP Support: Да
XPM Support: Да
XBM Support: Да
JIS-mapped Japanese Font Support: Нет
```

5.28.2. Получение информации об изображении

Несколько функций позволяют получить информацию об изображениях:

- `getimagesize()` — возвращает информацию об изображении в виде ассоциативного массива. В качестве примера возьмем стандартный баннер 468×60:

```
$Mass = getimagesize("banner.gif");
foreach ($Mass as $key=>$val) {
  echo "$key: $val <br>\n";
}
```

Такой пример выведет HTML-код, отображаемый Web-браузером примерно так:

```
0: 468
1: 60
2: 1
3: width="468" height="60"
bits: 6
channels: 3
mime: image/gif
```

Третий элемент возвращенного функцией массива (с индексом 2) содержит значение одной из следующих констант:

- 1 — `IMAGETYPE_GIF`;
- 2 — `IMAGETYPE_JPEG`;
- 3 — `IMAGETYPE_PNG`;
- 4 — `IMAGETYPE_SWF`;
- 5 — `IMAGETYPE_PSD`;

- 6 — IMAGETYPE_BMP;
- 7 — IMAGETYPE_TIFF_II;
- 8 — IMAGETYPE_TIFF_MM;
- 9 — IMAGETYPE_JPC;
- 10 — IMAGETYPE_JP2;
- 11 — IMAGETYPE_JPX;
- 12 — IMAGETYPE_JB2;
- 13 — IMAGETYPE_SWC;
- 14 — IMAGETYPE_IFF;
- 15 — IMAGETYPE_WBMP;
- 16 — IMAGETYPE_XBM;
- 17 — IMAGETYPE_ICO.

- imagesx() — возвращает ширину изображения, загруженного с помощью функций imagecreatefrompng(), imagecreatefromgif(), imagecreatefromjpeg(), imagecreatefromwbmp() или др.;
- imagesy() — возвращает высоту изображения, загруженного с помощью функций imagecreatefrompng(), imagecreatefromgif(), imagecreatefromjpeg(), imagecreatefromwbmp() или др.:

```
$img = imagecreatefromgif("banner.gif");
echo "Ширина " . imagesx($img) . "<br>";
echo "Высота " . imagesy($img) . "<br>";
// Выведет: Ширина 468<br>Высота 60<br>
```

Получить детальную информацию о JPEG- и TIFF-изображениях позволяет библиотека php_exif.dll. Для использования функций из этой библиотеки необходимо в файле php.ini убрать комментарий (;) перед строкой

```
;extension=php_exif.dll
```

После этого необходимо перенести строку в самый конец перечисления всех расширений.

> ***Обратите внимание***
>
> Если не перенести строку, то библиотека будет недоступна для использования. Кроме того, библиотека php_exif.dll требует подключения библиотеки php_mbstring.dll. А переносим мы строку именно из-за того, что библиотека php_exif.dll должна загружаться после библиотеки php_mbstring.dll.

Библиотека php_exif.dll предоставляет следующие функции:

- exif_imagetype(<Путь к файлу>) — позволяет определить формат файла. Функция возвращает false, если формат файла определить не удалось, или значение одной из констант IMAGETYPE_*XXX*, перечисленных в описании функции getimagesize().

Проверить формат можно так:

```
if (exif_imagetype('foto.jpg') == IMAGETYPE_JPEG) {
   echo 'Это фото в формате JPEG';
}
```

Или так:

```
if (exif_imagetype('foto.jpg') == 2) {
   echo 'Это фото в формате JPEG';
}
```

- `exif_read_data()` — позволяет вывести информацию из заголовков JPEG- и TIFF-файлов. Возвращает результат в виде ассоциативного массива. Формат функции:

```
exif_read_data(<Имя файла>, [<Список разделов>],
               [<Тип массива>], [<Вывод миниатюры>]);
```

В параметре `<Список разделов>` можно перечислить через запятую разделы, которые должны присутствовать в файле. Если указанный раздел отсутствует, то функция возвращает `false`. По умолчанию параметр имеет значение `NULL`.

Параметр `<Тип массива>` определяет, будет ли каждый раздел представлен в виде отдельного массива (значение `true`) или нет (значение `false`). По умолчанию параметр имеет значение `false`. Следует учитывать, что разделы `COMPUTED`, `THUMBNAIL` и `COMMENT` всегда представлены как отдельные массивы.

Параметр `<Вывод миниатюры>` определяет, будет ли загружена миниатюра (значение `true`) или только информация о ней (значение `false`). По умолчанию параметр имеет значение `false`.

Пример использования функции:

```
<?php
$arr = exif_read_data('foto.jpg');
if (!$arr) die('Информации нет');
echo '<b>Информация об изображении:</b><br>';
echo 'Название: ' . $arr['FileName'] . '<br>';
echo 'Размер: ';
echo number_format($arr['FileSize'], 0, '.', ' ');
echo '<br>';
echo 'Mime-тип: ' . $arr['MimeType'] . '<br>';
echo 'Ширина: ' . $arr['COMPUTED']['Width'] . '<br>';
echo 'Высота: ' . $arr['COMPUTED']['Height'] . '<br>';
echo 'Дата создания: ' . $arr['DateTimeOriginal'] . '<br>';
echo 'Выдержка: ' . $arr['ExposureTime'] . '<br>';
echo 'Чувствительность: ' . $arr['ISOSpeedRatings'] . '<br>';
echo '<br>';
echo '<b>Информация о фотоаппарате:</b><br>';
echo 'Производитель: ' . $arr['Make'] . '<br>';
echo 'Модель: ' . $arr['Model'] . '<br>';
?>
```

Выведет примерно:

```
Информация об изображении:
Название: foto.jpg
Размер: 5 578 604
Mime-тип: image/jpeg
Ширина: 3648
Высота: 2736
Дата создания: 2008:07:05 10:26:41
Выдержка: 1/320
Чувствительность: 80
Информация о фотоаппарате:
Производитель: Canon
Модель: Canon DIGITAL IXUS 85 IS
```

Вывести полную информацию по разделам можно следующим образом:

```
<?php
$arr = exif_read_data('foto.jpg', NULL, true, false);
if (!$arr) die('Информации нет');
echo '<pre>';
print_r($arr);
echo '</pre>';
?>
```

- `exif_thumbnail()` — позволяет вывести миниатюру из JPEG- и TIFF-файлов. Формат функции:

```
exif_thumbnail(<Имя файла>, [<Ширина>], [<Высота>],
               [<Формат миниатюры>]);
```

В необязательных параметрах `<Ширина>`, `<Высота>` и `<Формат миниатюры>` можно указать переменные, в которых будут сохранены соответствующие параметры миниатюры. Пример использования функции:

```
<?php
$image = exif_thumbnail('foto.jpg');
if (!$image) die('Миниатюры нет');
header('Content-type: image/jpeg');
echo $image;
exit();
?>
```

5.28.3. Работа с готовыми изображениями

Для загрузки готового изображения в качестве холста предусмотрены следующие функции:

- `<Идентификатор> = imagecreatefrompng(<Имя файла>)` — для изображений в формате PNG;

- `<Идентификатор> = imagecreatefromgif(<Имя файла>)` — для изображений в формате GIF;
- `<Идентификатор> = imagecreatefromjpeg(<Имя файла>)` — для изображений в формате JPEG;
- `<Идентификатор> = imagecreatefromwbmp(<Имя файла>)` — для изображений в формате WBMP.

Чтобы вывести изображение в Web-браузер, нужно вначале сформировать соответствующий заголовок с помощью функции `header()`:

```
header("Content-type: image/png");
header("Content-type: image/gif");
header("Content-type: image/jpeg");
header("Content-type: image/vnd.wap.wbmp");
```

А затем вывести изображение с помощью соответствующей формату функции:

- `imagepng()` — для изображений в формате PNG:

  `imagepng(<Идентификатор>, [<Имя файла>], [<Сжатие>])`

  `<Сжатие>` — число от 0 до 9;
- `imagegif()` — для изображений в формате GIF:

  `imagegif(<Идентификатор>, [<Имя файла>])`
- `imagejpeg()` — для изображений в формате JPEG:

  `imagejpeg(<Идентификатор>, [<Имя файла>], [<Сжатие>])`

  `<Сжатие>` — число от 0 до 100, по умолчанию — 75;
- `imagewbmp()` — для изображений в формате WBMP:

  `imagewbmp(<Идентификатор>, [<Имя файла>])`

В этих функциях необязательный параметр `<Имя файла>` задает имя файла, в который осуществляется вывод. Это означает, что изображение можно вывести не только в Web-браузер, но и сохранить в файл.

После вывода изображения следует освободить ресурсы с помощью функции `imagedestroy()`:

```
imagedestroy(<Идентификатор>);
```

В качестве примера выведем баннер banner.gif в окно Web-браузера. Для этого создадим файл banner.php (листинг 5.63).

Листинг 5.63. Файл banner.php для вывода баннера

```
<?php
$img=imagecreatefromgif("banner.gif");
header("Content-type: image/gif");
imagegif($img);
imagedestroy($img);
?>
```

Вывести баннер в окно Web-браузера в любом документе позволяет следующий HTML-код:

```
<img src="banner.php" alt="">
```

Это аналогично встраиванию обычного изображения:

```
<img src="banner.gif" alt="">
```

Есть одно отличие. Если изображение содержит анимацию, то в окне Web-браузера будет отображен только первый кадр анимации.

5.28.4. Создание нового изображения

Новое палитровое изображение создается с помощью функции `imagecreate()`:

```
<Идентификатор> = imagecreate(<Ширина>, <Высота>);
```

Для создания нового полноцветного изображения можно использовать функцию `imagecreatetruecolor()`:

```
<Идентификатор> = imagecreatetruecolor(<Ширина>, <Высота>);
```

Пример:

```
$img = imagecreatetruecolor(100, 100);
header('Content-type: image/gif');
imagegif($img);
imagedestroy($img);
```

В результате в окно Web-браузера будет выведен квадрат черного цвета. Следует обратить внимание на то, что мы можем выводить созданное изображение в любом формате, указав соответствующую функцию при выводе. Это справедливо не только для созданных изображений, но и для загруженных с помощью функций `imagecreatefrompng()`, `imagecreatefromgif()`, `imagecreatefromjpeg()` и `imagecreatefromwbmp()`:

```
$img = imagecreatefromgif('banner.gif');
header('Content-type: image/jpeg');
imagejpeg($img);
imagedestroy($img);
```

Динамическое создание изображений не имело бы смысла при отсутствии возможности формировать их содержимое. Библиотека GD предоставляет много функций для изменения созданных или прочитанных изображений, которые мы рассмотрим в следующих разделах.

5.28.5. Работа с цветом

Добавить новый цвет в палитру позволяет функция `imagecolorallocate()`:

```
imagecolorallocate(<Идентификатор>, <Доля красного>, <Доля зеленого>,
      <Доля синего>);
```

Пример:

```
$white = imagecolorallocate($img, 255, 255, 255);
```

Первый созданный с помощью функции `imagecolorallocate()` цвет будет использован для заливки фона изображения.

Функция `imagecolordeallocate()` уничтожает объект цвета, созданный функцией `imagecolorallocate()`.

```
imagecolordeallocate(<Идентификатор>, <Цвет>);
```

С помощью функции `imagecolortransparent()` можно сделать определенный цвет палитры прозрачным:

```
imagecolortransparent(<Идентификатор>, <Цвет>)
```

Пример:

```
$white = imagecolorallocate($img, 255, 255, 255);
imagecolortransparent($img, $white);
```

Функция `imagecolorclosest()` возвращает ближайший к указанному цвет из имеющихся в палитре:

```
imagecolorclosest(<Идентификатор>, <Доля красного>, <Доля зеленого>,
      <Доля синего>);
```

Пример:

```
$white = imagecolorclosest($img, 255, 255, 255);
```

Функция `imagecolorat()` возвращает цвет указанной точки изображения:

```
imagecolorat(<Идентификатор>, <X>, <Y>);
```

Координаты точки отсчитываются от верхнего левого угла изображения.

Функция `imagecolorsforindex()` возвращает ассоциативный массив значений RGB-составляющих для указанного идентификатора цвета:

```
imagecolorsforindex(<Идентификатор изображения>, <Цвет>);
```

В качестве примера выведем числовые значения для цвета указанной точки изображения:

```
$img = imagecreatefromgif('banner.gif');
$color = imagecolorat($img, 20, 20);
$rgb = imagecolorsforindex($img, $color);
echo '<pre>';
print_r($rgb);
echo '</pre>';
imagedestroy($img);
```

В результате получим:

```
Array
(
    [red] => 255
```

```
    [green] => 255
    [blue] => 255
    [alpha] => 0
)
```

Функция `imagefill()` позволяет закрасить область одного цвета другим цветом. Достаточно указать координаты точки и новый цвет:

```
imagefill(<Идентификатор>, <X>, <Y>, <Цвет>);
```

Функция `imagefilltoborder()` позволяет закрасить область, ограниченную точками какого-то цвета, другим цветом. Достаточно указать координаты точки, цвета границы и закраски:

```
imagefilltoborder(<Идентификатор>, <X>, <Y>, <Цвет границы>, <Цвет>);
```

Функция `imagecolorstotal()` возвращает число цветов в палитре изображения:

```
imagecolorstotal(<Идентификатор>);
```

Пример:

```
$var = imagecolorstotal($img);
```

Для примера напишем скрипт image.php (листинг 5.64), в котором создается новое изображение 88×31 красного цвета и выводится в Web-браузер.

Листинг 5.64. Файл image.php для вывода динамически созданного изображения

```
<?php
header("Content-type: image/png");
$img = @imagecreate(88, 31);
$background_color = imagecolorallocate($img, 255, 0, 0);
imagefill($img, 0, 0, $background_color);
imagepng($img);
imagedestroy($img);
?>
```

5.28.6. Рисование линий и фигур

Библиотека GD позволяет рисовать следующие фигуры:

- точка:

  ```
  imagesetpixel(<Идентификатор>, <X>, <Y>, <Цвет>);
  ```

  Здесь `<X>` и `<Y>` — координаты точки, которые, как обычно, отсчитываются от верхнего левого угла;

- сплошная линия:

  ```
  imageline(<Идентификатор>, <X1>, <Y1>, <X2>, <Y2>, <Цвет>);
  ```

  Линия задается двумя точками с координатами `(<X1>,<Y1>)` и `(<X2>,<Y2>)`;

- пунктирная линия:

```
imagedashedline(<Идентификатор>, <X1>, <Y1>, <X2>, <Y2>, <Цвет>);
```

Линия задается двумя точками с координатами (<X1>,<Y1>) и (<X2>,<Y2>);

- прямоугольник без заливки:

```
imagerectangle(<Идентификатор>, <X1>, <Y1>, <X2>, <Y2>, <Цвет>);
```

  - <X1> и <Y1> — координаты левого верхнего угла;
  - <X2> и <Y2> — координаты правого нижнего угла;
  - <Цвет> — цвет границы;

- прямоугольник с заливкой:

```
imagefilledrectangle(<Идентификатор>, <X1>, <Y1>, <X2>, <Y2>, <Цвет>);
```

  - <X1> и <Y1> — координаты левого верхнего угла;
  - <X2> и <Y2> — координаты правого нижнего угла;
  - <Цвет> — цвет прямоугольника;

- эллипс без заливки:

```
imageellipse(<Идентификатор>, <X>, <Y>, <Ширина>, <Высота>, <Цвет>);
```

  - <X> и <Y> — координаты центра;
  - <Ширина> и <Высота> — размеры;
  - <Цвет> — цвет границы;

- эллипс с заливкой:

```
imagefilledellipse(<Идентификатор>, <X>, <Y>, <Ширина>,
      <Высота>, <Цвет>);
```

  - <X> и <Y> — координаты центра;
  - <Ширина> и <Высота> — размеры;
  - <Цвет> — цвет эллипса;

- многоугольник без заливки:

```
imagepolygon(<Идентификатор>, <Массив координат>,
      <Кол-во вершин>, <Цвет>);
```

  - <Массив координат> — массив координат вершин;
  - <Кол-во вершин> — число вершин многоугольника;
  - <Цвет> — цвет границы;

- многоугольник с заливкой:

```
imagefilledpolygon(<Идентификатор>, <Массив координат>,
      <Кол-во вершин>, <Цвет>);
```

- <Массив координат> — массив координат вершин;
- <Кол-во вершин> — число вершин многоугольника;
- <Цвет> — цвет многоугольника;

❒ дуга:

```
imagearc(<Идентификатор>, <X>, <Y>, <Ширина>, <Высота>, <Старт>,
      <Конец>, <Цвет>)
```

- <X> и <Y> — координаты центра;
- <Ширина> — ширина;
- <Высота> — высота;
- <Старт> — начальный угол в градусах;
- <Конец> — конечный угол в градусах. Угол 0° соответствует направлению вправо, углы отсчитываются по часовой стрелке;
- <Цвет> — цвет границы.

Для примера выведем дугу окружности радиусом 75 точек с центром в точке (100,100), которая соединяет точки (175,100) и (100,175):

```
$img = imagecreate(200, 200);
$white = imagecolorallocate($img, 255, 255, 255);
$red = imagecolorallocate($img, 255, 0, 0);
imagearc($img, 100, 100, 150, 150, 0, 90, $red);
header('Content-type: image/gif');
imagegif($img);
imagedestroy($img);
```

❒ дуга с заливкой. Начальная и конечная точки дуги будут соединены с ее центром, и получившаяся фигура будет закрашена:

```
imagefilledarc(<Идентификатор>, <X>, <Y>, <Ширина>, <Высота>,
      <Старт>, <Конец>, <Цвет>, <Стиль>)
```

- <X> и <Y> — координаты центра;
- <Ширина> — ширина;
- <Высота> — высота;
- <Старт> — начальный угол в градусах;
- <Конец> — конечный угол в градусах. Угол 0° соответствует направлению вправо, углы отсчитываются по часовой стрелке;
- <Цвет> — цвет дуги;
- <Стиль> — стиль соединения начальной точки дуги с ее конечной точкой:
  - IMG_ARC_PIE — соединение прямой линией;
  - IMG_ARC_CHORD — соединение частью окружности;

- IMG_ARC_NOFILL — точки не соединяются, и заливка не выводится (т. е. будет нарисована обычная дуга);
- IMG_ARC_EDGED — если использовано вместе с IMG_ARC_NOFILL, начало и конец дуги соединяются с центром, и заливка не выводится.

Нарисуем сектор окружности с заливкой:

```
$img = imagecreate(200, 200);
$white = imagecolorallocate($img, 255, 255, 255);
$red = imagecolorallocate($img, 255, 0, 0);
imagefilledarc($img, 100, 100, 150, 150, 0, 90, $red,
IMG_ARC_PIE);
header('Content-type: image/gif');
imagegif($img);
imagedestroy($img);
```

Функция `imagesetthickness()` устанавливает толщину линий при рисовании:

```
imagesetthickness(<Идентификатор>, <Толщина в пикселах>);
```

По умолчанию толщина линий составляет 1 пиксел.

Функция `imagesetstyle()` позволяет указать стиль линий при рисовании:

```
imagesetstyle(<Идентификатор>, <Массив цветов пикселов>);
```

В качестве примера нарисуем линию из черных и белых точек:

```
$img = imagecreate(200, 200);
$white = imagecolorallocate($img, 255, 255, 255);
$black = imagecolorallocate($img, 0, 0, 0);
$style = array($black, $white, $black, $white, $black, $white, $black,
$white);
imagesetstyle($img, $style);
imageline($img, 0, 100, 200, 100, IMG_COLOR_STYLED);
header('Content-type: image/gif');
imagegif($img);
imagedestroy($img);
```

5.28.7. Вывод текста в изображение. Создаем счетчик посещений

Для вывода текста используются следующие функции:

- `imagechar()` — рисует символ на изображении по горизонтали:

  ```
  imagechar(<Идентификатор>, <Шрифт>, <X>, <Y>, <Символ>, <Цвет>);
  ```

- `imagecharup()` — рисует символ на изображении по вертикали:

  ```
  imagecharup(<Идентификатор>, <Шрифт>, <X>, <Y>, <Символ>, <Цвет>);
  ```

- `imagestring()` — отображает строку на изображении по горизонтали:

  ```
  imagestring(<Идентификатор>, <Шрифт>, <X>, <Y>, <Строка>, <Цвет>);
  ```

- `imagestringup()` — отображает строку на изображении по вертикали:

```
imagestringup(<Идентификатор>, <Шрифт>, <X>, <Y>, <Строка>, <Цвет>);
```

В этих функциях параметр `<Шрифт>` задает размер встроенного шрифта, который выражается числом от 1 до 5.

Все четыре функции с буквами русского языка не работают. Для русского языка следует применять TrueType-шрифты (например, arial.ttf). В Windows шрифты расположены в папке C:\Windows\Fonts. Для работы с TrueType-шрифтами предназначены следующие функции:

- `imagettftext()` — рисует строку на изображении TrueType-шрифтом. Функция возвращает массив координат четырех вершин прямоугольника, в который будет вписан текст. Вершины перечисляются в следующем порядке: нижняя левая, нижняя правая, верхняя правая, верхпяя левая. Формат вызова функции:

```
imagettftext(<Идентификатор>, <Размер>, <Угол>, <X>, <Y>, <Цвет>, <Шрифт>,
<Строка>);
```

Параметры имеют следующий смысл:

- `<X>` и `<Y>` — координаты левой крайней точки базовой линии;
- `<Размер>` — размер шрифта;
- `<Угол>` — угол поворота текста. Угол 0 соответствует обычному горизонтальному расположению текста, а поворот осуществляется против часовой стрелки;
- `<Шрифт>` — имя файла со шрифтом.

Например, следующий код

```
$img = imagecreate(200, 200);
$red = imagecolorallocate($img, 255, 0, 0);
$black = imagecolorallocate($img, 0, 0, 0);
$text = 'Testing...';
$fontfile = "C:/WINDOWS/Fonts/arial.ttf";
$arr = imagettftext($img, 20, 0, 11, 21, $black, $fontfile, $text);
print_r($arr);
```

выведет в PHP 5.4

```
Array ( [0] => 11 [1] => 27 [2] => 116 [3] => 27 [4] => 116
[5] => 1 [6] => 11 [7] => 1 )
```

Это означает, что текст вписан в такой прямоугольник:

- `11, 27` — координаты левого нижнего угла;
- `116, 27` — координаты правого нижнего угла;
- `116, 1` — координаты правого верхнего угла;
- `11, 1` — координаты левого верхнего угла;

- `imagettfbbox()` — возвращает координаты прямоугольника, в который вписана строка с помощью TrueType-шрифта:

```
imagettfbbox(<Размер>, <Угол>, <Шрифт>, <Строка>);
```

Например, такой код

```
$text = 'Testing...';
$fontfile = "C:/WINDOWS/Fonts/arial.ttf";
$arr = imagettfbbox(20, 0, $fontfile, $text);
print_r($arr);
```

выведет в PHP 5.4

```
Array ( [0] => -1 [1] => 5 [2] => 104 [3] => 5 [4] => 104
[5] => -21 [6] => -1 [7] => -21 )
```

Это означает, что заданный текст, выведенный под заданным углом таким шрифтом и размером, поместится в прямоугольник со следующими координатами:

- `-1, 5` — левый нижний угол;
- `104, 5` — правый нижний угол;
- `104, -21` — правый верхний угол;
- `-1, -21` — левый верхний угол.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Две координаты по `Y` имеют отрицательные значения. Это происходит потому, что началом координат считается левая точка базовой линии. Базовая линия — это линия, соприкасающаяся с большинством букв снизу. Части некоторых букв могут быть расположены ниже базовой линии (например, буква "у"). Все, что расположено ниже базовой линии, имеет положительные координаты `Y`, а все, что выше — отрицательные. Кроме того, отрицательные значения может иметь и координата `X`.

В качестве примера создадим счетчик посещения с использованием cookies и выведем результат в графическом виде (листинг 5.65).

Листинг 5.65. Счетчик посещений

```
<?php
if (!isset($_COOKIE['id_count'])) $id_count = 0;
else $id_count = $_COOKIE['id_count'];
$id_count++;
setcookie('id_count', $id_count, 0x6FFFFFFF);
header("Content-type: image/png");
$img = imagecreate(88, 31);
$white = imagecolorallocate($img, 255, 255, 255);
$grey = imagecolorallocate($img, 128, 128, 128);
$black = imagecolorallocate($img, 0, 0, 0);
imagerectangle($img, 0, 0, 87, 30, $black);
```

```
$fontfile = 'C:/Windows/Fonts/arial.ttf';
$str = 'Мой счетчик';
$str = iconv("windows-1251", "UTF-8", $str);
imagettftext($img, 8, 0, 11, 13, $grey, $fontfile, $str);
$mass = imagettfbbox(12, 0, $fontfile, $id_count);
$width = intval((88 - $mass[2])/2);
imagettftext($img, 12, 0, $width, 27, $grey, $fontfile, $id_count);
imagepng($img);
imagedestroy($img);
?>
```

При наличии русских букв могут возникнуть проблемы, поэтому мы преобразуем текст из кодировки windows-1251 в UTF-8 при помощи функции `iconv()`. Попробуйте обновить страницу, цифры на счетчике будут увеличиваться.

При выводе текста на готовое изображение возможны проблемы с цветом текста. Особенно это характерно для изображений в формате GIF, т. к. количество цветов ограничено числом 256. Если попытаться добавить новый цвет при максимальном количестве цветов в палитре, функция `imagecolorallocate()` вместо идентификатора цвета вернет значение `false`, а цвет текста будет соответствовать цвету фона. Один из способов решения этой проблемы заключается в использовании функции `imagecolorclosest()`, которая возвращает ближайший цвет, имеющийся в палитре (листинг 5.66).

Листинг 5.66. Вывод текста на готовое изображение

```
<?php
$img = imagecreatefromgif('foto.gif');
header('Content-type: image/gif');
$white = imagecolorallocate($img, 255, 255, 255);
if ($white !== false) {
   $str = 'Точный цвет';
}
else {
   $white2 = imagecolorclosest($img, 255, 255, 255);
   $str = 'Ближайший цвет';
}
$fontfile = 'C:/Windows/Fonts/arial.ttf';
$str = iconv("windows-1251", "UTF-8", $str);
imagettftext($img, 28, 0, 100, 400, $white2, $fontfile, $str);
$str = 'Точный цвет';
$str = iconv("windows-1251", "UTF-8", $str);
imagettftext($img, 28, 0, 140, 440, $white, $fontfile, $str);
imagegif($img);
imagedestroy($img);
?>
```

Итак, мы выводим на изображение две строки. Первая строка выводится цветом, ближайшим к указанному цвету в палитре. Вторая строка демонстрирует цвет текста, который получится, если бы мы не использовали функцию `imagecolorclosest()`. Результат выполнения листинга 5.66 изображен на рис. 5.2.

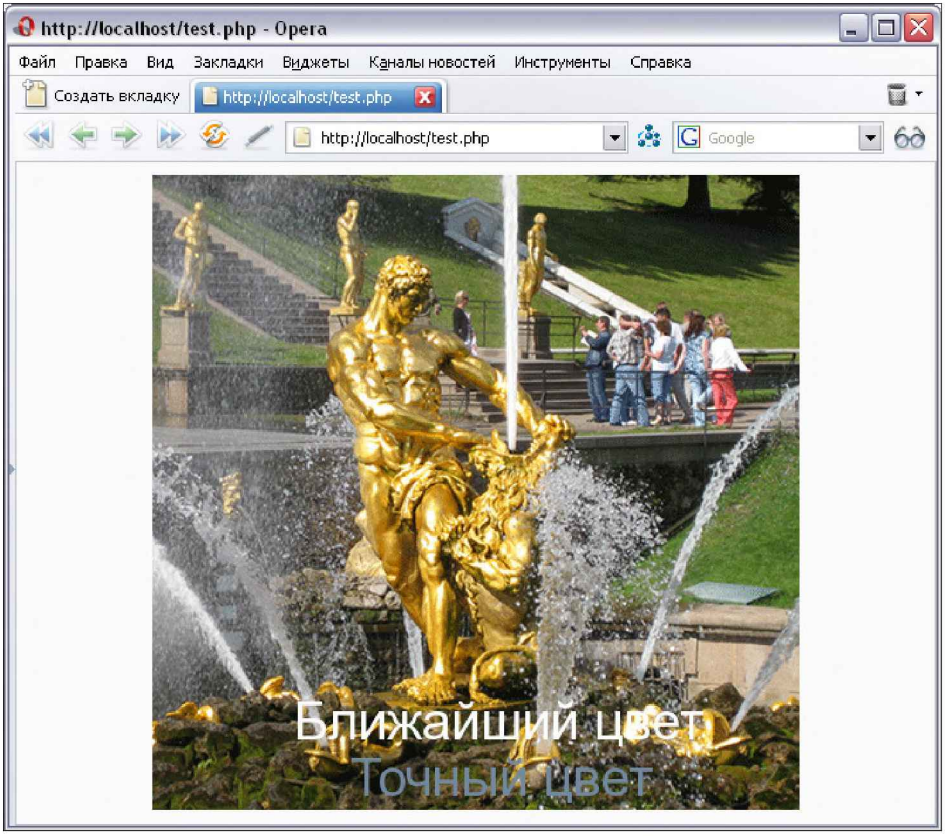


Рис. 5.2. Вывод текста на готовое изображение

5.28.8. Изменение размеров и копирование изображений

Для изменения размеров и копирования изображений применяется функция `imagecopyresampled()`. Формат функции:

```
imagecopyresampled(<Идентификатор1>, <Идентификатор2>, <X1>, <Y1>,
                   <X2>, <Y2>, <Ширина1>, <Высота1>, <Ширина2>,
                   <Высота2>)
```

Параметр `<Идентификатор1>` задает изображение, в которое требуется скопировать изображение, заданное параметром `<Идентификатор2>`. Изображение, в которое копируем, должно быть полноцветным или созданным с помощью функции

imagecreatetruecolor(). Функция imagecreatetruecolor() часто применяется для создания подложки при изменении размеров изображения.

Параметры <X2>, <Y2>, <Ширина2> и <Высота2> задают прямоугольную область в изображении, заданном в параметре <Идентификатор2>, которую мы будем копировать. А параметры <X1>, <Y1>, <Ширина1> и <Высота1> определяют прямоугольную область, в которую будет вставлен копируемый фрагмент.

В качестве примера уменьшим изображение в два раза и выведем полученное изображение в Web-браузер (листинг 5.67).

Листинг 5.67. Изменение размера изображения

```
<?php
$img2 = imagecreatefromgif('foto.gif');
if (!$img2) die('Изображение не удалось загрузить');
// Получаем размеры изображения
list($width2, $height2) = getimagesize('foto.gif');
// Получаем ширину и высоту нового изображения
$width1 = $width2/2;
$height1 = $height2/2;
// Создаем подложку для нового изображения
$img1 = imagecreatetruecolor($width1, $height1);
if (!$img1) die('Изображение не удалось создать');
header('Content-type: image/jpeg');
// Копируем и изменяем размер
imagecopyresampled($img1, $img2, 0, 0, 0, 0, $width1, $height1,
                   $width2, $height2);
imagejpeg($img1);
imagedestroy($img1);
imagedestroy($img2);
?>
```

В листинге 5.66 мы рассмотрели проблему вывода текста на готовое изображение и не получили точный цвет текста. Теперь реализуем вывод текста указанным цветом. Для этого получаем исходное изображение. Создаем подложку такого же размера и копируем исходное изображение на созданную подложку. Далее выводим надпись нужным цветом (листинг 5.68).

Листинг 5.68. Вывод текста на готовое изображение указанным цветом

```
<?php
$foto_name = 'foto.gif';
// Загружаем исходное изображение
$img2 = imagecreatefromgif($foto_name);
if (!$img2) die('Изображение не удалось загрузить');
// Получаем размеры изображения
list($width, $height) = getimagesize($foto_name);
```

```
// Создаем подложку для нового изображения
$img1 = imagecreatetruecolor($width, $height);
if (!$img1) die('Изображение не удалось создать');
// Копируем исходное изображение на подложку
imagecopyresampled($img1, $img2, 0, 0, 0, 0, $width, $height,
                   $width, $height);
imagedestroy($img2);
$white = imagecolorallocate($img1, 255, 255, 255);
if ($white === false) die('Ошибка');
$str = 'Фонтан Самсон';
$str = iconv("windows-1251", "UTF-8", $str);
$fontfile = 'C:/Windows/Fonts/arial.ttf';
// Выводим надпись на изображение
imagettftext($img1, 28, 0, 100, 440, $white, $fontfile, $str);
// Выводим изображение
header('Content-type: image/gif');
imagegif($img1);
imagedestroy($img1);
?>
```

Результат выполнения скрипта изображен на рис. 5.3.

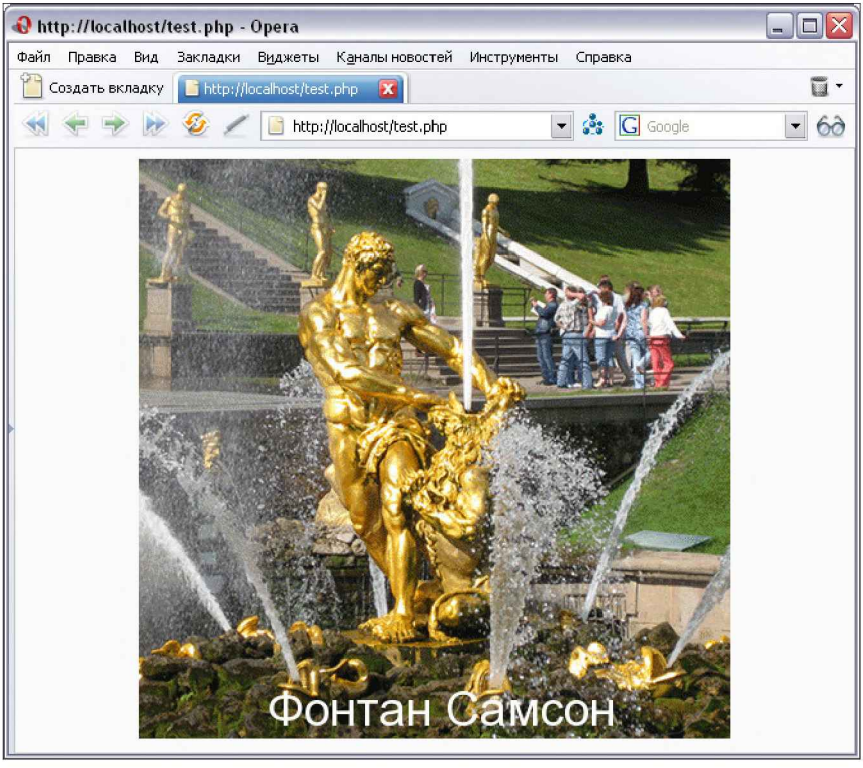


Рис. 5.3. Вывод текста на готовое изображение определенным цветом

Скопировать часть изображения, попутно изменив ее размеры, позволит функция `imagecopyresized()`:

```
imagecopyresized(<Идентификатор1>, <Идентификатор2>, <X1>, <Y1>,
                 <X2>, <Y2>, <Ширина1>, <Высота1>, <Ширина2>,
                 <Высота2>)
```

Параметр `<Идентификатор1>` задает изображение, в которое требуется скопировать фрагмент изображения, заданного параметром `<Идентификатор2>`. Параметры `<X2>`, `<Y2>`, `<Ширина2>` и `<Высота2>` определяют координаты и размеры копируемого фрагмента исходного изображения. А параметры `<X1>`, `<Y1>`, `<Ширина1>` и `<Высота1>` задают прямоугольную область, в которую он будет вставлен.

Функция `imagecopymerge()` выполняет копирование фрагмента изображения с наложением:

```
imagecopyresized(<Идентификатор1>, <Идентификатор2>, <X1>, <Y1>,
                 <X2>, <Y2>, <Ширина1>, <Высота1>, <Ширина2>,
                 <Высота2>, <Наложение>)
```

Последний параметр указывает степень наложения копируемого фрагмента на целевое изображение в виде числа от 0 (наложения не происходит; целевое изображение остается неизмененным) до 100 (полное наложение копируемого фрагмента на целевое изображение).

Аналогичная функция `imagecopymergegray()` выполняет наложение в градациях серого.

5.29. Обработка данных формы

Рассмотрим способы обработки данных каждого элемента формы по отдельности. Напомним, что о работе с полями для выбора файла мы уже говорили в *разд. 5.25.8*.

5.29.1. Текстовое поле, поле ввода пароля и скрытое поле

После отправки формы, содержащей поля

```
<input type="text" name="txt">
<input type="password" name="passw">
<input type="hidden" name="hid" value="">
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод `GET`:

```
$_GET["txt"]
$_GET["passw"]
$_GET["hid"]
```

- метод POST:

```
$_POST["txt"]
$_POST["passw"]
$_POST["hid"]
```

Название переменной совпадает со значением параметра name тега <input>.

Предположим, что в эти поля необходимо ввести первоначальные значения в сценарии. В этом случае возможны следующие проблемы.

- Если в строке есть пробелы, то использование кавычек обязательно.

  Если вывести так:

```
$str = 'Привет всем';
echo '<input type="text" name="txt" value=' . $str . '>';
```

  то в результате поле txt будет содержать текст "Привет", а не "Привет всем". Правильно будет так:

```
$str = 'Привет всем';
echo '<input type="text" name="txt" value="' . $str . '">';
```

- Если в строке есть кавычки, то их следует заменить HTML-эквивалентами.

  Если вывести так:

```
$str = 'Группа "Кино"';
echo '<input type="text" name="txt" value="' . $str . '">';
```

  то в результате поле txt будет содержать текст "Группа ", а не "Группа "Кино"". Правильно будет так:

```
$str = 'Группа "Кино"';
$str = htmlspecialchars($str);
echo '<input type="text" name="txt" value="' . $str . '">';
```

5.29.2. Поле для ввода многострочного текста

После отправки формы

```
<textarea name="txt">Текст</textarea>
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод GET:

```
$_GET["txt"]
```

- метод POST:

```
$_POST["txt"]
```

Предположим, что в поле ввода многострочного текста необходимо ввести первоначальное значение в сценарии. Это можно сделать так:

```
$str = 'Привет всем';
echo '<textarea name="txt" cols="15" rows="10">' . $str . '</textarea>';
```

Однако если строка содержит теги, то могут возникпуть проблемы. Поэтому их необходимо заменить на HTML-эквиваленты:

```
$str = 'Привет </textarea>всем';
$str = htmlspecialchars($str);
echo '<textarea name="txt" cols="15" rows="10">' . $str . '</textarea>';
```

5.29.3. Список с возможными значениями

После отправки формы, содержащей список

```
<select name="color">
<option value="1">White</option>
<option>Red</option>
</select>
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод `GET`:

  `$_GET["color"]`

- метод `POST`:

  `$_POST["color"]`

Название переменной совпадает со значением параметра `name` тега `<select>`.

Значение переменной будет присвоено в зависимости от опции, выбранной в списке. Если выбран пункт **White**, то переменная `$_GET["color"]` будет иметь значение `1` (значение параметра `value`). Если выбран пункт **Red**, то переменная `$_GET["color"]` будет иметь значение `"Red"`, т. к. параметр `value` отсутствует.

Если в списке можно выбрать сразу несколько значений, то все оказывается несколько сложнее (листинг 5.69).

Листинг 5.69. Выбор нескольких значений из списка

```
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<select name="day[]" size="7" multiple>
<option value="1">Понедельник</option>
<option value="2">Вторник</option>
<option value="3">Среда</option>
<option value="4">Четверг</option>
<option value="5">Пятница</option>
<option value="6">Суббота</option>
<option value="7">Воскресенье</option>
</select><br>
<input type="submit" value="Отправить">
</form>
```

```
<?php
if (isset($_GET['day']) && is_array($_GET['day'])) {
   echo 'Выбранные пункты<br>';
   foreach ($_GET['day'] as $item) {
      echo $item . '<br>';
   }
}
?>
```

В параметре `name` после имени следует указать квадратные скобки (символ массива). Все выбранные значения будут помещены в массив.

5.29.4. Флажок

После отправки формы

```
<input type="checkbox" name="check1" value="1"> Текст
<input type="checkbox" name="check2"> Текст
```

в случае, если флажки установлены, на сервере будут созданы следующие переменные окружения:

- метод `GET`:

```
$_GET["check1"]
$_GET["check2"]
```

- метод `POST`:

```
$_POST["check1"]
$_POST["check2"]
```

Если флажки установлены, то переменные будут иметь следующие значения: переменная `$_GET["check1"]` — `1` (значение параметра `value`), а переменная `$_GET["check2"]` — `on` (нет параметра `value`).

Если флажки не установлены, то переменные не создаются! Поэтому необходимо проверять существование переменной:

```
if (isset($_GET['check1'])) echo $_GET['check1'] . '<br>';
if (isset($_GET['check2'])) echo $_GET['check2'] . '<br>';
```

Если флажки объединены в группу, то после имени следует указать квадратные скобки. Значение параметра `name` у всех флажков должно быть одинаковым, а значение параметра `value` — разным:

```
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="checkbox" name="check[]" value="1"> Текст1
<input type="checkbox" name="check[]" value="2"> Текст2
<input type="checkbox" name="check[]" value="3"> Текст3
<input type="submit" value="Отправить">
</form>
```

```
<?php
if (isset($_GET['check']) && is_array($_GET['check'])) {
   echo 'Выбранные пункты<br>';
   foreach ($_GET['check'] as $item) {
      echo $item . '<br>';
   }
}
?>
```

5.29.5. Элемент-переключатель

После отправки формы, содержащей такой код HTML:

```
<input type="radio" name="pol" value="1" checked> Мужской
<input type="radio" name="pol" value="2"> Женский
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод `GET`:

  `$_GET["pol"]`

- метод `POST`:

  `$_POST["pol"]`

> ***Внимание!***
> Если ни один из переключателей не выбран, то переменные не создаются!

Значение переменной `$_GET["pol"]` зависит от выбранного переключателя (`1` или `2`).

5.29.6. Кнопка *Submit*

После отправки формы

```
<input type="submit" name="go" value="Отправить">
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод `GET`:

  `$_GET["go"]`

- метод `POST`:

  `$_POST["go"]`

Зачем для кнопки указывать параметр `name`? Все дело в том, что в одной форме может быть несколько кнопок **Submit**. Кроме того, если наш сценарий обрабатывает сразу несколько форм, то это позволит определнть, какая форма отправлена. Также часто один и тот же сценарий и отображает форму, и обрабатывает ее данные. Если форма отправлена, то переменная будет существовать, если не отправлена, то переменная создана не будет:

```
if (isset($_GET['go'])) {
   echo 'Форма отправлена';
}
else {
   // Вывести форму
}
```

Для этой же цели может использоваться скрытое поле:

```
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="radio" name="pol" value="1" checked> Мужской
<input type="radio" name="pol"> Женский
<input type="hidden" name="go" value="send">
<input type="submit" value="Отправить">
</form>
```

5.29.7. Проверка корректности данных. Создание формы регистрации пользователя

Рассмотрим форму регистрации пользователя с проверкой корректности введенных данных, а заодно и обработку данных в кодировке UTF-8 (листинг 5.70).

Листинг 5.70. Проверка корректности данных

```
<?php
// Файл в кодировке UTF-8 без BOM !!!
mb_internal_encoding('UTF-8'); // Установка кодировки
// Если форма отправлена
if (isset($_POST['go'])) {
   // Создаем короткие имена переменных
   if (isset($_POST['user'])) $user = $_POST['user'];
   else $user = '';
   if (isset($_POST['fam'])) $fam = $_POST['fam'];
   else $fam = '';
   if (isset($_POST['age'])) $age = $_POST['age'];
   else $age = '';
   if (isset($_POST['email'])) $email = $_POST['email'];
   else $email = '';
   if (isset($_POST['pass1'])) $pass1 = $_POST['pass1'];
   else $pass1 = '';
   if (isset($_POST['pass2'])) $pass2 = $_POST['pass2'];
   else $pass2 = '';
   $age = intval($age);
   // Создаем переменную для описания всех ошибок
   $err = '';
   // Проверяем корректность введенных данных
   // Если произошла ошибка, присваиваем переменной $err описание ошибки
```

```
    if (mb_strlen($user)>50 || mb_strlen($user)<2) {
       $err .= 'Недопустимая длина поля Имя<br>';
    }
    if (mb_strlen($fam)>50 || mb_strlen($fam)<2) {
       $err .= 'Недопустимая длина поля Фамилия<br>';
    }
    if (!preg_match('/^[0-9]{1,3}$/su', $age) || $age==0) {
       $err .= 'Неверный возраст<br>';
    }
    if (!preg_match('/^[a-z0-9_.-]+@([a-z0-9-]+\.)+[a-z]{2,6}$/isu', $email)
        || mb_strlen($email)>50) {
       $err .= 'Неверный адрес E-mail<br>';
    }
    if (!preg_match('/^[a-z0-9_.-]{6,16}$/isu', $pass1)) {
       $err .= 'Неверный пароль<br>';
    }
    else {
       if ($pass1 !== $pass2) {
          $err .= 'Пароли должны совпадать<br>';
       }
    }
    // Если ошибок нет, то переменная $err будет пустой
    if ($err == '') {
       // Добавляем данные в базу данных
       // и отправляем подтверждение на E-mail
       // Делаем авторедирект, чтобы очистить данные формы
       header('Location: test.php?reg=ok');
       exit(); // Завершаем работу скрипта
    }
    else { // Если возникли ошибки
       // Заменяем все спецсимволы на HTML-эквиваленты
       $user = htmlspecialchars($user, ENT_COMPAT, 'UTF-8');
       $fam = htmlspecialchars($fam, ENT_COMPAT, 'UTF-8');
       $age = htmlspecialchars($age, ENT_COMPAT, 'UTF-8');
       $email = htmlspecialchars($email, ENT_COMPAT, 'UTF-8');
    }
}
else { // Если форма не отправлена
 $user = $fam = $age = $email = '';
}
header('Content-Type: text/html; charset=utf-8');
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
```

```
 <title>Регистрация пользователя</title>
 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
<h2>Регистрация пользователя</h2>
<?php
if (isset($err)) {
 echo '<div><span style="color:red">При заполнении формы были допущены ';
 echo 'ошибки:</span><br><br>';
 echo $err, "</div>\n";
}
if (isset($_GET['reg']) && $_GET['reg'] == 'ok') {
 // Если регистрация прошла успешно, выводим подтверждение
 echo '<div><b>Регистрация прошла успешно</b></div>';
}
?>
<form action="test.php" method="POST">
<div>
Имя:<br>
<input type="text" name="user" value="<?php echo $user; ?>"><br>
Фамилия:<br>
<input type="text" name="fam" value="<?php echo $fam; ?>"><br>
Возраст:<br>
<input type="text" name="age" value="<?php echo $age; ?>"><br>
E-mail:<br>
<input type="text" name="email" value="<?php echo $email; ?>"><br>
Пароль:<br>
<input type="password" name="pass1"><br>
Повторите пароль:<br>
<input type="password" name="pass2">
<br><br>
<input type="submit" name="go" value="Отправить">
</div>
</form>
</body>
</html>
```

Как видно из примера, все имена полей, заданные с помощью параметра `name`, доступны через переменную окружения `$_POST`. Если форма отправлена (переменная `$_POST['go']` будет существовать), то создаем короткие имена переменных. Может возникнуть вопрос, почему просто не присвоить значения напрямую:

```
$user = $_POST['user'];
```

Все дело в том, что если переменная `$_POST['user']` будет неопределена, то при включенном режиме вывода всех ошибок интерпретатор выдаст предупреждающее сообщение:

```
Notice: Undefined index: user
```

Избежать появления таких сообщений можно, предварительно проверив существование переменной или выключив вывод предупреждающих сообщений, применив один из трех способов:

- в файле php.ini присвоить значение `E_ALL & ~E_NOTICE` директиве `error_reporting`:

```
error_reporting = E_ALL & ~E_NOTICE
```

- в самом начале скрипта вставить функцию `error_reporting()`:

```
error_reporting(E_ALL & ~E_NOTICE);
```

- в самом начале скрипта вставить функцию `ini_set()`:

```
ini_set('error_reporting', 31743);
```

Чтобы сохранить описания всех ошибок, сделанных пользователем при вводе, мы определяем переменную `$err` и присваиваем ей пустую строку. Затем проверяем корректность введенных данных. С помощью функции `mb_strlen()` контролируем длину строк, а с помощью регулярных выражений и функции `preg_match()` проверяем соответствие E-mail и возраста определенному шаблопу. Если во время проверки возникли ошибки, то записываем их в переменпую `$err`.

Если ошибок нет (переменная `$err` останется пустой), добавляем данные в базу и отсылаем подтверждение на E-mail.

Далее делаем авторедирект и завершаем работу скрипта:

```
header('Location: test.php?reg=ok');
exit();
```

Зачем нужен авторедирект? После отправки данных они сохраняются в кэше Web-браузера. Если нажать кнопку **Обновнть** на панели инструментов Web-браузера, то данные повторно будут отправлены серверу. После авторедиректа нажатие кнопки **Обновнть** не будет приводить к повторной отправке данных формы.

Если при проверке были выявлены ошибки, то необходимо заново отобразить форму, вывести сообщения об ошибках и заполнить все поля для редактирования. Так как данные могут содержать специальные символы, мы с помощью функции `htmlspecialchars()` заменяем их на HTML-эквиваленты. После этого можно без опаски заполнить все поля, подставив значения в параметр `value`.

Если форма не была отправлена, то присваиваем переменным пустую строку и выводим форму. В этом случае параметр `value` в элементах формы будет равен пустой строке.

Чтобы документ был правильно обработан Web-браузером, желательно с помощью функции `header()` указать MIME-тип и кодировку документа:

```
header('Content-Type: text/html; charset=utf-8');
```

Часто на хостингах встречается ситуация, когда сервер настроен на кодировку windows-1251. В этом случае сервер автоматически отправит заголовок:

```
Content-Type: text/html; charset=windows-1251
```

Учитывая, что мы работаем с кодировкой UTF-8, Web-браузер может неправильно распознать кодировку и русские буквы будут искажены. По этой причине рекомендую всегда явно указывать кодировку, посылая соответствующий заголовок с помощью функции `header()`.

5.30. Другие полезные функции

В этом разделе мы рассмотрим дополнительные функции, которые могут пригодиться при написании скриптов на PHP. Например, если не удалось сразу подключиться к базе данных, то при помощи функции `sleep()` можно прервать выполнение сценария на некоторое время, а затем снова попытаться подключиться.

5.30.1. Выделение фрагментов исходного кода

С помощью функций `show_source()` и `highlight_file()` можно подсветить синтаксис PHP-кода. Функции абсолютно идентичны. В качестве параметра нужно передать имя файла с PHP-кодом, и в результате получим содержимое файла с выделением синтаксиса в окне Web-браузера.

```
show_source(<Имя файла>);
highlight_file(<Имя файла>);
```

Управлять цветами можно с помощью следующих директив в файле php.ini:

```
highlight.string  = #DD0000
highlight.comment = #FF9900
highlight.keyword = #007700
highlight.default = #0000BB
highlight.html    = #000000
```

5.30.2. Получение информации об интерпретаторе

Для этого предназначены следующие функции:

- `phpinfo()` — возвращает детальную информацию об интерпретаторе:

  ```
  phpinfo();
  ```

- `phpversion()` — служит для определения версии интерпретатора:

  ```
  echo phpversion(); // Выведет: 5.4.35
  ```

- `getlastmod()` — возвращает время последнего изменения сценария:

  ```
  echo date("d.m.y", getlastmod()); // Выведет: 27.11.14
  ```

- `get_current_user()` — позволяет узнать имя пользователя, являющегося владельцем запущенного сценария:

  ```
  echo get_current_user();
  ```

5.30.3. Вывод всех доступных сценарию функций

Функция `get_loaded_extensions()` возвращает массив всех наборов функций, а `get_extension_funcs()` — массив всех функций в наборе, заданном в качестве параметра.

Код листинга 5.71 позволяет получить список всех доступных для сценария функций.

Листинг 5.71. Выдача списка всех функций, доступных сценарию

```
$ext = get_loaded_extensions();
$count = count($ext);
for ($i=0; $i<$count; $i++) {
   echo $ext[$i] . "<br>\n";
   echo "<ul>\n";
   $extf = get_extension_funcs($ext[$i]);
   $count2 = count($extf);
   for ($j=0; $j<$count2; $j++) {
      echo '<li>' . $extf[$j] . "</li>\n";
   }
   echo "</ul>\n";
}
```

Функция `function_exists()` проверяет, определена ли указанная функция. Возвращает `true` в случае, если функция определена среди встроенных или пользовательских функций (листинг 5.72).

Листинг 5.72. Пример использования функции `function_exists()`

```
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="text" name="name_func">
<input type="submit" value="Проверить">
</form>
<?php
if (isset($_GET['name_func'])) {
   if (function_exists($_GET['name_func'])) {
      echo 'Функция ' . $_GET['name_func'] . ' существует';
   }
   else {
      echo 'Функции ' . $_GET['name_func'] . ' нет';
   }
}
?>
```

5.30.4. Засыпание сценария

Функции `sleep()` и `usleep()` прерывают выполнение сценария на указанное время, по истечении которого сценарий продолжит работу:

```
sleep(<Время в секундах>);
usleep(<Время в микросекундах>);
```

Пример:

```
sleep(5);
usleep(1000);
```

5.30.5. Изменение значения директив во время выполнения сценария

С помощью функции `ini_set()` можно изменить значение какой-либо директивы из файла php.ini на время выполнения сценария. Формат функции:

```
ini_set(<Директива>, <Новое значение>);
```

С помощью функции `ini_get()` можно посмотреть текущее значение какой-либо директивы. Формат функции:

```
ini_get(<Директива>);
```

Пример:

```
echo ini_get("default_charset"); // Выведет windows-1251
```

Функция `ini_get_all()` возвращает массив значений всех директив:

```
echo "<pre>";
print_r(ini_get_all());
echo "</pre>";
```

Фрагмент HTML-кода, выведенного этим кодом PHP, будет отображен в Web-браузере так:

```
...
    [date.timezone] => Array
        (
            [global_value] => Europe/Moscow
            [local_value] => Europe/Moscow
            [access] => 7
        )

    [default_charset] => Array
        (
            [global_value] => windows-1251
            [local_value] => windows-1251
            [access] => 7
        )
...
```

Установить значение директивы с помощью функции `ini_set()` можно не всегда. Опция `access`, возвращаемая функцией `ini_get_all()`, позволяет определить, можно ли изменить значение директивы. Она может принимать следующие значения:

- 4 — директива может быть изменена в php.ini или httpd.conf;
- 6 — директива может быть изменена в php.ini, .htaccess или httpd.conf;
- 7 — директива может быть изменена где угодно.

Посмотреть текущее значение какой-либо директивы и откуда ее можно изменить позволяет скрипт, приведенный в листинге 5.73.

Листинг 5.73. Получение значения директивы

```
<form action="<?php echo $_SERVER['SCRIPT_NAME']; ?>">
<input type="text" name="name_ini">
<input type="submit" value="Определить">
</form>
<?php
if (isset($_GET['name_ini'])) {
   $ini = $_GET['name_ini'];
   $arr = ini_get_all();
   if (!isset($arr[$ini])) {
      echo 'Директива не найдена';
      exit();
   }
   echo 'Директива <b>' . $ini . '</b>';
   echo '<br>Глобальное значение: ';
   echo htmlspecialchars($arr[$ini]['global_value']);
   echo '<br>Локальное значение: ';
   echo htmlspecialchars($arr[$ini]['local_value']);
   echo '<br>Изменить можно ';
   switch ($arr[$ini]['access']) {
      case 4: echo 'в php.ini или httpd.conf'; break;
      case 6: echo 'в php.ini, .htaccess или httpd.conf';
              break;
      case 7: echo 'где угодно'; break;
   }
}
?>
```

Для изменения директив PHP из файла .htaccess или httpd.conf используются две директивы: `php_value` и `php_flag`. Директива `php_flag` служит для установки логических значений директив, а `php_value` — для строковых и числовых значений:

```
php_value <Директива> <Значение>
php_flag <Директива> On | Off
```

Пример:

```
php_flag asp_tags Off
```

5.30.6. Выполнение команд, содержащихся в строке

С помощью функции `eval()` можно выполнить строку как PHP-код:

```
$str1 = $str2 = $str3 = $str4 = $str5 = 'Привет';
for ($i=1; $i<6; $i++) {
   $str = '$str' . $i . ' = "Строка' . $i . '";';
   eval($str);
}
echo $str1 . '<br>';
echo $str2 . '<br>';
echo $str3 . '<br>';
echo $str4 . '<br>';
echo $str5 . '<br>';
```

В Web-браузере результат выполнения этого фрагмента будет выведен так:

```
Строка1
Строка2
Строка3
Строка4
Строка5
```

5.31. Объектно-ориентированное программирование

Класс — это тип объекта, включающий в себя набор переменных и функций для управления этими переменными.

Переменные называют *свойствами*, а функции — *методами*.

Для использования методов и свойств класса необходимо создать экземпляр класса. Для этого предназначен оператор `new`. После оператора указывается имя класса, к которому будет относиться данный экземпляр. После имени класса в круглых скобках можно передавать некоторые параметры, задавая таким образом начальные значения свойствам класса.

```
<Экземпляр класса> = new <Имя класса> ([<Параметры>]);
```

Формат обращения к свойствам:

```
<Экземпляр класса>-><Имя свойства без знака $>;
```

Обращение к методам осуществляется аналогично, только после имени метода необходимо указать круглые скобки:

```
<Экземпляр класса>-><Имя метода>();
```

Для удаления экземпляра класса служит функция `unset()`:

```
unset(<Экземпляр класса>);
```

Экземпляр класса также можно удалить, присвоив ему значение `null`:

```
<Экземпляр класса> = null;
```

5.31.1. Создание класса

Описание класса начинается с ключевого слова `class`:

```
class <Имя класса> {
// свойства и методы класса
}
```

Для создания переменной (свойства) внутри класса применяется следующий синтаксис:

```
class <Имя класса> {
   <Область видимости> <Имя переменной со знаком $>;
}
```

Метод внутри класса создается так же, как и обычная функция, с помощью ключевого слова `function`:

```
class <Имя класса> {
   [<Область видимости>] function <Имя функции> ([Параметры]) {
      // Тело функции
   }
}
```

Для обращения к переменным класса или другим функциям внутри функции используется указатель `$this`:

```
class <Имя класса> {
   <Область видимости> <Имя переменной со знаком $>;
   [<Область видимости>] function <Имя функции> ([Параметры]) {
      $this-><Имя переменной без знака $> = <Значение>;
      $this-><Имя функции>();
   }
}
```

5.31.2. Конструктор и деструктор

Чтобы при создании класса присвоить начальные значения каким-либо переменным, необходимо создать метод, имеющий предопределенное название `__construct()`. Такой метод называется *конструктором*. Конструктор всегда автоматически вызывается сразу после создания объекта.

```
class <Имя класса> {
   <Область видимости> <Имя переменной со знаком $>;
   [<Область видимости>] function <Имя функции> ([Параметры]) {
      $this-><Имя переменной без знака $> = <Значение>;
      $this-><Имя функции>();
   }
```

```
   public function __construct(<Параметр1>) {
      $this-><Имя переменной без знака $> = <Параметр1>;
   }
}
```

При создании экземпляра класса параметр `<Параметр1>` можно указать после имени класса в круглых скобках:

```
<Экземпляр класса> = new <Имя класса>(<Параметр1>);
```

Если конструктор вызывается при создании объекта, то перед уничтожением объекта автоматически вызывается метод, называемый *деструктором*. В языке PHP деструктор реализуется в виде предопределенного метода `__destruct()`. Пример приведен в листинге 5.74.

Листинг 5.74. Пример деструктора

```
<?php
class Class1 {
   public $var;
   public function __construct($var) {
      $this->var = $var;
      echo 'Вызван конструктор<br>';
   }
   public function __destruct() {
      echo 'Вызван деструктор';
   }
   public function f_get() {
      return $this->var;
   }
}
$obj = new Class1(5);
echo 'Значение свойства var равно ' . $obj->f_get() . '<br>';
echo 'Вывод перед удалением объекта<br>';
unset($obj);
?>
```

Этот простейший пример выведет:

```
Вызван конструктор
Значение свойства var равно 5
Вывод перед удалением объекта
Вызван деструктор
```

5.31.3. Наследование

Предположим, у нас есть класс (например, `Class1`). С помощью *наследования* мы можем создать новый класс (например, `Class2`), в котором будет доступ ко всем

свойствам и методам класса `Class1`, а также к некоторым новым свойствам и методам (листинг 5.75).

Листинг 5.75. Пример наследования

```
class Class1 {
   public function f_print() {
      echo 'Метод f_print класса Class1<br>';
   }
   public function f_display() {
      echo 'Метод f_display класса Class1<br>';
   }
}

class Class2 extends Class1 {
   public function f_new() {
      echo 'Метод f_new класса Class2<br>';
   }
}
```

Ключевое слово `extends` указывает, что класс `Class2` наследует все свойства и методы класса `Class1`.

```
$obj = new Class2();
$obj->f_new();
$obj->f_print();
$obj->f_display();
```

Результат выполнения листинга 5.75 будет отображен Web-браузером так:

```
Функция f_new класса Class2
Функция f_print класса Class1
Функция f_display класса Class1
```

Если имена функций в классах `Class2` и `Class1` совпадают, то будет использоваться функция из класса `Class2`:

```
class Class2 extends Class1 {
   public function f_new() {
      echo 'Метод f_new класса Class2<br>';
   }
   public function f_display() {
      echo 'Привет';
   }
}

$obj = new Class2();
$obj->f_display();
```

Этот пример выведет `Привет`, а не `Метод f_display класса Class1`.

Чтобы использовать метод, объявленный в родительском классе, следует вызвать его с помощью ключевого слова `parent` (листинг 5.76).

Листинг 5.76. Использование метода, объявленного в родительском классе

```
class Class1 {
   public function f_display() {
      echo 'Метод f_display класса Class1<br>';
   }
}
class Class2 extends Class1 {
   public function f_display() {
      parent::f_display();
      echo 'Привет';
   }
}
$obj = new Class2();
$obj->f_display();
```

Выведет:

```
Метод f_display класса Class1
Привет
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Конструктор и деструктор в родительском классе автоматически не вызываются. Для их вызова также необходимо использовать ключевое слово `parent`.

В некоторых случаях необходимо запретить переопределение метода. Для этого перед определением метода следует указать ключевое слово `final`:

```
class Class1 {
   final public function f_display() {
      echo 'Метод f_display класса Class1<br>';
   }
}
class Class2 extends Class1 {
   public function f_display($msg) {
      echo $msg;
   }
}
$obj = new Class2();
```

Так как перед методом `f_display()` в классе `Class1` стоит ключевое слово `final`, интерпретатор выведет сообщение об ошибке:

```
Fatal error: Cannot override final method Class1::f_display()
```

5.31.4. Статические свойства и методы

Внутри класса можно создать свойство или метод, которые будут доступны без создания экземпляра класса. Для этого перед определением свойства или метода следует указать ключевое слово `static`.

Пример:

```
public static $var = 5;
public static function f_print() {
   // Тело функции
}
```

Доступ к статическому свойству вне класса осуществляется так:

```
echo <Название класса>::$var;
```

Вызов статического метода без создания класса осуществляется следующим образом:

```
<Название класса>::<Название метода>(<Параметры>);
```

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> В таком методе не будет доступа к обычным свойствам и методам класса. Попытка обратиться к ним приведет к ошибке.

Чтобы обратиться к статической переменной из метода класса, можно использовать стандартный способ

```
<Название класса>::<Название переменной с символом $>
```

или указать ключевое слово `self` вместо названия класса:

```
self::<Название переменной с символом $>
```

5.31.5. Объявление констант внутри класса

Константу внутри класса можно объявить с помощью ключевого слова `const`:

```
class <Имя класса> {
   const <Имя константы без $> = <Значение>;
   // Описание свойств и методов класса
}
```

Доступ к константе вне класса осуществляется следующим образом:

```
<Имя класса без $>::<Имя константы без $>
```

Внутри класса к константе можно также обратиться с помощью ключевого слова `self`:

```
self::<Имя константы без $>
```

Пример иллюстрирует листинг 5.77.

Листинг 5.77. Пример обращения к константе

```
class CMyClass {
   const myconst = 10;
   public $myvar;
   public function __construct($i) {
      $this->myvar = $i;
   }
   public function f_Sum1($x) {
      return ($x + self::myconst);
   }
}
$obj = new CMyClass(20);
echo $obj->f_Sum1(5), '<br>';
echo CMyClass::myconst;
```

5.31.6. Определение области видимости

В предыдущем примере любой пользователь может напрямую изменить значение свойства `myvar` нутем присваивания:

```
$obj->myvar = 20;
```

В некоторых случаях требуется контролировать значение свойств класса, а также запрещать использование методов, которые предназначены только для внутренней реализации класса. Например, если в свойстве предполагается хранение определенных значений, то перед присвоением значения мы можем проверить соответствие значения некоторому условию. Если же любой пользователь будет иметь возможность ввести что угодно, минуя нашу проверку, то ни о каком контроле не может быть и речи. Такая концепция сокрытия данных называется *инкапсуляцией*.

В определении свойства и метода могут быть указаны следующие ключевые слова, определяющие область видимости идентификаторов:

- `public` — открытый. Идентификатор доступен для внешнего использования. Если перед определением метода ключевое слово не указано, то метод по умолчанию является открытым. Если ключевое слово не указано перед свойством, то интерпретатор выведет сообщение об ошибке;
- `private` — закрытый. Идентификатор доступен только внутри данного класса;
- `protected` — защищенный. Идентификатор недоступен для внешнего использования, но доступен для данного класса и для его потомков.

Рассмотрим все на примере (листинг 5.78).

Листинг 5.78. Определение области видимости

```
<?php
class Class1 {
   public $var1 = 'public';
```

```
    private $var2 = 'private';
    protected $var3 = 'protected';
    public function f_get_var3() {
        return $this->var2; // Нормально
    }
    public function f_set_var2($s) {
        // Здесь проверяем на допустимость
        $this->var2 = $s;
    }
    public function f_get_var3() {
        return $this->var3; // Нормально
    }
}
class Class2 extends Class1 {
    public function f_var2() {
        return $this->var2; // Ошибка
    }
    public function f_var3() {
        return $this->var3; // Нормально
    }
}
$obj = new Class1();
echo $obj->var1, '<br>'; // 'public'
echo $obj->var2, '<br>'; // Ошибка доступа
echo $obj->f_get_var2(), '<br>'; // 'private'
echo $obj->var3, '<br>'; // Ошибка доступа
echo $obj->f_get_var3(), '<br>'; // 'protected'
$obj = null;
$obj = new Class2();
echo $obj->f_var2(), '<br>'; // Ошибка доступа
echo $obj->f_var3(), '<br>'; // 'protected'
?>
```

Итак, свойство `$var1`, описанное как открытое, доступно для свободного изменения.

Получить доступ к закрытому свойству `$var2` можно только через открытый метод `f_get_var2()`. Кроме того, внутри класса `Class2` свойство `$var2` также напрямую недоступно. Для изменения свойства `$var2` предназначен метод `f_set_var2()`. Именно в этом методе мы можем проверить значение на допустимость.

Свойство `$var3`, объявленное с помощью ключевого слова `protected`, не доступно для свободного изменения. Получить доступ к защищенному свойству `$var3` можно только через открытый метод `f_get_var3()`. В отличие от свойства `$var2`, свойство `$var3` доступно внутри класса `Class2`.

5.31.7. Абстрактные классы и методы

Абстрактным называется класс, служащий своего рода шаблоном для порождения от него унаследованных классов. Создать экземпляр такого класса нельзя — попытка выполнить это приведет к фатальной ошибке.

Аналогично, *абстрактный метод* служит заготовкой для переопределения этого метода в унаследованных классах.

Абстрактный класс объявляется добавлением перед ключевым словом `class` слова `abstract`. Чтобы объявить абстрактный метод, следует вставить ключевое слово `abstract` перед обозначением области видимости метода.

Запомним следующие моменты:

- Если класс содержит хоть один абстрактный метод, он также должен быть объявлен абстрактным.
- Объявление абстрактного метода не должно содержать тела. Фигурные скобки, в которые заключается тело метода, также не ставятся.
- При переопределении абстрактного метода в классе-потомке ключевое слово `abstract` не указывается. Изменение области видимости метода недопустимо и приведет к возникновению фатальной ошибки.
- Наряду с абстрактными методами, абстрактный класс может содержать обычные методы, свойства и константы.

Примеры:

```
abstract class Control {
  private $value = "";
  private $checked = false;
  abstract public function setValue(sNewValue);
  abstract public function setChecked(sNewChecked);
}
```

Объявляем абстрактный класс `Control`, представляющий элемент управления.

```
class TextField extends Control {
  public function setValue(sNewValue) {
    $this->value = sNewValue;
  }
  public function setChecked(sNewChecked) { };
}
```

Объявляем класс `TextField` — потомок класса `Control`. Этот класс будет представлять поле ввода. В нем мы переопределяем унаследованный абстрактный метод `setValue`, который будет заносить значение переданного ему параметра в свойство `value` (значение, занесенное в поле ввода). Переопределенный метод `setChecked` (признак, установлен или сброшен элемент управления) ничего делать не будет, т. к. поле ввода, в отличие от флажка, не может быть установлено или сброшено.

```
class CheckBox extends Control {
  public function setValue(sNewValue) { };
  public function setChecked(sNewChecked) {
    $this->checked = sNewChecked;
  }
}
```

Объявляем класс `CheckBox` — потомок класса `Control`; он представит в нашем коде флажок.

```
$oTextField = new TextField();
$oTextField->setValue("Вася Пупкин");
$oTextField->setChecked(true);
$oCheckBox = new CheckBox();
$oCheckBox->setChecked(true);
$oControl = new Control(); // Ошибка!
```

3.31.8. Интерфейсы

Интерфейс похож на абстрактный класс за следующими исключениями:

- Интерфейс не может содержать свойств.
- Все методы, объявленные в интерфейсе, являются абстрактными и должны быть переопределены в классе, который реализует этот интерфейс. Обычных методов интерфейс содержать не может.
- Все методы интерфейса должны иметь область видимости `public`.
- Класс может реализовывать сразу несколько интерфейсов (множественное наследование).

Синтаксис объявления интерфейса:

```
interface <Имя интерфейса> {
// свойства, методы и константы интерфейса
}
```

Интерфейсы можно наследовать друг от друга. Для указания наследования используется знакомое нам ключевое слово `extends`. Интерфейс может быть потомком нескольких интерфейсов; в этом случае имена интерфейсов-родителей перечисляются через запятую.

Примеры:

```
interface IControl {
  public function setDisabled(sNewDisabled);
}
```

Объявляем интерфейс `IControl`, представляющий элемент управления.

```
interface IControlWithValue extends IControl {
  public function setValue(sNewValue);
}
```

Объявляем интерфейс `IControlWithValue`, представляющий элемент управления с возможностью хранения какого-либо значения и наследующий от интерфейса `IControl`.

```
interface ICheckBox extends IControl {
  public function setChecked(sNewChecked);
}
```

Объявляем интерфейс `ICheckBox`, представляющий флажок и наследующий от интерфейса `IControl`.

В описании класса, реализующего интерфейс, вместо ключевого слова `extends` используется слово `implements`, за которым через запятую перечисляются реализуемые интерфейсы.

```
class TextField implements IControlWithValue {
  private $disabled = "";
  private $value = "";
  public function setDisabled(sNewDisabled) {
    $this->disabled = sNewDisabled;
  }
  public function setValue(sNewValue) {
    $this->value = sNewValue;
  }
}
```

Объявляем класс `TextField`, реализующий интерфейс `IControlWithValue`.

```
class CheckBox implements IControlWithValue, ICheckBox {
  private $disabled = "";
  private $value = "";
  private $checked = false;
  public function setDisabled(sNewDisabled) {
    $this->disabled = sNewDisabled;
  }
  public function setValue(sNewValue) {
    $this->value = sNewValue;
  }
  public function setChecked(sNewChecked) {
    $this->checked = sNewChecked;
  }
}
```

Объявляем класс `CheckBox`, реализующий интерфейсы `IControlWithValue` и `ICheckBox`.

5.31.9. Оператор проверки типа *instanceof*

Оператор проверки типа `instanceof` записывается в формате:

<объект> `instanceof` *<класс или интерфейс>*

Он возвращает `true`, если:

- *объект* создан на основе указанного *класса*;
- *объект* создан на основе потомка указанного `класса`;
- *объект* создан на основе класса, реализующего указанный *интерфейс*.

Примеры:

```
$oCheckBox = new CheckBox();
echo $oCheckBox instanceof CheckBox;
// Выведет на экран: true
echo $oCheckBox instanceof TextField;
// false
echo $oCheckBox instanceof IControlWithValue;
// true
```

5.31.10. Создание шаблона сайта при помощи класса

При создании больших сайтов обычно страницу делят на три части — верхний колонтитул, тело страницы и нижний колонтитул. Для подключения колонтитулов к основному документу используются операторы `require` и `include`.

Нижний колонтитул практически всегда одинаков для всех страниц, а вот верхние колонтитулы по определению не могут совпадать. Для всех страниц сайта недонустим одинаковый заголовок (тег `<title>`). Более того, каждая страница должна иметь уникальное описание для поисковых роботов (тег `<meta>`).

Для реализации верхнего колонтитула создадим класс, позволяющий менять заголовок и описание страницы. Для этого сформируем три файла:

- header.php — верхний колонтитул (листинг 5.79);
- index.php — основное содержание страницы (листинг 5.80);
- footer.php — нижний колонтитул (листинг 5.81).

Листинг 5.79. Содержимое файла header.php

```
<?php
class Header {
   private $title;
   private $meta;
   public function __construct($var1, $var2) {
      $this->title = $var1;
      $this->meta = $var2;
   }
   public function f_display() {
      echo "<html><head>\n";
      echo '<title>' . $this->title . "</title>\n";
      echo '<meta name="description" content="';
      echo $this->meta . "\">\n";
```

```
        echo '<meta http-equiv="Content-Type" content="text/html; ';
        echo "charset=windows-1251\">\n";
        echo '</head>';
        echo "<body>\n";
    }
}
?>
```

Листинг 5.80. Содержимое файла index.php

```
<?php
require_once('header.php');
$title = 'Заголовок';
$meta = 'Описание';
$obj = new Header($title, $meta);
$obj->f_display();
echo '<div>Основное содержание страницы</div>';
require_once('footer.php');
?>
```

Листинг 5.81. Содержимое файла footer.php

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

Если открыть файл index.php в Web-браузере и отобразить исходный код, то мы увидим:

```
<html><head>
<title>Заголовок</title>
<meta name="description" content="Описание">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head><body>
<div>Основное содержание страницы</div></body>
</html>
```

Таким образом, меняя значения переменных `$title` и `$meta`, можно сделать уникальными заголовок и описание каждой страницы.

5.32. Поддержка AJAX со стороны сервера. Кодирование данных в формате JSON

В *разд. 3.20*, описывая технологию AJAX, мы упомянули, что скрипт может загружать с сервера не только файлы, но и данные, сгенерированные серверной программой, написанной, например, на PHP. Теперь настало время поговорить о том, как пишутся такие программы.

Собственно, подготовить данные для передачи клиентскому скрипту мы сможем. Но как закодировать их в формат JSON?

С помощью функции `json_encode()`:

```
json_encode(<кодируемый массив>[, <параметры>])
```

Все строковые значения, присутствующие в кодируемом массиве, должны храниться в кодировке UTF-8. Это ограничение накладывается самим форматом JSON.

Что касается *параметров*, то доступны следующие значения:

- `JSON_HEX_TAG` — преобразовывать символы `<` и `>` в последовательности `\u003C` и `\u003E`;
- `JSON_HEX_AMP` — преобразовывать символы амперсанда (`&`) в последовательности `\u0026`;
- `JSON_HEX_APOS` — преобразовывать символы апострофа (`'`) в последовательности `\u0027`;
- `JSON_HEX_QUOT` — преобразовывать символы двойных кавычек (") в последовательности `\u0022`;
- `JSON_FORCE_OBJECT` — при кодировании обычного, неассоциативного массива возвращать объект, а не массив;
- `JSON_NUMERIC_CHECK` — кодировать строки, содержащие числа, как числа;
- `JSON_BIGINT_AS_STRING` — кодировать большие целые числа в виде строк;
- `JSON_PRETTY_PRINT` — посредством пробелов форматировать возвращаемые данные для удобства чтения;
- `JSON_UNESCAPED_SLASHES` — не экранировать символы слэша (`/`);
- `JSON_UNESCAPED_UNICODE` — не кодировать многобайтовые символы (по умолчанию они преобразуются в последовательности вида `\u<код символа>`).

Эти значения можно объединять оператором двоичного ИЛИ (`|`).

Функция `json_encode()` возвращает строку с закодированными данными или `false`, если при кодировании произошла ошибка.

Примеры:

```
echo json_encode($data);
```

Кодируем массив `$data` и отправляем готовые данные Web-обозревателю.

```
$s = json_encode($data, JSON_PRETTY_PRINT | JSON_NUMERIC_CHECK);
if ($s !== false) {
  echo $s;
} else {
  echo "Ошибка!";
}
```

Кодируем тот же массив, указав форматирование возвращаемых данных и преобразование строк с числами в числа, и учитываем возможность возникновения ошибки.

Выяснить, какого рода ошибка возникла при кодировании данных, можно вызовом не принимающей параметров функции `json_last_error()`. Она вернет одно из следующих значений:

- `JSON_ERROR_NONE` — никаких ошибок не возникло;
- `JSON_ERROR_DEPTH` — слишком много вложенных друг в друга массивов;
- `JSON_ERROR_STATE_MISMATCH` — некорректная структура изначального массива;
- `JSON_ERROR_CTRL_CHAR` — некорректный управляющий символ или неверная кодировка;
- `JSON_ERROR_SYNTAX` — синтаксическая ошибка;
- `JSON_ERROR_UTF8` — некорректный символ UTF-8 или неверная кодировка.

Теперь еще два момента. Во-первых, нам следует задать в качестве MIME-типа отправляемых данных `application/json`, поставив в самое начало скрипта строку

```
header('Content-Type: application/json; charset=utf-8');
```

Во-вторых, настоятельно рекомендуется отключить кэширование отправляемых данных на стороне клиента, добавив в начало скрипта следующие выражения:

```
header("Expires: 0");
header("Cache-Control: no-cache, must-revalidate, post-check=0, ↳
pre-check=0");
header("Pragma: no-cache");
```

В качестве примера мы можем взять страницу, код которой приведен в листинге 3.78, и заменить строку

```
oAJAX.open("GET", "data.json", true);
```

на

```
oAJAX.open("GET", "get_data.php", true);
```

Код программы get_data.php, генерирующей данные, которые будут загружены и выведены на экран помещенным на странице скриптом, показан в листинге 5.82. Этот файл следует сохранить в кодировке UTF-8 без BOM.

Листинг 5.82. Генерирование данных JSON

```
<?php
header("Expires: 0");
header("Cache-Control: no-cache, must-revalidate, post-check=0, ↳
pre-check=0");
header("Pragma: no-cache");
header('Content-Type: application/json; charset=utf-8');
$data = array(
  "status" => 1,
  "data" => array(
    array(
```

```
      "id" => "3.20.1",
      "title" => "Подготовка к загрузке данных"
     ),
    array(
      "id" => "3.20.2",
      "title" => "Отправка запроса"
     ),
    array(
      "id" => "3.20.3",
      "title" => "Получение результата"
     ),
  )
);
$s = json_encode($data);
if ($s !== false) {
  echo $s;
} else {
  echo json_encode(array("status" => 0));
}
?>
```

Если в процессе кодирования данных произошла ошибка, мы отправляем клиентскому скрипту массив с элементом `status`, имеющим значение 0. Тем самым мы сообщим об ошибке.

Не забываем, что данные, имеющие формат JSON, должны быть закодированы с применением кодовой таблицы UTF-8. С связи с этим нам придется сохранить в кодовой таблице UTF-8 (обязательно без BOM!) и сами страницы, выводящие данные JSON на экран, и PHP-скрипты, генерирующие эти данные.

5.33. Шаблонизатор Smarty

Очень часто разработкой сайта занимаются несколько человек. Например, дизайнер делает HTML-верстку и наполняет страницы содержимым, а программист создает динамическую часть. В этом случае дизайнер может абсолютно не разбираться в программировании. Если HTML-код расположен внутри PHP-кода, то это станет серьезным пренятствием для работы дизайнера.

При использовании шаблонизаторов HTML-код отделяется от PHP-кода и располагается в отдельном файле в виде шаблона. В этом случае дизайнеры получают чистый HTML-код с небольшими вкраплениями вида

```
<title>{$title}</title>
```

В этом разделе мы рассмотрим базовые возможности шаблонизатора Smarty, который позволяет не только отделить HTML-код от PHP-кода, но и управлять кэшированием результатов обработки шаблона.

5.33.1. Установка и настройка

Со страницы **http://www.smarty.net/download.php** скачиваем архив Smarty-3.1.21.zip и распаковываем его в текущую папку. Из этого архива нам понадобится папка libs. Переименовываем ее в smarty и копируем в C:\Apache2. Таким образом, файл Smarty.class.php должен быть расположен в C:\Apache2\smarty.

В C:\Apache2\smarty создаем каталог site1, а внутри него четыре папки:

- templates — здесь будем размещать создаваемые шаблоны;
- templates_c — при первой загрузке шаблона он автоматически преобразуется в соответствующий PHP-код, который сохраняется в этой папке. PHP-код создается только один раз при первом запуске, а также после изменения текущего шаблона. Каждый последующий запуск скрипта, использующего шаблон, будет приводить к выполнению PHP-кода, а не к новой компиляции шаблона. Изменять файлы из этой папки вручную не следует;
- configs — для файлов с глобальными переменными. Файлы из этой папки следует загружать внутри шаблона с помощью инструкции `{config_load}`;
- cache — для кэшированных страниц.

Местоположение этих папок задается с помощью свойств `template_dir`, `compile_dir`, `config_dir` и `cache_dir` соответственно. Чтобы в каждом скрипте не указывать путь, создадим новый класс, наследующий все свойства и методы класса `Smarty`, а также определяющий местоположение папок (листинг 5.83).

Листинг 5.83. Содержимое файла MySmarty.php

```
<?php
define('SMARTY_DIR', 'C:/Apache2/smarty/');
require_once(SMARTY_DIR . 'Smarty.class.php');
class MySmarty extends Smarty {
   function __construct() {
      parent::__construct();
      $this->template_dir = 'C:/Apache2/smarty/site1/templates/';
      $this->compile_dir = 'C:/Apache2/smarty/site1/templates_c/';
      $this->config_dir = 'C:/Apache2/smarty/site1/configs/';
      $this->cache_dir = 'C:/Apache2/smarty/site1/cache/';
   }
}
?>
```

Этот файл мы будем подключать во всех скриптах. Разместить файл необходимо в одной папке со скринтом или в одном из каталогов, указанных в директиве `include_path`.

По умолчанию Smarty считает, что все шаблоны создаются с применением кодовой таблицы UTF-8. Если мы собираемся использовать другую кодировку, нам следует

дать знать об этом шаблонизатору. Откроем упомянутый ранее файл C:\Apache2\smarty\Smarty.class.php и поместим в его начало сразу после открывающего тега <?php выражение

```
define('SMARTY_RESOURCE_CHAR_SET', 'windows-1251');
```

Теперь проверим Smarty на работоспособность. Для этого в папке C:\Apache2\smarty\site1\templates создаем файл index.tpl и сохраним в нем код из листинга 5.84.

Листинг 5.84. Содержимое шаблона C:\Apache2\smarty\site1\templates\index.tpl

```
{* Smarty *}
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>{$title}</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
 {* Это комментарий внутри шаблона. В HTML не попадет *}
 <!-- Это HTML-комментарий. Отображается в HTML-документе -->
 <div>Привет, {$name}. Smarty работает!</div>
</body>
</html>
```

Все, что расположено между символами {* и *}, является комментарием, который доступен только внутри шаблона. Он не попадает в исходный HTML-код страницы. Инструкции {$title} и {$name} в дальнейшем будут заменены на значения, указанные в методе assign() (листинг 5.85).

Листинг 5.85. Содержимое файла C:\Apache2\htdocs\index.php

```
<?php
require_once('MySmarty.php'); // См. листинг 5.83
$smarty = new MySmarty();
// Указываем значения для переменных внутри шаблона
$smarty->assign('title', 'Первый шаблон');
$smarty->assign('name', 'Николай');
// Выводим шаблон
$smarty->display('index.tpl');
?>
```

В первой строке подключается файл, содержащий класс MySmarty (листинг 5.83). В следующей строке создается экземпляр класса. Далее с помощью метода assign() указываются значения для переменных внутри шаблона. Чтобы вывести результат в окно Web-браузера, вызывается метод display(), в параметре которого указы-

вается название шаблона. Результат выполнения программы приведен в листинге 5.86.

Листинг 5.86. Результат выполнения листинга 5.85

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>Первый шаблон</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
  <!-- Это HTML-комментарий. Отображается в HTML-документе -->
 <div>Привет, Николай. Smarty работает!</div>
</body>
</html>
```

Как видно из примера, инструкции `{$title}` и `{$name}` были заменены на значения, указанные в методе `assign()`. Никаких признаков Smarty в исходном HTML-коде нет. Теперь откроем файл, который был автоматически создан в папке C:\Apache2\smarty\site1\templates_c после компиляции шаблона (этот файл имеет имя вида *<идентификатор UID>*.file.*<имя файла шаблона с расширением>*.php). Содержимое файла показано в листинге 5.87.

Листинг 5.87. Результат компиляции шаблона index.tpl

```
<?php /* Smarty version Smarty-3.1.21-dev, created on 2014-12-01 10:49:39
         compiled from "C:\Apache2\smarty\site1\templates\index.tpl" */ ?>
<?php /*%%SmartyHeaderCode:26766547c1d9354a701-
49440931%%*/if(!defined('SMARTY_DIR')) exit('no direct access allowed');
$_valid = $_smarty_tpl->decodeProperties(array (
  'file_dependency' =>
  array (
    'e8e82d6f92d3379e3e6578c9c8a4546756264236' =>
    array (
      0 => 'C:\\Apache2\\smarty\\site1\\templates\\index.tpl',
      1 => 1417420175,
      2 => 'file',
    ),
  ),
  'nocache_hash' => '26766547c1d9354a701-49440931',
  'function' =>
  array (
  ),
```

```
  'variables' =>
  array (
    'title' => 0,
    'name' => 0,
  ),
  'has_nocache_code' => false,
  'version' => 'Smarty-3.1.21-dev',
  'unifunc' => 'content_547c1d93615b80_37995063',
),false); /*/%%SmartyHeaderCode%%*/?>
<?php if ($_valid && !is_callable('content_547c1d93615b80_37995063')) {function
content_547c1d93615b80_37995063($_smarty_tpl) {?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title><?php echo $_smarty_tpl->tpl_vars['title']->value;?>
</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>

 <!-- Это HTML-комментарий. Отображается в HTML-документе -->
 <div>Привет, <?php echo $_smarty_tpl->tpl_vars['name']->value;?>
. Smarty работает!</div>
</body>
</html>
<?php }} ?>
```

В начале созданного файла шаблонизатор поместил довольно объемистый служебный код, который нас не интересует. Поэтому сразу обратим внимание на фрагмент, хранящий собственно код нашего шаблона, — там все инструкции Smarty заменены на фрагменты PHP-кода.

Данный файл создается лишь один раз и изменяется только после редактирования шаблона. Во всех остальных случаях запускается PHP-код из этого файла. Таким образом достигается высокая производительность. Еще раз напомним, что изменять этот файл вручную не следует. Все изменения должны вноситься в шаблон.

5.33.2. Управляющие конструкции

Все инструкции Smarty размещаются внутри фигурных скобок. Обратим внимание, что между фигурными скобками и наименованием инструкции не должно быть пробелов, в противном случае шаблонизатор не сможет их обработать. Так, если мы напишем `{if}`, он правильно распознает написанное нами как инструкцию, но, встретив запись `{ if }`, он посчитает, что мы собираемся вывести на экран открывающую фигурную скобку, символы `if` и закрывающую фигурную скобку. (Это позволяет поместить в HTML-код генерируемой страницы фигурные скобки, кото-

рые, как мы знаем, активно используются в таблицах стилей и JavaScript-программах.)

Символ { также можно вставить, вызвав инструкцию {ldelim}, а символ } — инструкцию {rdelim}:

```
{ldelim}Текст внутри фигурных скобок{rdelim}
```

Результат после компиляции шаблона:

```
{Текст внутри фигурных скобок}
```

Еще CSS- или JavaScript-код, включающий символы фигурных скобок, можно заключить в инструкции {literal} и {/literal}:

```
{literal}
function f_print() {
  // Код внутри функции
}
{/literal}
```

Как вы уже знаете, с помощью метода assign() можно передать значения переменных в шаблон:

```
$smarty->assign('name', 'Николай');
```

Внутри шаблона название переменной (с предваряющим символом $) указывается внутри фигурных скобок (например, {$name}). Если передаваемое значение является массивом, то для обращения к его элементу следует указать индекс внутри квадратных скобок:

```
$arr = array(1, 2, 3);
$smarty->assign('arr', $arr);
...
{$arr[0]}
```

При передаче ассоциативного массива или экземпляра класса обратиться к его элементу или свойству можно с помощью точечной нотации:

```
$arr = array('elem1' => 1, 'elem2' => 2);
$obj = new SomeClass();
$smarty->assign('arr', $arr);
$smarty->assign('obj', $obj);
...
{$arr.elem2}
{$obj.prop}
```

С помощью специальной переменной $smarty можно получить доступ к переменным окружения и константам, объявленным в программе:

```
{$smarty.const.MYCONST} {* Обращение к константе *}
{$smarty.get.var1} {* Обращение к $_GET['var1'] *}
{$smarty.post.var2} {* Обращение к $_POST['var2'] *}
{$smarty.server.SCRIPT_NAME} {* Обращение к $_SERVER['SCRIPT_NAME'] *}
{$smarty.cookies.id_count} {* Обращение к $_COOKIE['id_count'] *}
```

Значения переменных могут быть получены из файла конфигурации. В качестве примера создадим файл myconf.conf в папке C:\Apache2\smarty\site1\configs. Содержимое файла:

```
name = 'Николай'
txt = """Текст
   расположен на
   нескольких строках"""
[section1]
var1 = 10
```

Если текст расположен на нескольких строках, то его необходимо разместить внутри тройных кавычек. Подключить такой файл позволяет инструкция `{config_load}`. В параметре `file` указывается название файла:

```
{config_load file='myconf.conf'}
```

Получить значение переменной можно, указав ее название внутри символов `#` или с помощью переменной `$smarty`:

```
{#name#}
{$smarty.config.txt}
```

Чтобы получить значения переменных, которые расположены внутри секций, необходимо указать название секции в параметре `section`:

```
{config_load file='myconf.conf' section='section1'}
{#var1#}
```

Внутри шаблонов возможны условия. Проверка условий осуществляется с помощью конструкции `{if}...{elseif}...{else}...{/if}`. В условиях применяются следующие операторы сравнения:

- `==`, `eq` — равно. Пример:

```
{if $x eq 10}
Переменная x равна 10
{else}
Не равна
{/if}
```

- `!=`, `ne`, `neq` — не равно;
- `>`, `gt` — больше;
- `<`, `lt` — меньше;
- `>=`, `gte`, `ge` — больше или равно;
- `<=`, `lte`, `le` — меньше или равно;
- `===` — идентично;
- `!`, `not` — отрицание;
- `%`, `mod` — остаток от деления;

- `is div by`, `is not div by` — деление без остатка или деление с остатком соответственно. Пример:

```
{if $x is div by 2}
Переменная x делится на 2 без остатка
{else}
Не делится
{/if}
```

- `is even`, `is not even` — проверка на четность или на нечетность соответственно. Пример:

```
{if $x is even}
Переменная x содержит четное значение
{else}
Нет
{/if}
```

- `is odd`, `is not odd` — проверка на нечетность или на четность соответственно.

Внутри условия допустимо использование функций PHP. Например, можно проверить существование переменной с помощью функции `isset()`:

```
{if isset($x)}
Переменная x существует
{else}
Нет
{/if}
```

Объединить несколько логических выражений в одно большое позволяют следующие операторы:

- `&&`, `and` — логическое И;
- `||`, `or` — логическое ИЛИ.

Пример:

```
{if $x > 5 and $x < 10}
Переменная x содержит число от 6 до 9
{elseif $x gte 10 }
Переменная x содержит число >= 10
{else}
Переменная x содержит число < 6 или значение не является числом
{/if}
```

Как вы уже знаете, в шаблон можно передать массив. Перебрать значения массива позволяет конструкция `{section}...{/section}`. В открывающей инструкции `{section}` указываются следующие параметры:

- `name` — имя секции. Через него доступен текущий индекс внутри тела конструкции. Это обязательный параметр;
- `loop` — значение, определяющее число итераций цикла. В качестве значения можно указать название массива. Параметр обязательный;

- `start` — начальный индекс;
- `step` — шаг;
- `max` — максимальное число итераций;
- `show` — если `false`, секция не будет выведена на экран.

Выведем содержимое массива:

```
$arr = array(1, 2, 3, 4);
$smarty->assign('arr', $arr);
...
{section name=i loop=$arr}
{$arr[i]}<br>
{/section}
```

Теперь выведем все четные числа от 2 до 100:

```
{section name=i loop=101 start=2 step=2}
{$smarty.section.i.index}<br>
{/section}
```

Секции поддерживают целый набор свойств, которые можно использовать внутри секций, а некоторые — и вне их. Для доступа к свойствам секции применяется синтаксис вида `$smarty.section.`*<имя секции>*`.`*<имя свойства>*. Доступны следующие свойства:

- `index` — текущий индекс массива. Начинается с нуля (или значения параметра `start`) и увеличивается на единицу (или на значение параметра `step`):

  ```
  {section name=i loop=$arr}
  {$smarty.section.i.index}: {$arr[i]}<br>
  {/section}
  ```

- `index_prev` — индекс предыдущего элемента массива. На первом проходе установлен в –1;
- `index_next` — индекс следующего элемента массива. На последнем проходе больше текущего на единицу (или на значение параметра `step`);
- `iteration` — номер текущего прохода цикла. Всегда начинается с 1;

  ```
  {section name=i loop=$arr step=2}
  {$smarty.section.i.iteration}: {$arr[i]}<br>
  {/section}
  ```

- `first` — `true`, если это первый проход цикла;
- `last` — `true`, если это последний проход цикла:

  ```
  {section name=i loop=$arr}
  {if $smarty.section.i.first}
  <table>
  {/if}
  <tr><td>{$arr[i]}</td></tr>
  ```

```
{if $smarty.section.i.last}
</table>
{/if}
{/section}
```

- `rownum` — то же, что `iteration`;
- `loop` — индекс последнего элемента массива, по которому проходила итерация цикла. Может использоваться вне секции:

```
{section name=i loop=$arr}
{$arr[i]}<br>
{/section}
Всего {$smarty.section.i.loop} элементов.
```

- `show` — значение одноименного параметра секции. Может использоваться вне секции:

```
{section name=i loop=$arr show=$show_section}
{$arr[i]}<br>
{/section}
{if !$smarty.section.i.show}
Секция не выводится на экран
{/if}
```

- `total` — общее число проходов цикла. Может использоваться вне секции:

```
{section name=i loop=$arr step=2}
{$arr[i]}<br>
{/section}
Всего {$smarty.section.i.total} элементов.
```

Если параметр `loop` секции не содержит значений, будет выполнен код, находящийся после инструкции `{sectionelse}`:

```
{section name=i loop=$arr step=2}
{$arr[i]}<br>
{sectionelse}
Массив пуст.
{/section}
```

Для перебора ассоциативного массива предназначена конструкция `{foreach}...{/foreach}`. В открывающей инструкции `{foreach}` указываются следующие параметры:

- `from` — название переменной, в которой хранится массив. Параметр обязательный;
- `item` — позволяет указать название переменной, через которую внутри тела конструкции будет доступно значение текущего элемента массива. Параметр обязательный;
- `key` — позволяет указать название переменной, через которую внутри тела конструкции будет доступен ключ текущего элемента массива;

- `name` — имя цикла для доступа к его свойствам;
- `show` — если `false`, вывод на экран выполняться не будет.

В качестве примера выведем все значения ассоциативного массива:

```
$arr = array('var1' => 1, 'var2' => 2, 'var3' => 3);
$smarty->assign('arr', $arr);
...
{foreach from=$arr item=n key=k}
{$k} =&gt; {$n}<br>
{/foreach}
```

Инструкция `{foreach}` поддерживает свойства `index`, `iteration`, `first`, `last`, `show` и `total`, знакомые нам по инструкции `{section}`. Доступ к ним осуществляется с помощью синтаксиса `$smarty.foreach.`*<имя цикла>*`.`*<имя свойства>*:

```
{foreach from=$arr item=n key=k name=i}
{$smarty.foreach.i.iteration}: {$k} =&gt; {$n}<br>
{/foreach}
```

Если параметр `from` не содержит значений, будет выполнен код, находящийся после инструкции `{foreachelse}`.

Конструкция `{foreach}...{/foreach}` позволяет также перебрать обычный массив. Пример:

```
$arr = array(1, 2, 3, 4, 5);
$smarty->assign('arr', $arr);
...
{foreach from=$arr item=n}
{$n}<br>
{/foreach}
```

В Smarty 3 появилась возможность использовать новый синтаксис директивы `{foreach}...{/foreach}` — в стиле PHP:

```
$arr1 = array(1, 2, 3, 4, 5);
$arr2 = array('var1' => 1, 'var2' => 2, 'var3' => 3);
$smarty->assign('arr1', $arr1);
$smarty->assign('arr2', $arr2);
...
{foreach $arr1 as $n}
{$n}<br>
{/foreach}
{foreach $arr2 as $k=>$n }
{$k} =&gt; {$n}<br>
{/foreach}
```

В этом случае синтаксис обращения к свойствам `index`, `iteration`, `first`, `last`, `show` и `total` должен быть таким: `$`*<имя переменной, где хранится значение элемента массива>*`@`*<имя свойства>*:

```
{foreach $arr2 as $k=>$n}
{$n@iteration}: {$k} =&gt; {$n}<br>
{/foreach}
```

Также появилась возможность вставлять в цикл директивы `{break}` и `{continue}`, аналогичные одноименным командам PHP.

Как вы уже знаете, при создании больших сайтов страницу делят на три части — верхний колонтитул, тело страницы и нижний колонтитул. Инструкция `{include}` позволяет внутри шаблона подключить другой шаблон. Название подключаемого шаблона указывается в параметре `file`. В качестве примера использования инструкции `{include}` создадим четыре файла:

- header.tpl — верхний колонтитул (листинг 5.88);
- footer.tpl — нижний колонтитул (листинг 5.89);
- index.tpl — "тело" страницы (листинг 5.90);
- index.php — основная программа (листинг 5.91).

Листинг 5.88. Содержимое шаблона header.tpl

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
 <title>{$title}</title>
 <meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
</head>
<body>
```

Листинг 5.89. Содержимое шаблона footer.tpl

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

Листинг 5.90. Содержимое шаблона index.tpl

```
{* Smarty *}
{include file=header.tpl}
<div>Привет, {$name}!</div>
{include file=footer.tpl}
```

Листинг 5.91. Содержимое файла index.php

```
<?php
require_once('MySmarty.php'); // См. листинг 5.83
$smarty = new MySmarty();
```

```
// Указываем значения для переменных внутри шаблона
$smarty->assign('title', 'Заголовок страницы');
$smarty->assign('name', 'Николай');
// Выводим шаблон
$smarty->display('index.tpl');
?>
```

Мы можем передать вставляемому фрагменту любые данные через переменные с произвольными именами:

```
{include file=header.tpl body_class="body"}
```

после чего использовать их в коде фрагмента:

```
<body class="{$body_class}">
```

5.33.3. Модификаторы переменных

Внутри шаблона после названия переменной через символ | можно указать один или несколько модификаторов. Эти модификаторы позволяют изменить данные перед их вставкой в шаблон. Например, с помощью модификаторов `upper` и `lower` можно поменять регистр символов. В Smarty доступны следующие модификаторы переменных:

- `upper` — заменяет все символы строки соответствующими прописными буквами:

```
$smarty->assign('str', 'строка в нижнем регистре');
...
{$str|upper} {* Выведет: СТРОКА В НИЖНЕМ РЕГИСТРЕ *}
```

- `lower` — заменяет все символы строки соответствующими строчными буквами:

```
$smarty->assign('str', 'СТРОКА В ВЕРХНЕМ РЕГИСТРЕ');
...
{$str|lower} {* Выведет: строка в верхнем регистре *}
```

- `capitalize` — делает первые символы всех слов прописными:

```
$smarty->assign('str', 'строка в нижнем регистре');
...
{$str|capitalize} {* Выведет: Строка В Нижнем Регистре *}
```

- `cat` — добавляет указанный фрагмент в конец строки:

```
$smarty->assign('str', 'строка');
...
{$str|cat:" добавленный фрагмент"}
{* Выведет: строка добавленный фрагмент *}
```

- `count_characters` — возвращает число символов в строке. Если в качестве значения указать `true`, то при подсчете будут учитываться пробелы:

```
$smarty->assign('str', 'строка с пробелами');
...
{$str|count_characters} {* Выведет: 16 *}
{$str|count_characters:true} {* Выведет: 18 *}
```

- `count_paragraphs` — возвращает число непустых строк:

```
$smarty->assign('str', "строка1\nстрока2\n\nстрока3");
...
{$str|count_paragraphs} {* Выведет: 3 *}
```

- `count_sentences` — возвращает число предложений в строке:

```
$smarty->assign('str', "Это предложение 1. Предложение 2.");
...
{$str|count_sentences} {* Выведет: 2 *}
```

- `count_words` — возвращает число слов в строке:

```
$smarty->assign('str', 'строка с пробелами');
...
{$str|count_words} {* Выведет: 3 *}
```

- `date_format` — форматирует дату согласно указанному шаблону. В строке формата можно указать специальные символы, которые применяются в функции `strftime()` *(см. разд. 5.17)*. Пример:

```
setlocale(LC_ALL, "ru_RU.CP1251", "Russian_Russia.1251");
$smarty->assign('d', time());
...
{$d|date_format} {* Выведет: ноя 17, 2009 *}
{$d|date_format:"%d.%m.%Y"} {* Выведет: 17.11.2009 *}
{$smarty.now|date_format:"%d.%m.%Y"} {* Выведет: 17.11.2009 *}
```

- `default` — позволяет указать значение по умолчанию, если переменная не определена или нустая:

```
$smarty->assign('var1', '');
$smarty->assign('var2', '5');
...
{$var1|default:"Значение по умолчанию"}
{* Выведет: Значение по умолчанию *}
{$var2|default:"Значение по умолчанию"} {* Выведет: 5 *}
```

- `escape` — кодирует или экранирует спецсимволы в строке. В первом параметре могут быть указаны следующие значения: "html", "htmlall", "url", "urlpathinfo", "quotes", "hex", "hexentity", "javascript", "mail". Во втором параметре задается кодировка. Пример:

```
$smarty->assign('str', 'абв"<>\'?');
...
{$str|escape:"html"} {* Выведет: абв&quot;&lt;&gt;&#039;? *}
{$str|escape:"htmlall":"cp1251"}
{* Выведет: &#1072;&#1073;&#1074;&quot;&lt;&gt;&#039;? *}
{$str|escape:"url"} {* Выведет: %E0%E1%E2%22%3C%3E%27%3F *}
{$str|escape:"hex"} {* Выведет: %e0%e1%e2%22%3c%3e%27%3f *}
{$str|escape:"javascript"} {* Выведет: абв\"<>\'? *}
```

- `indent` — создает отступы в начале каждой строки. В первом параметре можно указать число повторений, а во втором — символы, которые будут повторяться. По умолчанию пробел повторяется четыре раза. Пример:

```
$smarty->assign('str', "Строка 1\nСтрока 2");
...
{$str|indent}
{* Выведет:
    Строка 1
    Строка 2
*}
{$str|indent:3:" "}
{* Выведет:
   Строка 1
   Строка 2
*}
```

- `nl2br` — добавляет перед всеми символами новой строки (`\n`) тег `<br />`:

```
$smarty->assign('str', "Строка 1\nСтрока 2\nСтрока 3");
...
{$str|nl2br}
{* Выведет:
Строка 1<br />
Строка 2<br />
Строка 3
*}
```

- `replace` — выполняет замену в строке. Эквивалент PHP-функции `str_replace()` *(см. разд. 5.15.4)*. В первом параметре указывается искомая строка, а во втором — строка для замены:

```
$smarty->assign('str', "Привет, Вася");
...
{$str|replace:"Вася":"Петя"} {* Выведет: Привет, Петя *}
```

- `regex_replace` — осуществляет замену в строке с помощью регулярных выражений. Эквивалент PHP-функции `preg_replace()` *(см. разд. 5.15.9)*. В первом параметре указывается регулярное выражение, а во втором — строка для замены. Удалим все пробельные символы в начале и конце строки:

```
$smarty->assign('str', "   \t  Строка  \r\n  ");
...
{$str|regex_replace:"/(^\s+)|(\s+$)/s":""}
{* Выведет: "Строка" *}
```

- `spacify` — вставляет указанный фрагмент между символами в строке. По умолчанию вставляется пробел:

```
$smarty->assign('str', 'Строка с пробелами');
...
{$str|spacify}
```

```
{* Выведет: С т р о к а  с  п р о б е л а м и *}
{$str|spacify:"-"}
{* Выведет: С-т-р-о-к-а- -с- -п-р-о-б-е-л-а-м-и *}
```

- `string_format` — форматирует число:

```
$smarty->assign('str', 1256.5684);
...
{$str|string_format:"%.2f"} {* Выведет: 1256.57 *}
```

- `strip` — заменяет несколько (идущих подряд) пробельных символов на пробел или указанный фрагмент:

```
$smarty->assign('str', "Строка \n\t с \r пробелами");
...
{$str|strip} {* Выведет: Строка с пробелами *}
{$str|strip:" "}
{* Выведет: Строка с пробелами *}
```

- `strip_tags` — удаляет из строки все HTML-теги. Если в качестве параметра указать значение `true` (значение по умолчанию), то теги будут заменяться на пробел, а если `false` — то никакой символ вставляться вместо тегов не будет:

```
$smarty->assign('str', "'<b>Строка</b>'");
...
{$str|strip_tags} {* Выведет: ' Строка ' *}
{$str|strip_tags:false} {* Выведет: 'Строка' *}
```

- `truncate` — обрезает строку до определенной длины. В первом параметре указывается максимальная длина строки (по умолчанию 80 символов). Во втором параметре можно задать фрагмент, который будет добавлен к обрезанной строке (по умолчанию "..."). Если в третьем параметре указать значение `false` (значение по умолчанию), то строка будет обрезана между словами, а если `true` — то будет строго учитываться длина строки. Если в четвертом параметре указать значение `false` (значение по умолчанию), то строка будет обрезана в конце, а если `true` — то в середине. Пример:

```
$smarty->assign('str', 'Это очень длинная строка');
...
{$str|truncate:15}
{* Выведет: Это очень... *}
{$str|truncate:15:"...":true}
{* Выведет: Это очень дл... *}
{$str|truncate:15:"...":false:true}
{* Выведет: Это оч...строка *}
```

- `wordwrap` — позволяет разбить длинный текст на строки указанной длины. В первом параметре указывается число символов, после которого вставляется символ новой строки (по умолчанию 80 символов). Во втором параметре задается фрагмент, который будет вставлен (по умолчанию "\n"). Если в третьем параметре указать значение `false` (значение по умолчанию), то перевод строки будет

вставляться между словами, а если `true` — то будет строго учитываться длина фрагмента. Пример:

```
$smarty->assign('str', "Очень длинная строка");
...
{$str|wordwrap:5}
{* Выведет:
Очень
длинная
строка
*}
{$str|wordwrap:5:"<br>":true}
{* Выведет: Очень<br>длинн<br>ая<br>строк<br>а *}
```

5.33.4. Кэширование страниц

Как вы уже знаете, шаблон при первом запуске автоматически преобразуется в соответствующий PHP-код, который сохраняется в папке templates_c. Каждый последующий запуск скрипта, использующего шаблон, будет приводить к выполнению этого PHP-кода, а не к новой компиляции шаблона. Шаблонизатор Smarty позволяет также кэшировать результаты обработки шаблона. Это существенно ускоряет выполнение программы, особенно при работе с базой данных или значительном объеме вычислений.

Для включения кэширования результатов обработки шаблона свойству `caching` необходимо присвоить значение `Smarty::CACHING_LIFETIME_CURRENT`:

```
$smarty->caching = Smarty::CACHING_LIFETIME_CURRENT;
```

Если кэширование включено, то после первого запуска скрипта в папке C:\Apache2\smarty\site1\cache будет создан файл, содержащий результаты обработки шаблона. Все последующие запуски скрипта, использующего шаблон, будут приводить к выводу данных из этого файла. Изменять файлы из папки cache вручную не следует.

По умолчанию данные кэшируются на 3600 секунд (1 час). Свойство `cache_lifetime` позволяет задать время жизни кэша (в секундах):

```
$smarty->cache_lifetime = 60; // 1 минута
```

После того как это время жизни истекает, кэш больше не считается актуальным и будет обновлен. Если свойство `compile_check` имеет значение `true`, то все файлы, связанные с кэшем, проверяются на наличие изменений. При существовании изменений кэш больше не считается актуальным и будет обновлен. Чтобы отключить проверку, следует свойству `compile_check` присвоить значение `false`:

```
// Не проверять связанные с кэшем файлы на наличие изменений
$smarty->compile_check = false;
```

Проверить наличие актуального кэша позволяет метод `isCached()`. В качестве параметра указывается название шаблона. Если кэш актуален, то метод возвращает

значение `true`. Это позволяет не выполнять какие-либо действия (например, запрос к базе данных), если кэш является актуальным:

```
if(!$smarty->isCached('index.tpl')) {
   // Обрабатываем данные и передаем значения в шаблон
}
```

Если необходимо очистить кэш определенного шаблона, то следует воспользоваться методом `clearCache()`. Название шаблона указывается в первом параметре:

```
$smarty->clearCache('index.tpl');
```

Очистить все файлы с кэшем позволяет метод `clearAllCache()`:

```
$smarty->clearAllCache();
```

В качестве примера создадим два файла: index.php (листинг 5.92) и index.tpl (листинг 5.93).

Листинг 5.92. Содержимое файла index.php

```
<?php
require_once('MySmarty.php'); // См. листинг 5.83
$smarty = new MySmarty();
// Включаем кэширование
$smarty->caching = true;
// Время жизни кэша
$smarty->cache_lifetime = 60; // 1 минута
// Проверяем наличие актуального кэша
if(!$smarty->isCached('index.tpl')) {
   // Если кэш не является актуальным, то передаем значения
   $smarty->assign('d', date('H:i:s'));
}
// Выводим шаблон
$smarty->display('index.tpl');
?>
```

Листинг 5.93. Содержимое файла index.tpl

```
Это время {$d} помещается в кэш
```

Попробуйте запустить в Web-браузере файл index.php, а затем несколько раз обновите страницу. Время, указанное в первой строке, останется неизменным в течение одной минуты, и лишь по истечении этого срока оно будет обновлено.

Очень часто недостаточно одной копии кэша. Например, при выводе большого количества записей из базы данных необходимо разбить вывод на несколько страниц. Шаблонизатор Smarty позволяет создавать несколько копий кэша для разных идентификаторов. Для этого идентификатор следует указать во втором параметре в методах `display()`, `isCached()` и `clearCache()`. В этом случае в папке cache будет соз-

дано несколько файлов с кэшем. Причем значение идентификатора становится частью названия файла. Поэтому следует тщательно проверять идентификатор на допустимость значения, если он получается от пользователя.

В качестве примера создадим два файла: index.php (листинг 5.94) и index.tpl (листинг 5.95).

Листинг 5.94. Содержимое файла index.php

```
<?php
require_once('MySmarty.php'); // См. листинг 5.83
$smarty = new MySmarty();
// Включаем кэширование страницы
$smarty->caching = true;
// Время жизни кэша
$smarty->cache_lifetime = 60; // 1 минута
if (isset($_GET['page'])) {
   $p = (int)$_GET['page'];
   if ($p < 1 || $p > 3) $p = 1;
}
else $p = 1;
// Проверяем наличие актуального кэша
if(!$smarty->isCached('index.tpl', $p)) {
   // Если кэш не является актуальным, то присваиваем значения
   $smarty->assign('d', date('H:i:s'));
   $smarty->assign('p', $p);
}
// Выводим шаблон
$smarty->display('index.tpl', $p);
?>
```

Листинг 5.95. Содержимое файла index.tpl

```
Время {$d}. Страница номер {$p}<br>
<a href="?page=1">Первая страница</a><br>
<a href="?page=2">Вторая страница</a><br>
<a href="?page=3">Третья страница</a>
```

В этом примере мы использовали один шаблон, но так как указали идентификатор во втором параметре, то кэшированные данные для разных страниц будут разными. Попробуйте перейти по ссылкам. Время, указанное в первой строке, будет меняться только один раз в мипуту, и оно различно для разных страниц.

Если необходимо вывести содержимое каталога, который имеет несколько категорий, то, помимо номера страницы, добавляется еще один параметр — идентификатор категории. В этом случае можно указать идентификатор категории и номер страницы через символ |. В качестве примера создадим два файла: index.php (листинг 5.96) и index.tpl (листинг 5.97).

Листинг 5.96. Содержимое файла index.php

```
<?php
require_once('MySmarty.php'); // См. листинг 5.74
$smarty = new MySmarty();
// Включаем кэширование страницы
$smarty->caching = true;
// Время жизни кэша
$smarty->cache_lifetime = 60; // 1 минута
if (isset($_GET['id'])) {
   $id = (int)$_GET['id'];
   if ($id < 1 || $id > 2) $id = 1;
}
else $id = 1;
if (isset($_GET['page'])) {
   $p = (int)$_GET['page'];
   if ($p < 1 || $p > 2) $p = 1;
}
else $p = 1;
$cache_id = $id . '|' . $p; // Идентификатор кэша
// Проверяем наличие актуального кэша
if(!$smarty->isCached('index.tpl', $cache_id)) {
   // Если кэш не является актуальным, то присваиваем значения
   $smarty->assign('d', date('H:i:s'));
   $smarty->assign('p', $p);
   $smarty->assign('i', $id);
}
// Выводим шаблон
$smarty->display('index.tpl', $cache_id);
?>
```

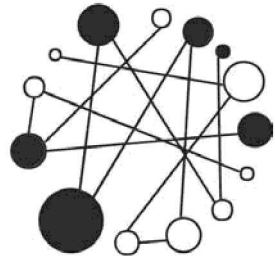
Листинг 5.97. Содержимое файла index.tpl

```
Время {$d}. Категория {$i}. Страница номер {$p}<br>
<a href="?id=1&amp;page=1">Первая страница в категории 1</a>
<a href="?id=1&amp;page=2">Вторая страница в категории 1</a><br>
<a href="?id=2&amp;page=1">Первая страница в категории 2</a>
<a href="?id=2&amp;page=2">Вторая страница в категории 2</a>
```

Если, помимо номера страницы и идентификатора категории, добавляется еще один параметр, то все параметры также можно указать через символ `"|"` во втором параметре в методах `display()`, `isCached()` и `clearCache()`.

В этом разделе мы рассмотрели лишь базовые возможности шаблонизатора Smarty, которых вполне достаточно для использования его на практике. Чтобы получить описание дополнительных возможностей, следует обратиться к документации, английская (русская, к сожалению, отсутствует) версия которой расположена по адресу **http://www.smarty.net/docs/en/**.

ГЛАВА 6



Основы MySQL. Работаем с базами данных

6.1. Основные понятия

MySQL — это система управления реляционными базами данных. Сервер MySQL позволяет эффективно работать с данными и обеспечивает быстрый доступ к информации одновременно нескольким пользователям. При этом доступ к данным предоставляется только пользователям, имеющим на это право.

Что же такое база данных? *Реляционная база данных* — это совокупность двумерных таблиц, связанных отношениями друг с другом. Каждая *таблица* содержит совокупность записей. В свою очередь *запись* — это набор полей, содержащих связанную информацию. Любое *поле* в базе данных имеет имя и определенный тип. Имя таблицы должно быть уникальным в пределах базы данных. В свою очередь имя поля должно быть уникальным в пределах таблицы.

Для выборки записей из базы данных разработан специализированный язык — SQL (Structured Query Language, структурированный язык запросов). С помощью этого языка можно создавать базы данных и таблицы, добавлять, изменять и удалять данные, получать данные по запросу. Но прежде чем изучать SQL, рассмотрим создание реляционных баз данных.

6.2. Нормализация базы данных

Для начала рассмотрим таблицу заказов (табл. 6.1).

Как видно из таблицы, господин Иванов Иван Иванович неоднократно делал покупки. Каждый раз в таблицу добавлялись его адрес, город и телефон. А теперь представьте себе ситуацию, когда господин Иванов Иван Иванович сменил номер телефона. Каждую запись о покупке пришлось бы изменить. Кроме того, напрасно тратится пространство на жестком диске.

По этим причинам имеет смысл вынести данные о клиенте в отдельную таблицу (табл. 6.2).

Таблица 6.1. *Заказы*

Name	Address	City	Phone	Tovar	Date_orders	Price	Col	Sum
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	HDD	2007-06-20	3400	1	3400
Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51	Тюнер	2007-06-20	3100	1	3100
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Монитор	2007-06-25	7200	1	7200
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Тюнер	2007-06-30	3100	1	3100
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Дискета	2007-07-01	10	10	100
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Сканер	2007-07-01	6000	1	6000

Таблица 6.2. *Данные о клиентах*

id_Customer	Name	Address	City	Phone
1	Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45
2	Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51

Теперь наша первоначальная табл. 6.1 примет вид табл. 6.3.

Таблица 6.3. *Заказы*

id_Customer	Tovar	Date_orders	Price	Col	Sum
1	HDD	2007-06-20	3400	1	3400
2	Тюнер	2007-06-20	3100	1	3100
1	Монитор	2007-06-25	7200	1	7200
1	Тюнер	2007-06-30	3100	1	3100
1	Дискета	2007-07-01	10	10	100
1	Сканер	2007-07-01	6000	1	6000

Поле `id_Customer` в табл. 6.2 называется *первичным ключом* и содержит только уникальные записи, т. е. однозначно определяет строку в таблице. Поле `id_Customer` в табл. 6.3 называется *внешним ключом* и может содержать повторяющиеся записи.

Название города также можно вынести в отдельную таблицу (табл. 6.4).

Таблица 6.4. *Названия городов*

id_City	City
1	Санкт-Петербург
2	Москва

В итоге табл. 6.2 примет вид табл. 6.5.

Таблица 6.5. *Данные о клиентах*

id_Customer	Name	Address	id_City	Phone
1	Иванов Иван Иванович	Седова, 7	1	125-14-45
2	Петров Сергей Николаевич	Невский, 88	1	312-12-51

Теперь то же самое можно сделать с названиями товаров (табл. 6.6).

Таблица 6.6. *Информация о товарах*

id_Tovar	Tovar	Price
1	HDD	3400
2	Тюнер	3100
3	Монитор	7200
4	Дискета	10
5	Сканер	6000

И табл. 6.1 еще уменьшится (табл. 6.7).

Таблица 6.7. *Заказы*

id_Customer	id_Tovar	Date_orders	Col	Sum
1	1	2007-06-20	1	3400
2	2	2007-06-20	1	3100
1	3	2007-06-25	1	7200
1	2	2007-06-30	1	3100
1	4	2007-07-01	10	100
1	5	2007-07-01	1	6000

Но это еще не все. Создадим еще табл. 6.8, содержащую элементы заказа.

Таблица 6.8. *Элементы заказа*

id_Orders	id_Tovar	Col
1	1	1
2	2	1
3	3	1
4	2	1
5	4	10
5	5	1

В итоге табл. 6.1 примет вид табл. 6.9.

Таблица 6.9. *Таблица заказов после нормализации*

id_Orders	id_Customer	Date_orders	Sum
1	1	2007-06-20	3400
2	2	2007-06-20	3100
3	1	2007-06-25	7200
4	1	2007-06-30	3100
5	1	2007-07-01	6100

Такой процесс оптимизации базы данных называется *нормализацией*.

Обратите внимание, в табл. 6.8 первичный ключ является *составным* (поля `id_Orders` и `id_Tovar`).

На первый взгляд может показаться, что работать с такой базой данных проблематично. Но это не так. При изменении адреса или телефона покупателя достаточно поменять эти данные только в одной таблице. А отсутствие повторяющихся записей позволит снизить размер базы данных. О том, как получить данные сразу из нескольких таблиц, мы узнаем при изучении языка SQL. Но вначале следует рассмотреть типы данных, которые могут храниться в полях таблицы.

6.3. Типы данных полей

При создании любой таблицы необходимо принимать решение, какой тип данных будет содержать поле, т. к. в отличие, скажем, от массивов в PHP, в базе данных поле может содержать данные только одного типа. Для хранения разных типов данных требуется различный объем памяти. Следует выбирать тип данных, который требует меньшего объема памяти.

Типы данных делятся на числовые, строковые (в которых также можно запоминать бинарные данные) и типы для хранения даты и времени.

6.3.1. Числовые типы

Для хранения чисел используются поля следующих типов:

- `TINYINT [(<Длина в символах>)]` — целые числа от –128 до 127 или от 0 до 255. Занимает 1 байт;
- `BOOL` или `BOOLEAN` — либо 0, либо 1. Синоним для `TINYINT(1)`. Занимает 1 байт;
- `SMALLINT [(<Длина в символах>)]` — целые числа от –32 768 до 32 767 или от 0 до 65 535. Занимает 2 байта;
- `MEDIUMINT [(<Длина в символах>)]` — целые числа от –8 388 608 до 8 388 607 или от 0 до 16 777 215. Занимает 3 байта;
- `INT [(<Длина в символах>)]` — целое 4-байтовое число;
- `INTEGER [(<Длина в символах>)]` — синоним для `INT`;
- `BIGINT [(<Длина в символах>)]` — целое 8-байтовое число;
- `FLOAT [(<Длина в символах>, <Число знаков после запятой>)]` — вещественные числа с диапазоном от ±1.175494351E–38 до ±3.402823466E+38. Занимает 4 байта;
- `DOUBLE [(<Длина в символах>, <Число знаков после запятой>)]` — вещественные числа двойной точности. Занимает 8 байтов;
- `REAL` — синоним для `DOUBLE`;
- `DECIMAL` — дробное число, обеспечивающее повышенную точность;
- `NUMERIC` — синоним для `DECIMAL`.

Если после типа указано слово `UNSIGNED`, значит, поле может содержать только числа без знака.

6.3.2. Строковые типы

Для хранения текста и бинарных данных можно использовать следующие типы:

- `CHAR (<Длина строки в символах>)` — строки фиксированной длины до 255 символов. Строки будут дополняться пробелами до максимальной длины, независимо от размеров строки;
- `VARCHAR (<Длина строки в символах>)` — строки переменной длины до 65 535 символов;
- `TINYTEXT` — строка до 255 символов;
- `TEXT` — строка до 65 535 символов;
- `MEDIUMTEXT` — строка до 16 777 215 символов;
- `LONGTEXT` — строка до 4 294 967 295 символов.

При поиске в текстовых полях регистр символов не учитывается.

Бинарные типы:

- `TINYBLOB` — до 255 байтов;
- `BLOB` — до 65 535 байтов;
- `MEDIUMBLOB` — до 16 777 215 байтов;
- `LONGBLOB` — до 4 294 967 295 байтов;
- `BINARY (<Длина строки в байтах>)` — то же, что `CHAR`, но строки хранятся в бинарном виде;
- `VARBINARY (<Длина строки в байтах>)` — то же, что `VARCHAR`, но строки хранятся в бинарном виде.

При поиске в бинарных полях учитывается регистр символов.

Перечисления и множества:

- `SET ('Значение1', 'Значение2', ...)` — поле может содержать несколько значений из перечисленных. Может быть указано до 64 значений;
- `ENUM ('Значение1', 'Значение2', ...)` — поле может содержать лишь одно из перечисленных значений или `NULL`. Может быть указано до 65 535 значений.

6.3.3. Дата и время

Календарные типы:

- `DATE` — дата в формате `ГГГГ-ММ-ДД`;
- `TIME` — время в формате `ЧЧ:ММ:СС`;
- `DATETIME` — дата и время в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`;
- `YEAR [(2|4)]` — год в двух- или четырехсимвольном формате;
- `TIMESTAMP [(<Тип>)]` — дата и время в формате timestamp: от '1970-01-01 00:00:00' до '2038-01-19 03:14:07'.

6.4. Основы языка SQL

Для выборки записей из базы данных разработан специализированный язык — *SQL* (Structured Query Language, структурированный язык запросов). С помощью этого языка можно создавать базы данных и таблицы, добавлять, изменять и удалять данные, получать данные по запросу. В настоящее время существует множество разновидностей языка SQL. В этой главе книги мы будем изучать SQL применительно к базам данных MySQL. Обратите внимание, некоторые SQL-команды работают только в MySQL.

Команды языка SQL нечувствительны к регистру, но в книге они набраны прописными буквами.

6.4.1. Создание базы данных

Для создания базы данных используется команда:

```
CREATE DATABASE <Имя базы данных>;
```

Пример:

```
CREATE DATABASE `tests`;
```

При создании базы данных можно сразу выбрать кодировку:

```
CREATE DATABASE `tests` DEFAULT CHARACTER SET cp1251
   COLLATE cp1251_general_ci;
```

или

```
CREATE DATABASE `tests` DEFAULT CHARACTER SET utf8
   COLLATE utf8mb4_general_ci;
```

> ***Примечание***
>
> Название базы данных заключают в обратные кавычки `` `tests` ``.

Если база данных с указанным нами именем уже существует, мы получим сообщение об ошибке. В этом случае обезопаситься можно, предусмотрев следующую команду:

```
CREATE DATABASE IF NOT EXISTS `tests` DEFAULT CHARACTER SET cp1251
   COLLATE cp1251_general_ci;
```

Для тестирования команд SQL можно воспользоваться программой phpMyAdmin, которая должна быть доступна по адресу **http://localhost/pma/**.

> ***Внимание!***
>
> Программа будет доступна, только если вы ее установили согласно инструкциям из *разд. 4.7*.

Итак, открываем программу. В левой части сверху находим значок с изображением белого цилиндра и зеленой стрелки. Если навести курсор, то появится подсказка **Окно занроса**. Щелкаем левой кнопкой мыши на значке. Откроется новое окно. В большом текстовом поле на вкладке **SQL** набираем команду:

```
CREATE DATABASE IF NOT EXISTS `tests` DEFAULT CHARACTER SET cp1251
COLLATE cp1251_general_ci;
```

Нажимаем кнопку **Внеред**. В основном окне программы появится сообщение "MySQL вернула пустой результат (т. е. ноль строк). (Запрос занял 0.0120 сек.)". Это значит, что команда создания базы данных была успешно выполнена, но, поскольку она не выполняет выборку данных, серверу нечего нам вывести.

Закрываем все окна, кроме первого. Если новая база данных не отобразилась в иерархическом списке, расположенном слева, обновим страницу. После обновления выберем из этого списка пункт **tests**. В правой части окна отобразится содержимое базы данных tests, точнее сказать, надпись "Таблиц в базе данных не обнаружено", т. к. таблицы мы еще не создавали.

Среди всех баз данных может быть выбрана одна текущая, к которой направляются все команды SQL. Выбирается текущая база данных с помощью команды SQL

```
USE <База данных>;
```

Например, только что созданную базу данных tests можно выбрать SQL-командой

```
USE `tests`;
```

В верхней части страницы расположены вкладки: **Структура**, **SQL**, **Поиск**, **Запрос по шаблону**, **Экспорт**, **Импорт**, **Операции**, **Привилегии**, **Процедуры**, **События** и **Триггеры**. Если в окне не будет хватать места для вывода всех этих вкладок, в верхней части окна появится кнопка **Еще**, при нажатии на которую откроется меню со всеми не представленными на экране вкладками.

Далее нас будет интересовать вкладка **SQL**. Все дальнейшие SQL-запросы к базе данных мы будем набирать именно здесь.

6.4.2. Создание пользователя базы данных

После создания базы данных необходимо создать пользователя базы данных и назначить ему полномочия. *Полномочия* (или *привилегии*) — это права определенного пользователя выполнять заданные действия над определенным объектом. Пользователь должен обладать наименьшим набором привилегий, необходимых для выполнения конкретных задач.

Создание и назначение полномочий осуществляются SQL-командой:

```
GRANT <Привилегии> [<Столбцы>]
ON <База данных>.<Таблица>
TO <Имя пользователя> [IDENTIFIED BY '<Пароль>']
[WITH GRANT OPTION];
```

В параметре `<Привилегии>` могут быть указаны через запятую следующие полномочия:

- `ALL` или `ALL PRIVILEGES` — все полномочия;
- `USAGE` — без всех полномочий;
- `SELECT` — возможность выбирать записи в таблицах;
- `INSERT` — право вставлять новые записи в таблицы;
- `UPDATE` — полномочия изменять значения в существующих полях таблиц;
- `DELETE` — разрешение удалять записи;
- `FILE` — возможность сохранять данные из таблиц в файл и, наоборот, восстанавливать их из файла;
- `CREATE` — право создавать новые базы данных или таблицы. Если в команде `GRANT` указана определенная база данных или таблица, то пользователь может создавать только указанную базу данных или таблицу;
- `ALTER` — полномочия изменять структуру существующих таблиц;

- INDEX — право создавать и удалять индексы определенных таблиц;
- DROP — возможность удаления базы данных или таблицы;
- CREATE TEMPORARY TABLES — разрешение создавать временные таблицы;
- PROCESS — разрешение просматривать и удалять процессы на сервере;
- RELOAD — возможность перезагружать таблицы полномочий;
- SHUTDOWN — право останавливать сервер MySQL;
- SHOW DATABASES — разрешение на просмотр списка всех баз данных на сервере;
- CREATE USER — разрешение создавать, править и удалять пользователей.

В необязательном параметре <Столбцы> может быть указан список имен столбцов, разделенных запятыми, к которым примепяются привилегии.

В параметре <База данных>.<Таблица> может быть указано:

- *.* или * — полномочия предоставляются для всех баз данных в целом;
- <Имя базы данных>.* — полномочия для всех таблиц указанной базы данных;
- <Имя базы данных>.<Имя таблицы> — привилегии относятся только к указанной таблице в указанной базе данных. Если дополнительно задан параметр <Столбцы>, то полномочия назначаются для указанных столбцов.

В параметре <Имя пользователя> указывается имя пользователя (например, den) или Имя_пользователя@Имя_хоста (например, den@localhost). Новому пользователю можно назначить пароль.

Если указана опция WITH GRANT OPTION, то пользователь может предоставлять свои полномочия другим.

Создадим нового пользователя с именем den и назначим ему ограниченные привилегии. Для этого на вкладке **SQL** набираем следующую команду:

```
GRANT select, insert, update, delete, index, alter, create, drop
ON `tests`.*
TO den@localhost IDENTIFIED BY '123';
```

и нажимаем кнопку **OK**. В итоге отобразится надпись "MySQL вернула пустой результат (т. е. ноль строк). (Запрос занял 0.0493 сек.)".

После создания пользователя или изменения привилегий необходимо перезагрузить привилегии с помощью SQL-команды

```
FLUSH PRIVILEGES;
```

Для лишения пользователя полномочий используется команда SQL

```
REVOKE <Привилегии> [<Столбцы>]
ON <База данных>.<Таблица>
TO <Имя пользователя>;
```

Если полномочия были предоставлены опцией WITH GRANT OPTION, то удалить их можно с помощью команды SQL

```
REVOKE GRANT OPTION
ON <База данных>.<Таблица>
TO <Имя пользователя>;
```

Для удаления пользователя используется SQL-команда

```
DROP USER <Имя пользователя>;
```

Для просмотра прав пользователя предназначена команда SQL

```
SHOW GRANTS FOR '<Имя пользователя>'@'<Хост>';
```

Для примера выведем полномочия созданного пользователя den:

```
SHOW GRANTS FOR 'den'@'localhost';
```

6.4.3. Создание таблицы

Создать таблицу в базе данных позволяет SQL-команда

```
CREATE TABLE [IF NOT EXISTS] <Имя таблицы> (
<Имя поля1> <Тип данных> [<Опции>],
<Имя поля2> <Тип данных> [<Опции>],
 ...
) [<Дополнительные опции>];
```

При попытке создать таблицу с именем, совпадающим с именем уже существующей в базе таблицы, будет сгенерирована ошибка. Чтобы исключить ее, следует вставить в команду создания таблицы слова `IF NOT EXISTS`.

В параметре `<Опции>` могут быть указаны следующие значения:

- `NOT NULL` — означает, что поле обязательно должно иметь значение при вставке новой записи в таблицу (если не задано значение по умолчанию). Если опция не указана, то поле может быть пустым;
- `PRIMARY KEY` — указывает, что поле является первичным ключом таблицы. Записи в таком поле должны быть уникальными. Опция также может быть указана после перечисления всех полей;
- `AUTO_INCREMENT` — указывает, что поле является счетчиком: если при вставке новой записи указать `NULL`, то MySQL автоматически генерирует значение, на единицу большее максимального значения, уже существующего в поле. В таблице может быть только одно поле с этой опцией;
- `DEFAULT` — задает для поля значение по умолчанию, которое будет использовано, если при вставке записи для этого поля не было явно указано значение;
- `CHARACTER SET` — определяет кодировку текстового поля;
- `COLLATE` — задает тип сортировки текстового поля.

Возможные значения параметра `<Дополнительные опции>`:

- `ENGINE` — тип таблицы (например, `MyISAM`);
- `DEFAULT CHARSET` — кодировка (например, cp1251);

- AUTO_INCREMENT — начальное значение для автоматической генерации значения поля.

Для вывода всех типов таблиц, поддерживаемых текущей версией MySQL, предназначена SQL-команда

```
SHOW ENGINES;
```

На практике обычно используются два типа таблиц — MyISAM и InnoDB. Ранее, в версиях MySQL, предшествующих 5.5, по умолчанию был назначен тип MyISAM; теперь же таковым является InnoDB. В отличие от типа MyISAM, таблицы типа InnoDB более устойчивы к сбоям, поддерживают транзакции и внешние ключи, но не имеют поддержки полнотекстового поиска и работают несколько медленнее.

Для вывода всех кодировок применяется SQL-команда

```
SHOW CHARACTER SET;
```

Чтобы получить список всех типов сортировки, можно воспользоваться SQL-командой

```
SHOW COLLATION;
```

Создадим таблицы из нашего первоначально рассмотренного примера. Для этого в левой части из списка выбираем базу **tests**. С правой стороны выбираем вкладку **SQL**.

В текстовом поле набираем следующие команды:

```
CREATE TABLE `City` (
   `id_City` INT NOT NULL AUTO_INCREMENT,
   `City` CHAR(50) NOT NULL,
   PRIMARY KEY  (`id_City`)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;

CREATE TABLE `Customers` (
   `id_Customer` INT NOT NULL AUTO_INCREMENT,
   `Name` CHAR(50) NOT NULL,
   `Address` CHAR(255) NOT NULL,
   `id_City` INT NOT NULL,
   `Phone` CHAR(30),
   PRIMARY KEY  (`id_Customer`)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;

CREATE TABLE `Tovar` (
   `id_Tovar` INT NOT NULL AUTO_INCREMENT,
   `Tovar` CHAR(50) NOT NULL,
   `Price` INT NOT NULL,
   PRIMARY KEY  (`id_Tovar`)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;

CREATE TABLE `Orders_Items` (
   `id_Orders` INT NOT NULL,
```

```
    `id_Tovar` INT NOT NULL,
    `Col` TINYINT unsigned,
    PRIMARY KEY  (`id_Orders`, `id_Tovar`)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;

CREATE TABLE `Orders` (
    `id_Orders` INT NOT NULL AUTO_INCREMENT,
    `id_Customer` INT NOT NULL,
    `Date_orders` DATE,
    `Sum` INT,
    PRIMARY KEY  (`id_Orders`)
) ENGINE=MyISAM  DEFAULT CHARSET=cp1251;
```

Можно набрать все команды одновременно, а можно и по отдельности. Чтобы выполнить запрос, нажимаем кнопку **OK**. Все созданные таблицы отображаются слева под списком баз данных (пункт **Повая** ведет на страницу создания новой таблицы):

```
tests
 Новая
 city
 customers
 orders
 orders_items
 tovar
```

Если таблицы не отобразились, то обновите страницу.

Если щелкнуть на названии таблицы, то справа отобразится ее структура.

Вывести все таблицы из указанной базы данных позволяет SQL-команда

```
SHOW TABLES FROM <Имя базы данных>;
```

Для примера выведем все таблицы из базы данных tests:

```
SHOW TABLES FROM `tests`;
```

Чтобы отобразить структуру конкретной таблицы из указанной базы данных, можно воспользоваться командой SQL

```
SHOW COLUMNS FROM <Таблица> FROM <Имя базы данных>;
```

Для примера выведем структуру таблицы `City` из базы данных tests:

```
SHOW COLUMNS FROM `City` FROM `tests`;
```

6.4.4. Добавление данных в таблицу

Для добавления записей в таблицу используется SQL-команда:

```
INSERT INTO <Имя таблицы> [(<Поле1>, <Поле2>, ...)]
VALUES ('<Значение1>', '<Значение2>', ...);
```

Например, добавить две записи в таблицу `City` можно одним из следующих способов:

```
INSERT INTO `City` (`id_City`, `City`)
VALUES (NULL, 'Санкт-Петербург');
INSERT INTO `City` (`id_City`, `City`)
VALUES (NULL, 'Москва');

INSERT INTO `City` (`City`)
VALUES ('Санкт-Петербург');
INSERT INTO `City` (`City`)
VALUES ('Москва');

INSERT INTO `City`
SET `id_City`=NULL, `City`='Санкт-Петербург';
INSERT INTO `City`
SET `id_City`=NULL, `City`='Москва';

INSERT INTO `City`
SET `City`='Санкт-Петербург';
INSERT INTO `City`
SET `City`='Москва';

INSERT INTO `City` VALUES
(NULL, 'Санкт-Петербург'),
(NULL, 'Москва');

INSERT INTO `City` VALUES (NULL, 'Санкт-Петербург');
INSERT INTO `City` VALUES (NULL, 'Москва');
```

Чаще всего на практике применяют последние два способа.

Обратите внимание, для первого поля мы указали значение `NULL`, т. к. для этого поля установлена опция `AUTO_INCREMENT` и MySQL автоматически вставит значение в поле.

Если название таблицы содержит пробел или совпадает с одним из ключевых слов MySQL, то название таблицы необходимо заключить в обратные кавычки.

Пример:

```
INSERT INTO `City` VALUES
(NULL, 'Санкт-Петербург'),
(NULL, 'Москва');
```

Давайте теперь заполним наши созданные таблицы значениями. Для этого выполним следующие SQL-команды:

```
INSERT INTO `City` VALUES
(1, 'Санкт-Петербург'),
(2, 'Москва');
```

```
INSERT INTO `Customers` VALUES
(1, 'Иванов Иван Иванович', 'Седова, 7', 1, '125-14-45'),
(2, 'Петров Сергей Николаевич', 'Невский, 88', 1, '312-12-51');

INSERT INTO `Tovar` VALUES
(1, 'HDD', 3400),
(2, 'Тюнер', 3100),
(3, 'Монитор', 7200),
(4, 'DVD', 30),
(5, 'Сканер', 6000);

INSERT INTO `Orders_Items` VALUES
(1, 1, 1),
(2, 2, 1),
(3, 3, 1),
(4, 2, 1),
(5, 4, 10),
(5, 5, 1);

INSERT INTO `Orders` VALUES
(1, 1, '2014-12-01', 3400),
(2, 2, '2014-11-30', 3100),
(3, 1, '2014-11-28', 7200),
(4, 1, '2014-12-02', 3100),
(5, 1, '2014-11-30', 6100);
```

Обратите внимание, что числа в кавычки не заключаются. А чтобы сохранить целостность базы данных, индексы указываются явным образом.

Если предпринимается попытка вставить запись, а в таблице уже есть запись с таким же значением первичного ключа (или значение индекса `UNIQUE` не уникально), то такая SQL-команда приводит к ошибке. Если необходимо, чтобы такие неуникальные записи обновлялись без вывода сообщения об ошибке, можно использовать следующую SQL-команду:

```
REPLACE [INTO] <Имя таблицы> [(<Поле1>, <Поле2>, ...)]
VALUES ('<Значение1>', '<Значение2>', ...);
```

В качестве примера изменим номер телефона господина Иванова:

```
REPLACE `Customers` VALUES
(1, 'Иванов Иван Иванович', 'Седова, 7', 1, '125-14-47');
```

Если передать уникальное значение, то SQL-команда `REPLACE` аналогична команде `INSERT`. Например, следующая SQL-команда добавит нового покупателя:

```
REPLACE `Customers` VALUES
(NULL, 'Сидоров Олег Николаевич', 'Передовиков, 12', 1, '529-15-63');
```

6.4.5. Обновление записей

Обновление записи осуществляется следующей SQL-командой:

```
UPDATE <Имя таблицы>
SET <Поле1>='<Значение>', <Поле2>='<Значение2>', ...
WHERE <Условие>;
```

> ***Внимание!***
> Если не указано `<Условие>`, то будут обновлены все записи в таблице.

В параметре `<Условие>` могут быть указаны следующие операторы:

- `=` — проверка на равенство;
- `>` — больше;
- `<` — меньше;
- `>=` — больше или равно;
- `<=` — меньше или равно;
- `!=` или `<>` — не равно;
- `IS NOT NULL` — проверка на наличие значения;
- `IS NULL` — проверка поля на отсутствие значения;
- `BETWEEN <Начало> AND <Конец>` — проверяет, является ли значение большим или равным `<Начало>` и меньшим или равным `<Конец>`, например, `pole BETWEEN 0 AND 100`;
- `IN` — содержится в определенном наборе, например, `pole IN ('Монитор', 'HDD')`;
- `NOT IN` — не содержится в определенном наборе, например, `pole NOT IN ('Монитор', 'HDD')`;
- `LIKE` — соответствие шаблону SQL;
- `NOT LIKE` — несоответствие шаблону SQL.

В шаблоне SQL могут использоваться следующие символы:

- `%` — любое число символов;
- `_` — любой одиночный символ.

Можно проверять сразу несколько условий, соединив их логическими операциями:

- `AND` или `&&` — логическое И;
- `OR` или `||` — логическое ИЛИ;
- `XOR` — логическое исключающее ИЛИ;
- `NOT` или `!` — логическое отрицание.

Если название таблицы содержит пробел или совпадает с одним из ключевых слов MySQL, то название таблицы необходимо заключить в обратные кавычки. Для примера изменим телефон одного из клиентов, например, Иванова:

```
UPDATE `Customers` SET `Phone`='125-14-46' WHERE `id_Customer`=1;
```

Господин Иванов у нас числится под номером 1 в таблице `Customers`. Это условие мы и указали.

6.4.6. Удаление записей из таблицы

Удаление записи осуществляется SQL-командой:

```
DELETE FROM <Имя таблицы> WHERE <Условие> [ LIMIT <Число> ];
```

> ***Внимание!***
> Если условие не указано, то будут удалены все записи из таблицы.

С помощью конструкции `LIMIT` можно ограничить максимальное число удаляемых записей. В качестве примера удалим клиента Сидорова:

```
DELETE FROM `Customers` WHERE `Name` LIKE 'Сидоров %' LIMIT 1;
```

Для очистки определенной таблицы используется SQL-команда:

```
TRUNCATE TABLE <Имя таблицы>;
```

Частое обновление и удаление записей приводит к фрагментации таблицы. Чтобы освободить неиспользуемое свободное пространство в таблицах типа `MyISAM`, можно воспользоваться SQL-командой:

```
OPTIMIZE TABLE <Имя таблицы>;
```

Если таблица была повреждена, то восстановить ее позволяет SQL-команда `REPAIR TABLE`:

```
REPAIR TABLE <Имя таблицы>;
```

6.4.7. Изменение структуры таблицы

В ряде случаев нужно изменить структуру уже созданной таблицы. Для этого используется SQL-команда

```
ALTER TABLE <Имя таблицы>
<Преобразование>;
```

В параметре `<Преобразование>` могут быть указаны следующие инструкции:

- `RENAME <Новое имя таблицы>` — переименовывает таблицу;
- `ADD <Имя нового поля> <Тип данных> [FIRST | AFTER <Имя поля>]` — добавляет в таблицу новое поле. Если указана опция `FIRST`, то поле будет добавлено в самое начало, а если `AFTER <Имя поля>` — то после указанного поля. По умолчанию новое поле вставляется в конец таблицы. Обратите внимание, в но-

вом поле нужно задать значение по умолчанию или значение NULL должно быть допустимым, т. к. в таблице уже есть записи;

- ADD PRIMARY KEY (<Имя поля>) — делает указанное поле первичным ключом;
- DROP PRIMARY KEY — удаляет первичный ключ;
- CHANGE <Имя поля> <Новое имя поля> <Новые параметры поля> — изменяет свойства поля. С помощью этой инструкции поле можно переименовать. Если этого не требуется, то <Новое имя поля> должно содержать то же имя, что и <Имя поля>;
- MODIFY <Имя поля> <Тип данных> — изменяет свойства поля;
- DROP <Имя поля> — удаляет поле.

Для примера изменим тип данных поля Address в таблице Customers:

```
ALTER TABLE `Customers` CHANGE `Address` `Address` CHAR(100) NOT NULL;
```

6.4.8. Выбор записей

Выполнить запрос позволяет SQL-команда

```
SELECT <Поле1>, <Поле2>, ...
FROM <Имя таблицы>
[ WHERE <Условие1> ]
[ GROUP BY <Имя поля1> ] [ HAVING <Условие2> ]
[ ORDER BY <Имя поля2> [DESC]]
[ LIMIT <Начало>, <Число записей> ]
```

SQL-команда SELECT ищет все записи в таблице <Имя таблицы>, которые удовлетворяют выражению <Условие1>. Если конструкция WHERE <Условие1> опущена, то будут возвращены все записи из таблицы <Имя таблицы>. Вместо перечисления полей можно указать символ *. В этом случае будут возвращены все поля.

Найденные записи при указанной конструкции ORDER BY <Имя поля2> сортируются по возрастанию. Если в конце указано слово DESC, то записи будут отсортированы в обратном порядке.

Для начала выберем все записи из таблицы City, но только из одного поля:

```
SELECT `City` FROM `City`;
```

В результате будут возвращены только названия городов:

```
Санкт-Петербург
Москва
```

Если вместо названия поля указать символ *, то будут возвращены все поля:

```
SELECT * FROM `City`;
```

Этот запрос вернет

```
1 Санкт-Петербург
2 Москва
```

Теперь выведем названия городов по алфавиту:

```
SELECT * FROM `City` ORDER BY `City`;
```

Теперь названия будут отсортированы:

```
2 Москва
1 Санкт-Петербург
```

А теперь выведем только город с индексом 2:

```
SELECT * FROM `City` WHERE `id_City`=2;
```

В результате мы получим только один город:

```
2 Москва
```

Если требуется, чтобы при поиске выдавались не все найденные записи, а лишь их часть, то нужно использовать параметр `LIMIT`. Этот параметр удобен при выводе большого числа записей. Например, есть каталог из 2000 записей. Вместо того чтобы выводить его за один раз, можно выводить его частями, скажем, по 25 записей за раз. В параметре `LIMIT` задается два значения: `<Начало>` и `<Число записей>`:

```
SELECT * FROM `City` ORDER BY `City` LIMIT 0, 25;
```

Обратите внимание, что первая запись имеет индекс 0. Если в таблице `City` было бы более 25 записей, то мы бы получили только первые 25.

```
SELECT * FROM `City` ORDER BY `City` LIMIT 25, 25;
```

Выбираем следующие 25 записей.

Кроме того, команда `SELECT` позволяет использовать следующие функции, называемые *агрегатными* функциями:

- `COUNT(<Поле>)` — число непустых (т. е. не имеющих значение `NULL`) записей в указанном поле;
- `MIN(<Поле>)` — минимальное значение в указанном поле;
- `MAX(<Поле>)` — максимальное значение в указанном поле;
- `SUM(<Поле>)` — сумма значений в указанном поле;
- `AVG(<Поле>)` — средняя величина значений в указанном поле.

Выведем общее количество заказов:

```
SELECT COUNT(`id_Orders`) FROM `Orders`;
```

Этот SQL-запрос выведет 5. Теперь найдем минимальную сумму заказа:

```
SELECT MIN(`Sum`) FROM `Orders`;
```

В результате мы получим 3100. А теперь выясним максимальную сумму заказа:

```
SELECT MAX(`Sum`) FROM `Orders`;
```

И получим ответ 7200.

Для получения более подробной информации можно воспользоваться конструкцией `GROUP BY`. Например, можно посмотреть среднюю сумму покупок каждого покупателя:

```
SELECT `id_Customer`, AVG(`Sum`) AS s
FROM `Orders`
GROUP BY `id_Customer`
ORDER BY s;
```

Обратите внимание

Мы используем псевдоним для имени поля и по нему сортируем записи от меньшего результата к большему.

Если нужно, например, выбрать клиентов, заказавших больше определенной суммы, то можно воспользоваться конструкцией `HAVING`. Она выполняет те же функции, что и конструкция `WHERE`, но только для конструкции `GROUP BY`.

```
SELECT `id_Customer`, Sum(`Sum`) AS s
FROM `Orders`
GROUP BY `id_Customer`
HAVING s > 4000
ORDER BY s;
```

Можно в одном запросе использовать конструкции `WHERE` и `HAVING`. В этом случае сперва отбираются записи, указанные в конструкции `WHERE`, они группируются, и по ним вычисляются агрегатные функции, а затем из результата отбираются лишь те записи, которые удовлетворяют условию в конструкции `HAVING`.

6.4.9. Выбор записей из нескольких таблиц

SQL-команда `SELECT` позволяет выбирать записи сразу из нескольких таблиц одновременно. Для этого нужно перечислить все таблицы через запятую в конструкции `FROM`. В конструкции `WHERE` через запятую указываются пары полей, являющиеся связуемыми для таблиц. Причем в условии и перечислении полей вначале указывается имя таблицы, а затем через точку имя поля.

Для примера выведем таблицу `Customers`, но вместо индекса города укажем его название:

```
SELECT `Customers`.`Name`, `Customers`.`Address`, `City`.`City`,
 `Customers`.`Phone`
FROM `Customers`, `City`
WHERE `Customers`.`id_City`=`City`.`id_City`;
```

В итоге мы получим табл. 6.10.

Таблица 6.10. *Данные о клиентах*

Name	Address	City	Phone
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46
Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51

Название таблицы можно заменить псевдонимом, который создается через ключевое слово AS после имени таблицы в конструкции FROM. Перепишем предыдущий пример с использованием псевдонимов:

```
SELECT `c`.`Name`, `c`.`Address`, `ct`.`City`, `c`.`Phone`
FROM `Customers` AS `c`, `City` AS `ct`
WHERE `c`.`id_City`=`ct`.`id_City`;
```

Результат будет таким же. Кроме того, если поля в таблицах имеют разные названия, то имя таблицы можно не указывать:

```
SELECT `Name`, `Address`, `City`, `Phone`
FROM `Customers` AS `c`, `City` AS `ct`
WHERE `c`.`id_City`=`ct`.`id_City`;
```

А теперь выведем нашу первоначальную таблицу (см. табл. 6.1):

```
SELECT `c`.`Name`, `c`.`Address`, `ct`.`City`, `c`.`Phone`, `t`.`Tovar`,
`o`.`Date_orders`, `t`.`Price`, `oi`.`Col`
FROM `Customers` AS `c`, `City` AS `ct`, `Tovar` AS `t`, `Orders` AS `o`,
`Orders_Items` AS `oi`
WHERE `c`.`id_City`=`ct`.`id_City` AND
`oi`.`id_Orders`=`o`.`id_Orders` AND
`t`.`id_Tovar`=`oi`.`id_Tovar` AND
`o`.`id_Customer`=`c`.`id_Customer`
ORDER BY `o`.`id_Orders`;
```

В итоге мы получим табл. 6.11.

Таблица 6.11. *Таблица заказов*

Name	Address	City	Phone	Tovar	Date_orders	Price	Col
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	HDD	2007-06-20	3400	1
Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51	Тюнер	2007-06-20	3100	1
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Монитор	2007-06-25	7200	1
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Тюнер	2007-06-30	3100	1
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Дискета	2007-07-01	10	10
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Сканер	2007-07-01	6000	1

Эта таблица практически совпадает с табл. 6.1, с двумя исключениями:

- нет поля Sum. Получить это поле несложно: достаточно перемножить значения полей Price и Col;

- номер телефона господина Иванова изменился, т. к. мы его чуть раньше сами поменяли.

Связывать таблицы можно также с помощью оператора JOIN. Для примера выведем таблицу Customers, но вместо индекса города укажем его название:

```
SELECT `Customers`.`Name`, `Customers`.`Address`, `City`.`City`,
`Customers`.`Phone`
FROM `Customers` JOIN `City`
ON `Customers`.`id_City`=`City`.`id_City`;
```

Отметим, что в этом случае вместо оператора WHERE мы используем ON.

Если необходимо указать дополнительное условие выборки, то это делают в инструкции WHERE.

Для примера выведем информацию о клиентах с фамилией Иванов:

```
SELECT `Customers`.`Name`, `Customers`.`Address`, `City`.`City`,
`Customers`.`Phone`
FROM `Customers` JOIN `City`
ON `Customers`.`id_City`=`City`.`id_City`
WHERE `Customers`.`Name` LIKE 'Иванов %';
```

Если названия полей в таблицах одинаковые, то инструкцию ON можно заменить на USING:

```
SELECT `Customers`.`Name`, `Customers`.`Address`, `City`.`City`,
`Customers`.`Phone`
FROM `Customers` JOIN `City` USING (`id_City`);
```

Оператор JOIN позволяет также объединить несколько таблиц. В качестве примера выведем нашу первоначальную таблицу (см. табл. 6.1):

```
SELECT `c`.`Name`, `c`.`Address`, `ct`.`City`, `c`.`Phone`, `t`.`Tovar`,
`o`.`Date_orders`, `t`.`Price`, `oi`.`Col`
FROM `Customers` AS `c` JOIN `City` AS `ct` JOIN `Tovar` AS `t`
JOIN `Orders` AS `o` JOIN `Orders_Items` AS `oi`
ON `c`.`id_City`=`ct`.`id_City` AND
`oi`.`id_Orders`=`o`.`id_Orders` AND
`t`.`id_Tovar`=`oi`.`id_Tovar` AND
`o`.`id_Customer`=`c`.`id_Customer`
ORDER BY `o`.`id_Orders`;
```

> ***Примечание***
>
> **Оператор** JOIN **имеет два синонима:** CROSS JOIN **и** INNER JOIN.

Добавим нового клиента в таблицу Customers и выведем общее число заказов каждого клиента:

```
INSERT INTO `Customers` VALUES
(NULL, 'Сидоров Олег Николаевич', 'Передовиков, 12', 1, '529-15-63');
```

```
SELECT `Customers`.`Name`, COUNT(`Orders`.`id_Orders`)
FROM `Customers` JOIN `Orders` USING (`id_Customer`)
GROUP BY `Orders`.`id_Customer`;
```

Получим следующий результат:

```
Иванов Иван Иванович      4
Петров Сергей Николаевич  1
```

Как видно из примера, этот запрос вывел только клиентов, сделавших хотя бы один заказ. Так как господин Сидоров не сделал ни одного заказа, то в таблице `Orders` отсутствует запись о нем. Чтобы получить всех клиентов, необходимо использовать левостороннее объединение с помощью инструкции `LEFT JOIN`:

```
<Таблица1> LEFT [OUTER] JOIN <Таблица2> ON
<Таблица1>.<Поле1>=<Таблица2>.<Поле2>
```

Ключевое слово `OUTER` необязательное, его поддержка оставлена для совместимости со стандартом SQL.

Если названия полей в таблицах одинаковые, то вместо инструкции `ON` можно использовать инструкцию `USING`:

```
<Таблица1> LEFT [OUTER] JOIN <Таблица2> USING (<Поле>)
```

При левостороннем объединении возвращаются записи, соответствующие условию `<Таблица1>.<Поле1>=<Таблица2>.<Поле2>`, а также записи из таблицы `<Таблица1>`, которым нет соответствия в таблице `<Таблица2>` (при этом поля из таблицы `<Таблица2>` будут иметь значение `NULL`).

Выведем общее число заказов каждого клиента с помощью левостороннего объединения:

```
SELECT `Customers`.`Name`, COUNT(`Orders`.`id_Orders`) AS `total`
FROM `Customers` LEFT JOIN `Orders` USING (`id_Customer`)
GROUP BY `Orders`.`id_Customer`
ORDER BY `total` DESC;
```

Получим следующий результат:

```
Иванов Иван Иванович      4
Петров Сергей Николаевич  1
Сидоров Олег Николаевич   0
```

Кроме левостороннего можно применить правостороннее объединение с помощью инструкции `RIGHT JOIN`:

```
<Таблица1> RIGHT [OUTER] JOIN <Таблица2> ON
<Таблица1>.<Поле1>=<Таблица2>.<Поле2>
```

Здесь также допустима конструкция с ключевым словом `USING`, если названия полей в обеих таблицах совпадают.

При правостороннем объединении возвращаются записи, соответствующие условию `<Таблица1>.<Поле1>=<Таблица2>.<Поле2>`, а также записи из таблицы `<Таблица2>`,

которым нет соответствия в таблице <Таблица1> (при этом поля из таблицы <Таблица1> будут иметь значение NULL).

Выведем общее число заказов каждого клиента с помощью правостороннего объединения:

```
SELECT `Customers`.`Name`, COUNT(`Orders`.`id_Orders`) AS `total`
FROM `Orders` RIGHT JOIN `Customers` USING (`id_Customer`)
GROUP BY `Orders`.`id_Customer`
ORDER BY `total` DESC;
```

В этом примере мы просто поменяли местами таблицы в инструкции FROM:

```
FROM `Orders` RIGHT JOIN `Customers`
```

6.4.10. Индексы. Ускорение выполнения запросов

Для определения эффективности SQL-запроса предусмотрены операторы EXPLAIN или DESCRIBE. Они выполняют одну и ту же задачу и имеют следующий формат:

```
EXPLAIN | DESCRIBE <Имя таблицы>;
EXPLAIN | DESCRIBE <Запрос SELECT>;
```

Первый вариант выведет структуру указанной таблицы, а второй вариант позволяет выяснить, каким образом выполняется запрос с помощью SQL-команды SELECT. В качестве примера выведем результат поиска клиента по его полному имени. SQL-команду будем выполнять с помощью программы MySQL monitor. Для запуска программы в меню **Пуск** выбираем пункт **Программы (Все программы) | MySQL | MySQL Server 5.5 | MySQL 5.5 Command Line Client**. Откроется черное окошко с запросом ввести пароль. Вводим пароль, заданный при установке сервера MySQL. Если установка проводилась по инструкциям из *разд. 4.6*, то пароль "123456". В случае успешного входа отобразится приветствие сервера, и программа перейдет в режим ожидания команд. В командной строке будет приглашение:

```
mysql>
```

Далее необходимо выбрать базу данных с помощью команды:

```
USE tests;
```

Теперь в командной строке набираем команду:

```
EXPLAIN SELECT * FROM `Customers` WHERE `Name`='Иванов Иван Иванович';
```

Эта команда вернет результат, показанный в табл. 6.12.

Таблица 6.12. *Результат, возвращенный командой* EXPLAIN

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	Customers	ALL	NULL	NULL	NULL	NULL	3	Using where

Рассмотрим таблицу по столбцам:

- `id` — порядковый номер выполненного запроса;
- `select_type` — тип запроса, в нашем случае `SIMPLE`, т. е. простой запрос;
- `table` — название таблицы или таблиц, к которым был выполнен запрос;
- `type` — эффективность выполнения запроса. Может принимать значения `ALL`, `index`, `range`, `index_subquery`, `unique_subquery`, `index_merge`, `ref_or_null`, `fulltext`, `ref`, `eq_ref`, `const` и `system`. Перечисленные значения расположены по возрастанию степени эффективности запроса. Значение `ALL` означает, что просматриваются все записи таблицы, — это самый неэффективный способ;
- `possible_keys` — список всех доступных индексов или `NULL`, если таковых нет;
- `key` — название задействованного в процессе выполнения запроса индекса или `NULL`, если ни один индекс не использовался;
- `key_len` — длина использованного индекса или `NULL`, если индексы не использовались;
- `ref` — названия полей, значения которых сравнивались со значениями, взятыми из индекса, или `NULL`, если индексы не использовались;
- `rows` — число просмотренных в процессе выполнения запроса записей таблицы;
- `Extra` — дополнительные сведения о том, как MySQL выполнял запрос.

После таблицы мы увидим еще одну строку:

```
1 row in set (0.00 sec)
```

Она показывает число выбранных записей и время выполнения запроса с точностью до сотых долей секунды.

Как видно из приведенного примера, для выполнения запроса пришлось просматривать все записи таблицы `Customers`, т. к. записи в неиндексированных полях таблицы расположены в произвольном порядке.

Для ускорения выполнения запросов применяются *индексы* (*ключи*). Индексированные поля всегда поддерживаются в отсортированном состоянии, что позволяет быстро найти необходимую запись, не просматривая все записи. Неиндексированное поле можно сравнить с книгой без предметного указателя, а индексированное поле — с книгой, где он присутствует. Чтобы найти что-либо в первом случае, необходимо последовательно перелистывать страницы книги. Во втором случае достаточно отыскать нужное понятие по алфавиту в предметном указателе, а затем сразу перейти на указанную страницу.

Необходимо сразу заметить, что применение индексов приводит к увеличению размера базы данных, а также к затратам времени на поддержание индекса в отсортированном состоянии при каждом добавлении данных. Поэтому индексировать следует поля, которые очень часто используются в запросах типа

```
SELECT <Список полей> FROM <Таблица> WHERE <Поле>=<Значение>;
```

Существуют следующие виды индексов:

- первичный ключ;
- уникальный индекс;
- обычный индекс;
- индекс FULLTEXT.

Первичный ключ служит для однозначной идентификации каждой записи в таблице. Для создания индекса предусмотрено ключевое слово PRIMARY KEY. При создании таблицы ключевое слово можно указать после определения параметров поля

```
CREATE TABLE `City` (
   `id_City` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
   `City` CHAR(50) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

или после перечисления всех полей

```
CREATE TABLE `City` (
   `id_City` INT NOT NULL AUTO_INCREMENT,
   `City` CHAR(50) NOT NULL,
   PRIMARY KEY (`id_City`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Вторым способом можно создать первичный ключ, состоящий из нескольких полей (нужно перечислить их в скобках через запятую):

```
PRIMARY KEY (`id_Orders`, `id_Tovar`)
```

Добавить первичный ключ в существующую таблицу позволяет SQL-команда:

```
ALTER TABLE <Таблица> ADD PRIMARY KEY (<Поле>);
```

Удалить первичный ключ позволяет SQL-команда:

```
ALTER TABLE <Таблица> DROP PRIMARY KEY;
```

В одной таблице не может быть более одного первичного ключа. А вот обычных и уникальных индексов в таблице может быть несколько. Создать индекс можно при определении структуры таблицы с помощью ключевых слов INDEX и KEY (UNIQUE INDEX и UNIQUE KEY для уникального индекса):

```
CREATE TABLE `Customers` (
   `id_Customer` INT NOT NULL AUTO_INCREMENT,
   `Name` CHAR(50) NOT NULL,
   `Address` CHAR(255) NOT NULL,
   `id_City` INT NOT NULL,
   `Phone` CHAR(30),
   PRIMARY KEY (`id_Customer`),
   KEY MyIndex (`Name`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Индекс может иметь название. Но поскольку название индекса не указывается в SQL-запросе, то чаще всего названием индекса служит имя поля. Сервер MySQL самостоятельно решает, каким индексом лучше воспользоваться в каждой конкретной ситуации. Знать название индекса необходимо для его удаления из таблицы.

При индексировании текстовых полей следует указать число символов (до 1000 символов), подлежащих индексации:

```
KEY MyIndex (Name(10))
```

> ***Примечание***
>
> **В большинстве случаев достаточно внести в индекс первые четыре или пять символов.**

Создать обычный индекс позволяют SQL-команды

```
CREATE INDEX <Имя индекса> ON <Таблица> (<Поле>(<Число символов>));
```

или

```
ALTER TABLE <Таблица>
ADD INDEX <Имя индекса> (<Поле>(<Число символов>));
```

Создать уникальный индекс можно с помощью SQL-команд

```
CREATE UNIQUE INDEX <Имя индекса>
ON <Таблица> (<Поле>(<Число символов>));
```

или

```
ALTER TABLE <Таблица>
ADD UNIQUE INDEX <Имя индекса> (<Поле>(<Число символов>));
```

Удалить обычный и уникальный индексы позволяют SQL-команды

```
DROP INDEX <Имя индекса> ON <Таблица>;
```

или

```
ALTER TABLE <Таблица> DROP INDEX <Имя индекса>;
```

В качестве примера создадим индекс для поля `Name` таблицы `Customers`:

```
CREATE INDEX `Name` ON `Customers` (`Name`(5));
```

А теперь сделаем запрос и проверим его эффективность с помощью оператора `EXPLAIN`:

```
EXPLAIN SELECT * FROM `Customers` WHERE `Name`='Иванов Иван Иванович';
```

Эта команда SQL выведет результат, показанный в табл. 6.13.

Таблица 6.13. *Результат, возвращенный командой* `EXPLAIN`*, после создания индекса*

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	Customers	ref	Name	Name	5	const	1	Using where

Сравните результат запроса с индексом и предыдущий пример без индекса.

> ***ОБРАТИТЕ ВНИМАНИЕ***
>
> Значение в столбце `type` уже не равно `ALL`, а число просмотренных записей равно 1. Это означает, что индекс полностью задействован.

Индекс `FULLTEXT` применяется для полнотекстового поиска. Реализацию полнотекстового поиска и способы создания индекса `FULLTEXT` мы подробно рассмотрим в *разд. 6.10*.

Получить полную информацию об индексах таблицы позволяет SQL-команда:

```
SHOW INDEX FROM <Таблица> [FROM <База данных>];
```

Например, посмотрим, какие индексы присутствуют в таблице `Customers` нашей базы данных, для чего наберем команду

```
SHOW INDEX FROM `Customers`\G
```

Результат также будет представлять собой таблицу, которую автор не может здесь привести из-за слишком большой ее ширины. Давайте "развернем" эту таблицу, превратив ее в список.

Первая строка таблицы:

```
Table: customers
Non_unique: 0
Key_name: PRIMARY
Seq_in_index: 1
Column_name: id_Customer
Collation: A
Cardinality: 3
Sub_part: NULL
Packed: NULL
Null:
Index_type: BTREE
Comment:
Index_comment:
```

Вторая строка таблицы:

```
Table: customers
Non_unique: 1
Key_name: Name
Seq_in_index: 1
Column_name: Name
Collation: A
Cardinality: NULL
Sub_part: 5
Packed: NULL
Null:
Index_type: BTREE
```

```
Comment:
Index_comment:
```

Рассмотрим столбцы получившейся таблицы и их назначение:

- `Table` — название таблицы;
- `Non_unique` — 0, если индекс может содержать лишь уникальные значения, 1 в противном случае;
- `Key_name` — название индекса; для первичного — `PRIMARY`;
- `Seq_in_index` — номер поля в индексе, начиная с единицы;
- `Column_name` — название поля;
- `Collation` — порядок сортировки значений: `A` — по возрастанию, `NULL` — без сортировки;
- `Cardinality` — число элементов в индексе;
- `Sub_part` — для строкового поля — число символов, включаемых в индекс; для строкового поля; для полей прочих типов — всегда `NULL`;
- `Packed` — способ упаковки индекса или `NULL`, если индекс не упакован;
- `Null` — `YES`, если индекс может включать значения `NULL`, и пустая строка в противном случае;
- `Index_type` — тип индекса: `BTREE` для обычного и `FULLTEXT` — для полнотекстового;
- `Comment` — дополнительные сведения об индексе, выдаваемые самим сервером;
- `Index_comment` — примечания к индексу, заданные его разработчиком.

Обратите внимание, в качестве значения строки `Cardinality` для индекса `Name` мы получили значение `NULL`. Может показаться, что в индексе нет элементов. Чтобы получить число элементов, необходимо перед использованием оператора `SHOW INDEX` выполнить SQL-команду:

```
ANALYZE TABLE <Таблица>;
```

6.4.11. Удаление таблицы и базы данных

Удалить таблицу позволяет SQL-команда:

```
DROP TABLE [IF EXISTS] <Имя таблицы>;
```

При попытке удалить несуществующую таблицу сервер выдаст сообщение об ошибке. Чтобы исключить такую ситуацию, следует вставить в упомянутую команду слова `IF EXISTS`.

Удалить всю базу данных можно с помощью SQL-команды

```
DROP DATABASE [IF EXISTS] <Имя базы данных>;
```

6.5. Доступ к базе данных из PHP с помощью библиотеки php_mysql.dll

Итак, изучение основ языка SQL закончено. Теперь мы рассмотрим встроенные функции PHP, которые позволяют получить доступ к базе данных из скрипта. В этом разделе мы рассмотрим возможности библиотеки php_mysql.dll, а в следующем разделе — возможности усовершенствованной библиотеки php_mysqli.dll. Чтобы можно было подключиться к MySQL из скрипта, необходимо в файле php.ini убрать символ комментария (;) перед строками:

```
extension=php_mysql.dll
extension=php_mysqli.dll
```

А также прописать путь к библиотекам в директиве `extension_dir`:

```
extension_dir = "C:/php5/ext"
```

> ***Внимание!***
>
> Начиная с PHP версий 5.5.*, расширение php_mysql.dll считается устаревшим, однако все еще нормально работает. Тем не менее, следует иметь в виду, что в последующих версиях PHP оно будет удалено.

6.5.1. Установка соединения

Для установки соединения используются две функции:

```
mysql_connect(<Имя хоста>, <Имя пользователя>, <Пароль>);
mysql_pconnect(<Имя хоста>, <Имя пользователя>, <Пароль>);
```

Функции возвращают идентификатор соединения, а в случае неудачи возвращают `false`. Вся дальнейшая работа с базой данных осуществляется через этот идентификатор.

Функция `mysql_connect()` устанавливает обычное соединение с сервером MySQL. Обычное соединение закрывается, когда сценарий завершает работу или когда вызывается функция `mysql_close()`:

```
mysql_close(<Идентификатор>);
```

Функция `mysql_pconnect()` устанавливает постоянное соединение с сервером MySQL. При вызове функция проверяет наличие уже открытого постоянного соединения. Если соединение существует, функция использует это соединение, а не открывает новое. По завершению работы сценария постоянное соединение не закрывается.

Для того чтобы подключиться к серверу MySQL, можно воспользоваться следующим кодом:

```
<?php
$db = @mysql_connect("localhost", "root", "123456");
if ($db) {
```

```
   // Выполняем работу с базой данных
   mysql_close($db); // Закрываем соединение
}
else {
   echo "Не удалось установить подключение к базе данных";
}
?>
```

6.5.2. Выбор базы данных

Для выбора базы данных предусмотрена функция `mysql_select_db()`. Формат функции:

```
mysql_select_db(<Имя базы данных>, [<Идентификатор>]);
```

Параметр `<Идентификатор>` можно не указывать. По умолчанию будет использоваться последнее открытое соединение.

Для подключения к базе tests можно воспользоваться следующим PHP-кодом:

```
<?php
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   // Выполняем работу с базой данных
   mysql_close($db); // Закрываем соединение
}
else {
   echo "Не удалось установить подключение к базе данных";
}
?>
```

6.5.3. Выполнение запроса к базе данных

Выполнить запрос к базе данных позволяет функция `mysql_query()`. Формат функции:

```
mysql_query(<SQL-запрос>, [<Идентификатор>]);
```

> ***Внимание!***
> SQL-запрос не требует указания в конце точки с запятой.

Функция возвращает идентификатор результата. Параметр `<Идентификатор>` можно не указывать. По умолчанию будет использоваться последнее открытое соединение.

Получить все записи таблицы `City` позволяет следующий код:

```
<?php
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   $res = mysql_query('SELECT * FROM `City`');
```

```
   // Обрабатываем полученный результат
   mysql_close($db); // Закрываем соединение
}
else {
   echo "Не удалось установить подключение к базе данных";
}
?>
```

Для того чтобы записи возвращались в нужной кодировке, следует после выбора базы данных указать один из запросов:

```
mysql_query('SET NAMES cp1251'); // Для кодировки Windows-1251
mysql_query('SET NAMES utf8'); // Для кодировки UTF-8
```

6.5.4. Обработка результата запроса

Для обработки результата запроса используются следующие функции:

- `mysql_num_rows(<Идентификатор результата>)` — возвращает число записей в результате:

```
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   $res = mysql_query('SELECT * FROM `City`');
   echo mysql_num_rows($res);
   // Выведет: 2
   mysql_close($db); // Закрываем соединение
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysql_num_fields(<Идентификатор результата>)` — возвращает число полей в результате:

```
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   $res = mysql_query('SELECT * FROM `City`');
   echo mysql_num_fields($res);
   // Выведет: 2
   mysql_close($db); // Закрываем соединение
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysql_result()` — позволяет получить доступ к отдельному полю по указанному номеру строки. Поле может быть обозначено его названием или порядковым номером, начиная с нуля; нумерация строк начинается с пуля:

```
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   mysql_query("SET NAMES cp1251");
   $res = mysql_query('SELECT * FROM `City`');
   $count = mysql_num_rows($res);
   for ($i=0; $i<$count; $i++) {
      $id = mysql_result($res, $i, "id_City");
      $city = mysql_result($res, $i, "City");
      echo "$id - $city<br>";
   }
   mysql_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

Этот код выведет

```
1 - Санкт-Петербург
2 - Москва
```

- `mysql_fetch_array(<Идентификатор результата>, [<Флаг>])` — возвращает результат в виде списка и ассоциативного массива и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`. Тип возвращенного массива зависит от необязательного параметра `<Флаг>`, который может принимать следующие значения:
  - `MYSQL_BOTH` — результат в виде списка и ассоциативного массива (значение по умолчанию);
  - `MYSQL_NUM` — результат в виде списка;
  - `MYSQL_ASSOC` — результат в виде ассоциативного массива.

Пример:

```
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   mysql_query("SET NAMES cp1251");
   $res = mysql_query('SELECT * FROM `City`');
   while ($pole = mysql_fetch_array($res)) {
      echo $pole['id_City'] . ' - ' . $pole['City'] . '<br>';
   }
   mysql_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

или

```
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
```

```
    mysql_query("SET NAMES cp1251");
    $res = mysql_query('SELECT * FROM `City`');
    while ($pole = mysql_fetch_array($res)) {
       echo $pole[0] . ' - ' . $pole[1] . '<br>';
    }
    mysql_close($db);
}
else {
    echo "Не удалось установить подключение к базе данных";
}
```

- `mysql_fetch_row(<Идентификатор результата>)` — возвращает результат в виде списка и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`. Вызов этой функции эквивалентен вызову `mysql_fetch_array` с флагом `MYSQL_NUM`:

```
if ($db = @mysql_connect("localhost", "root", "123456")) {
    mysql_select_db("tests");
    mysql_query("SET NAMES cp1251");
    $res = mysql_query('SELECT * FROM `City`');
    while ($pole = mysql_fetch_row($res)) {
       echo $pole[0] . ' - ' . $pole[1] . '<br>';
    }
    mysql_close($db);
}
else {
    echo "Не удалось установить подключение к базе данных";
}
```

- `mysql_fetch_assoc(<Идентификатор результата>)` — возвращает результат в виде ассоциативного массива и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`. Вызов этой функции эквивалентен вызову `mysql_fetch_array` с флагом `MYSQL_ASSOC`:

```
if ($db = @mysql_connect("localhost", "root", "123456")) {
    mysql_select_db("tests");
    mysql_query("SET NAMES cp1251");
    $res = mysql_query('SELECT * FROM `City`');
    while ($pole = mysql_fetch_assoc($res)) {
       echo $pole['id_City'] . ' - ' . $pole['City'] . '<br>';
    }
    mysql_close($db);
}
else {
    echo "Не удалось установить подключение к базе данных";
}
```

- `mysql_fetch_object(<Идентификатор результата>)` — возвращает результат в виде объекта и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`:

```
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   mysql_query("SET NAMES cp1251");
   $res = mysql_query('SELECT * FROM `City`');
   while ($pole = mysql_fetch_object($res)) {
      echo $pole->id_City . ' - ' . $pole->City . '<br>';
   }
   mysql_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysql_real_escape_string(<Строка>, [<Идентификатор соединения>])` — экранирует все специальные символы в строке, учитывая кодировку соединения. Возвращает строку, которую можно безопасно использовать в SQL-запросах.

> ***Обратите внимание***
>
> Функцию можно использовать только после подключения к базе данных. В противном случае получите сообщение об ошибке.

Пример:

```
$new_city = "Д'Арк";
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   mysql_query("SET NAMES cp1251");
   // Экранируем спецсимволы
   $new_city = mysql_real_escape_string($new_city);
   mysql_query("INSERT INTO `City` VALUES(NULL, '$new_city')");
   $res = mysql_query('SELECT * FROM `City`');
   while ($pole = mysql_fetch_object($res)) {
      echo $pole->id_City . ' - ' . $pole->City . '<br>';
   }
   mysql_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

> ***Внимание!***
>
> Никогда напрямую не передавайте в SQL-запрос данные, полученные из полей формы. Это потенциальная угроза безопасности. Всегда применяйте функцию `mysql_real_escape_string()`.

В следующем примере рассмотрим проблему, возникающую, если не применить функцию `mysql_real_escape_string()` для входных данных.

Создадим таблицу `user` и добавим в нее две записи:

```
CREATE TABLE `user` (
  `id_user` MEDIUMINT(9) AUTO_INCREMENT,
  `login` CHAR(50),
  `passw` CHAR(32),
  PRIMARY KEY (`id_user`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
INSERT INTO `user` VALUES (NULL, 'Admin', '123');
INSERT INTO `user` VALUES (NULL, 'Nik', '456');
```

Теперь инсценируем вход злоумышленника в систему под логином администратора. При этом злоумышленник даже не должен знать его пароль.

```
<?php
// Никогда так не делайте!!!
// Такие данные пришли из формы:
$_POST['login'] = "' OR ''='";
$_POST['passw'] = "' OR ''='";
$login = $_POST['login'];
$passw = $_POST['passw'];
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   mysql_query("SET NAMES cp1251");
   $query = "SELECT * FROM `user` WHERE `login`='$login' ";
   $query .= "AND `passw`='$passw'";
   echo $query . '<br>';
   $res = mysql_query($query);
   if (mysql_num_rows($res) > 0) {
      echo 'Полный доступ в систему!!!<br>';
   }
   while ($pole = mysql_fetch_object($res)) {
      echo $pole->login . '<br>';
   }
   mysql_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
?>
```

Введя указанные в начале примера строки в форме, злоумышленник получит

```
SELECT * FROM `user` WHERE `login`='' OR ''='' AND `passw`='' OR ''=''
Полный доступ в систему!!!
Admin
Nik
```

Итак, злоумышленник вошел в систему, не зная пароля. В данном примере, т. к. учетная запись администратора расположена на первой позиции, мы вошли под записью администратора. Нам просто повезло. А теперь войдем в систему именно под учетной записью администратора. Для этого входящие данные изменим на

```
$_POST['login'] = "Admin'/*";
$_POST['passw'] = "*/ '";
```

После выполнения скрипта получим следующий результат:

```
SELECT * FROM `user` WHERE `login`='Admin'/*' AND `passw`='*/ ''
Полный доступ в систему!!!
Admin
```

Как видно из результата, мы являемся администратором. Все, что расположено между `/*` и `*/`, это комментарий. В итоге SQL-запрос будет выглядеть так:

```
SELECT * FROM `user` WHERE `login`='Admin' ''
```

Пароль в данном случае вообще не проверяется. Достаточно знать логин пользователя и можно войти без пароля.

При обработке данных функцией `mysql_real_escape_string()` такого бы не случилось:

```
$login = mysql_real_escape_string($login);
$passw = mysql_real_escape_string($passw);
$query = "SELECT * FROM `user` WHERE `login`='$login' ";
$query .= "AND `passw`='$passw'";
echo $query . '<br>';
```

В первом случае скрипт выведет только

```
SELECT * FROM user WHERE login='\' OR \'\'=\'' AND passw='\' OR \'\'=\''
```

А во втором

```
SELECT * FROM `user` WHERE `login`='Admin\'/*' AND `passw`='*/ \''
```

В результате все опасные символы были экранированы.

> ***Примечание***
>
> Выводить код SQL напрямую в Web-страницу также не рекомендуется, т. к. это дает злоумышленнику лишнюю информацию о структуре базы данных.

6.6. Доступ к базе данных из PHP с помощью библиотеки php_mysqli.dll

Библиотека php_mysqli.dll предоставляет более современные методы доступа к базе данных MySQL. Она позволяет использовать как процедурный стиль доступа, так и объектный. В этом разделе мы рассмотрим оба стиля.

6.6.1. Установка соединения

Установить соединение можно двумя способами:

```
$db = mysqli_connect(<Имя хоста>, <Имя пользователя>, <Пароль>,
     <База данных>);
$db = new mysqli(<Имя хоста>, <Имя пользователя>, <Пароль>,
     <База данных>);
```

Все параметры необязательные. Процедурный стиль возвращает идентификатор соединения, а в случае неудачи возвращает `false`. Проверить соединение можно следующим образом:

```
if (@$db = mysqli_connect("localhost", "root", "123456", "tests")) {
   // Выполняем работу с базой данных
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

При объектном стиле такой способ не подходит. Проверить отсутствие ошибок при подключении позволяет функция `mysqli_connect_errno()`. Проверить соединение можно следующим образом:

```
@$db = new mysqli("localhost", "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   // Выполняем работу с базой данных
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

Также можно обратиться к свойству `connect_errno`:

```
@$db = new mysqli("localhost", "root", "123456", "tests");
if (!$db->connect_errno) {
   // Выполняем работу с базой данных
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

Закрыть соединение при процедурном стиле позволяет функция `mysqli_close()`:

```
mysqli_close(<Идентификатор>);
```

При объектном стиле используется метод `close()`:

```
<Экземпляр класса>->close();
```

Приведем код для подключения к серверу MySQL. Процедурный стиль:

```
if (@$db = mysqli_connect("localhost", "root", "123456", "tests")) {
   // Выполняем работу с базой данных
```

```
   mysqli_close($db); // Закрываем соединение
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

Объектный стиль:

```
@$db = new mysqli("localhost", "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   // Выполняем работу с базой данных
   $db->close(); // Закрываем соединение
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

6.6.2. Выбор базы данных

Выбрать базу данных можно при подключении в функции `mysqli_connect()` или в конструкторе класса. При процедурном стиле выбор базы данных уже после подключения осуществляет функция `mysqli_select_db()`. Формат функции:

```
mysqli_select_db(<Идентификатор>, <Имя базы данных>);
```

Пример:

```
if (@$db = mysqli_connect("localhost", "root", "123456")) {
   mysqli_select_db($db, "tests");
   // Выполняем работу с базой данных
   mysqli_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

При объектном стиле используется метод `select_db()`. Формат метода:

```
<Экземпляр класса>->select_db(<Имя базы данных>);
```

Пример:

```
@$db = new mysqli("localhost", "root", "123456");
if (!mysqli_connect_errno()) {
   $db->select_db("tests");
   // Выполняем работу с базой данных
   $db->close();
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

6.6.3. Выполнение запроса к базе данных

Выполнить запрос к базе данных при процедурном стиле позволяет функция `mysqli_query()`. Формат функции:

```
mysqli_query(<Идентификатор>, <SQL-запрос>);
```

> ***Внимание!***
> В конце SQL-запроса не следует указывать точку с запятой.

Функция возвращает идентификатор результата. Для удаления идентификатора результата и освобождения используемых ресурсов применяется функция `mysqli_free_result()`. Формат функции:

```
mysqli_free_result(<Идентификатор результата>);
```

Получить все записи таблицы `City` позволяет следующий код:

```
if (@$db = mysqli_connect("localhost", "root", "123456", "tests")) {
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      // Обрабатываем извлеченные записи
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

Выполнить запрос к базе данных при объектном стиле позволяет метод `query()`. Формат метода:

```
<Экземпляр класса>->query(<SQL-запрос>);
```

Метод возвращает экземпляр результата. Для удаления экземпляра результата следует применить метод `close()` или `free()`. Формат метода:

```
<Экземпляр результата>->close();
```

Получить все записи таблицы `City` позволяет следующий код:

```
@$db = new mysqli("localhost", "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   if ($res = $db->query('SELECT * FROM `City`')) {
      // Обрабатываем извлеченные записи
      $res->close();
   }
   $db->close();
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

Для того чтобы записи возвращались в нужной кодировке, следует после подключения выполнить запрос

```
mysqli_query($db, 'SET NAMES cp1251'); // Для кодировки windows-1251
mysqli_query($db, 'SET NAMES utf8'); // Для кодировки UTF-8
```

при процедурном стиле или

```
$db->query('SET NAMES cp1251'); // Для кодировки windows-1251
$db->query('SET NAMES utf8'); // Для кодировки UTF-8
```

при объектном стиле.

6.6.4. Обработка результата запроса

Для обработки результата запроса при процедурном стиле используются следующие функции:

- `mysqli_num_rows(<Идентификатор результата>)` — возвращает число записей в результате:

```
$host = "localhost";
if (@$db = mysqli_connect($host, "root", "123456", "tests")) {
   mysqli_query($db, "SET NAMES cp1251");
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      echo mysqli_num_rows($res) . "<br>";
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysqli_field_count(<Идентификатор соединения>)` — возвращает число полей в результате последнего SQL-запроса:

```
$host = "localhost";
if (@$db = mysqli_connect($host, "root", "123456", "tests")) {
   mysqli_query($db, "SET NAMES cp1251");
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      echo mysqli_field_count($db) . "<br>";
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysqli_fetch_array(<Идентификатор результата>, [<Флаг>])` — возвращает результат в виде списка и (или) ассоциативного массива и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`. Тип возвращенного результата зависит от необязательного параметра `<Флаг>`, который может принимать следующие значения:
  - `MYSQLI_BOTH` — результат в виде списка и ассоциативного массива (значение по умолчанию);
  - `MYSQLI_NUM` — результат в виде списка;
  - `MYSQLI_ASSOC` — результат в виде ассоциативного массива.

  Приведем пример, в котором сочетаются все эти варианты:

```
$host = "localhost";
if (@$db = mysqli_connect($host, "root", "123456", "tests")) {
   mysqli_query($db, "SET NAMES cp1251");
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      while ($pole = mysqli_fetch_array($res)) {
         echo $pole[0] . ' - ' . $pole['City'] . '<br>';
      }
      mysqli_free_result($res);
   }
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      $pole = mysqli_fetch_array($res, MYSQLI_BOTH);
      echo $pole[0] . ' - ' . $pole['City'] . '<br>';

      $pole = mysqli_fetch_array($res, MYSQLI_NUM);
      echo $pole[0] . ' - ' . $pole[1] . '<br>';

      $pole = mysqli_fetch_array($res, MYSQLI_ASSOC);
      echo $pole['id_City'] . ' - ' . $pole['City'] . '<br>';
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysqli_fetch_row(<Идентификатор результата>)` — возвращает результат в виде списка и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`. Вызов этой функции эквивалентен вызову `mysqli_fetch_array` с флагом `MYSQLI_NUM`:

```
$host = "localhost";
if (@$db = mysqli_connect($host, "root", "123456", "tests")) {
   mysqli_query($db, "SET NAMES cp1251");
```

```
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      while ($pole = mysqli_fetch_row($res)) {
         echo $pole[0] . ' - ' . $pole[1] . '<br>';
      }
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysqli_fetch_assoc(<Идентификатор результата>)` — возвращает результат в виде ассоциативного массива и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`. Вызов этой функции эквивалентен вызову `mysqli_fetch_array` с флагом `MYSQLI_ASSOC`:

```
$host = "localhost";
if (@$db = mysqli_connect($host, "root", "123456", "tests")) {
   mysqli_query($db, "SET NAMES cp1251");
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      while ($pole = mysqli_fetch_assoc($res)) {
         echo $pole['id_City'] . ' - ' . $pole['City'] . '<br>';
      }
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysqli_fetch_object(<Идентификатор результата>)` — возвращает результат в виде объекта и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`:

```
$host = "localhost";
if (@$db = mysqli_connect($host, "root", "123456", "tests")) {
   mysqli_query($db, "SET NAMES cp1251");
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      while ($pole = mysqli_fetch_object($res)) {
         echo $pole->id_City . ' - ' . $pole->City . '<br>';
      }
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
```

```
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysqli_data_seek(<Идентификатор результата>, <Смещение>)` — перемещает указатель результата на выбранную строку. Нумерация строк начинается с нуля:

```
$host = "localhost";
if (@$db = mysqli_connect($host, "root", "123456", "tests")) {
   mysqli_query($db, "SET NAMES cp1251");
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      mysqli_data_seek($res, 1);
      $pole = mysqli_fetch_object($res);
      echo $pole->id_City . ' - ' . $pole->City . '<br>';
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

Для обработки результата запроса при объектном стиле используются следующие методы и свойства:

- `num_rows` — возвращает число записей в результате:

```
$host = "localhost";
@$db = new mysqli($host, "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   $db->query("SET NAMES cp1251");
   if ($res = $db->query('SELECT * FROM `City`')) {
      echo $res->num_rows . '<br>';
      $res->close();
   }
   $db->close();
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `field_count` — возвращает число полей в результате:

```
$host = "localhost";
@$db = new mysqli($host, "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   $db->query("SET NAMES cp1251");
   if ($res = $db->query('SELECT * FROM `City`')) {
      echo $res->field_count . '<br>';
```

```
      $res->close();
   }
   $db->close();
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `fetch_array([<Флаг>])` — возвращает результат в виде списка и (или) ассоциативного массива и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`. Тип возвращенного результата зависит от необязательного параметра `<Флаг>`, который может принимать следующие значения:
  - `MYSQLI_BOTH` — результат в виде списка и ассоциативного массива (значение по умолчанию);
  - `MYSQLI_NUM` — результат в виде списка;
  - `MYSQLI_ASSOC` — результат в виде ассоциативного массива.

  Следующий пример иллюстрирует все эти варианты:

```
$host = "localhost";
@$db = new mysqli($host, "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   $db->query("SET NAMES cp1251");
   if ($res = $db->query('SELECT * FROM `City`')) {
      while ($pole = $res->fetch_array()) {
         echo $pole[0] . ' - ' . $pole['City'] . '<br>';
      }
      $res->close();
   }
   if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
      $pole = $res->fetch_array(MYSQLI_BOTH);
      echo $pole[0] . ' - ' . $pole['City'] . '<br>';

      $pole = $res->fetch_array(MYSQLI_NUM);
      echo $pole[0] . ' - ' . $pole[1] . '<br>';

      $pole = $res->fetch_array(MYSQLI_ASSOC);
      echo $pole['id_City'] . ' - ' . $pole['City'] . '<br>';
      $res->close();
   }
   $db->close();
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `fetch_row()` — возвращает результат в виде списка и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`. Вызов этого метода эквивалентен вызову `fetch_array` с флагом `MYSQLI_NUM`:

```
$host = "localhost";
@$db = new mysqli($host, "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   $db->query("SET NAMES cp1251");
   if ($res = $db->query('SELECT * FROM `City`')) {
      while ($pole = $res->fetch_row()) {
         echo $pole[0] . ' - ' . $pole[1] . '<br>';
      }
      $res->close();
   }
   $db->close();
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `fetch_assoc()` — возвращает результат в виде ассоциативного массива и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`. Вызов этого метода эквивалентен вызову `fetch_array` с флагом `MYSQLI_ASSOC`:

```
$host = "localhost";
@$db = new mysqli($host, "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   $db->query("SET NAMES cp1251");
   if ($res = $db->query('SELECT * FROM `City`')) {
      while ($pole = $res->fetch_assoc()) {
         echo $pole['id_City'] . ' - ' . $pole['City'] . '<br>';
      }
      $res->close();
   }
   $db->close();
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `fetch_object()` — возвращает результат в виде объекта и сдвигает внутренний указатель на следующую запись; если записей больше нет, возвращается `false`:

```
$host = "localhost";
@$db = new mysqli($host, "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   $db->query("SET NAMES cp1251");
```

```
    if ($res = $db->query('SELECT * FROM `City`')) {
       while ($pole = $res->fetch_object()) {
          echo $pole->id_City . ' - ' . $pole->City . '<br>';
       }
       $res->close();
    }
    $db->close();
}
else {
    echo "Не удалось установить подключение к базе данных";
}
```

- `data_seek(<Смещение>)` — перемещает указатель результата на выбранную строку. Нумерация строк начинается с нуля:

```
$host = "localhost";
@$db = new mysqli($host, "root", "123456", "tests");
if (!mysqli_connect_errno()) {
    $db->query("SET NAMES cp1251");
    if ($res = $db->query('SELECT * FROM `City`')) {
       $res->data_seek(1);
       $pole = $res->fetch_object();
       echo $pole->id_City . ' - ' . $pole->City . '<br>';
       $res->close();
    }
    $db->close();
}
else {
    echo "Не удалось установить подключение к базе данных";
}
```

Функция `mysqli_real_escape_string(<Идентификатор соединения>, <Строка>)` в процедурном стиле и метод `real_escape_string(<Строка>)` в объектном стиле экранируют все специальные символы в строке, учитывая кодировку соединения. Возвращают строку, которую можно безопасно использовать в SQL-запросах:

```
$new_city = "Д'Арк"; // Такие данные передаются из формы
$host = "localhost";
if (@$db = mysqli_connect($host, "root", "123456", "tests")) {
    mysqli_query($db, "SET NAMES cp1251");
    // Экранируем спецсимволы
    $new_city = mysqli_real_escape_string($db, $new_city);
    $query = "INSERT INTO `City` VALUES(NULL, '$new_city')";
    mysqli_query($db, $query);
    if ($res = mysqli_query($db, 'SELECT * FROM `City`')) {
       while ($pole = mysqli_fetch_row($res)) {
          echo $pole[0] . ' - ' . $pole[1] . '<br>';
       }
```

```
        mysqli_free_result($res);
    }
    mysqli_close($db);
}
else {
    echo 'Не удалось установить подключение к базе данных';
}
```

Внимание!

Никогда напрямую не передавайте в SQL-запрос данные, полученные из полей формы. Это потенциальная угроза безопасности. Всегда применяйте функцию `mysqli_real_escape_string()` или метод `real_escape_string()`.

6.7. Операторы MySQL

Операторы позволяют выполнить определенные действия с данными. Например, математические операторы предназначены для арифметических вычислений. Рассмотрим операторы, доступные в MySQL.

Примечание

Выполнять SQL-команды мы будем в программе MySQL Command Line Client. Для запуска программы в меню **Пуск** выбираем пункт **Программы | MySQL | MySQL Server 5.5 | MySQL 5.5 Command Line Client**. Откроется черное окошко с запросом ввести пароль. Вводим пароль, заданный при установке сервера MySQL. Если установка проводилась по инструкциям из *разд. 4.7*, то пароль "123456". В случае успешного входа отобразится приветствие сервера, и программа перейдет в режим ожидания команд. В командной строке набираем `USE tests;` для выбора базы данных.

6.7.1. Математические операторы

Математические операторы:

- `+` — сложение:

```
SELECT 8 + 5;
```

- `-` — вычитание:

```
SELECT 10 - 5;
```

- `*` — умножение:

```
SELECT 10 * 5;
```

- `/` — деление:

```
SELECT 10 / 5;
/* Выведет: 2.0000 */
```

- `DIV` — целочисленное деление:

```
SELECT 10 DIV 5;
/* Выведет: 2 */
```

```
SELECT 10 DIV 3;
/* Выведет: 3 */
```

- `%` и `MOD` — остаток от деления:

```
SELECT 10 % 2;
/* Выведет: 0 */
SELECT 9 % 2;
/* Выведет: 1 */
SELECT 10 MOD 2;
/* Выведет: 0 */
```

Вместо операторов `%` и `MOD` можно использовать функцию `MOD()`:

```
SELECT MOD(10, 2);
/* Выведет: 0 */
```

Следует отметить, что если один из операндов равен `NULL`, то результат операции также будет равен `NULL`. В отличие от языков программирования деление на ноль не приводит к генерации сообщения об ошибке. Результатом операции деления на ноль является значение `NULL`.

Если необходимо сменить знак числа, то перед операндом следует указать символ - (минус):

```
SELECT -(-5);
/* Выведет: 5 */
```

В качестве примера рассмотрим возможность подсчета переходов по рекламной ссылке. Для этого создадим таблицу `counter` в базе данных tests:

```
CREATE TABLE `counter` (
   `id_link` INT NOT NULL AUTO_INCREMENT,
   `total` INT,
   PRIMARY KEY (`id_link`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим одну запись:

```
INSERT INTO `counter` VALUES (1, 0);
```

Для подсчета переходов в тексте ссылки укажем идентификатор в базе данных и URL-адрес:

```
<html>
<head>
<title>Подсчет переходов по ссылкам</title>
</head>
<body>
<a href="go.php?id=1&url=http://www.mail.ru/">Перейти</a>
</body>
</html>
```

Переходы регистрируются в файле go.php. Исходный код файла приведен в листинге 6.1.

Листинг 6.1. Регистрация переходов по ссылке

```
<?php
if (!isset($_GET['id']) || !isset($_GET['url'])) die('Ошибка');
$id = $_GET['id'];
if (preg_match('/^[0-9]+$/s', $id) && $id != 0) {
  if ($db = @mysql_connect("localhost", "root", "123456")) {
    mysql_select_db("tests");
    $query = 'UPDATE `counter` SET `total` = `total` + 1 ';
    $query .= 'WHERE `id_link`=' . $id;
    @mysql_query($query);
    mysql_close($db);
  }
}
header('Location: ' . $_GET['url']);
?>
```

Оператор + позволяет увеличить счетчик за один запрос. Иначе пришлось бы вначале получить значение из базы данных, затем увеличить его и только во втором запросе обновить значение в базе данных.

6.7.2. Двоичные операторы

Двоичные операторы:

- ~ — двоичная инверсия;
- & — двоичное И;
- | — двоичное ИЛИ;
- ^ — двоичное исключающее ИЛИ;
- << — сдвиг влево на один или более разрядов с заполнением младших разрядов нулями;
- >> — сдвиг вправо на один или более разрядов с заполнением старших разрядов содержимым самого старшего разряда.

6.7.3. Операторы сравнения

Операторы сравнения используются, прежде всего, в конструкциях `WHERE` и `HAVING` при создании запросов. Перечислим их:

- = — равно;
- <=> — эквивалентно;
- != — не равно;
- <> — не равно;
- < — меньше;

- `>` — больше;
- `<=` — меньше или равно;
- `>=` — больше или равно;
- `IS NOT NULL` — проверка на наличие значения;
- `IS NULL` — проверка поля на отсутствие значения;
- `BETWEEN <Начало> AND <Конец>` — проверяет, является ли значение большим или равным `<Начало>` и меньшим или равным `<Конец>`, например, `pole BETWEEN 0 AND 100`;
- `IN` — содержится в определенном наборе, например, `pole IN ('HDD', 'Монитор')`;
- `NOT IN` — не содержится в определенном наборе, например, `pole NOT IN ('HDD','Монитор')`;
- `LIKE` — соответствие шаблону SQL;
- `NOT LIKE` — несоответствие шаблону SQL;
- `RLIKE` — соответствие регулярному выражению;
- `REGEXP` — соответствие регулярному выражению (синоним `RLIKE`);
- `NOT RLIKE` — несоответствие регулярному выражению;
- `NOT REGEXP` — несоответствие регулярному выражению (синоним `NOT RLIKE`).

В шаблоне SQL могут использоваться следующие символы:

- `%` — любое число символов;
- `_` — любой одиночный символ.

Можно проверять сразу несколько условий, указав логические операции:

- `AND` или `&&` — логическое И;
- `OR` или `||` — логическое ИЛИ;
- `XOR` — исключающее логическое ИЛИ;
- `NOT` или `!` — логическое НЕ.

Результаты операции сравнения:

- `0` — ложь;
- `1` — истина;
- `NULL` — возвращается, если хотя бы один из операндов равен `NULL`.

Исключением является оператор эквивалентности `<=>`. Он возвращает только два значения: `0` (ложь) и `1` (истина). Этот оператор введен специально для сравнения значения `NULL`.

Следует отметить, что по умолчанию сравнение строк происходит без учета регистра. Если указать ключевое слово `BINARY`, то регистр символов будет учитываться:

```
SELECT 'TEXT'='text';
/* Выведет: 1 (истина) */
SELECT BINARY 'TEXT'='text';
/* Выведет: 0 (ложь) */
```

Результат сравнения можно изменить на противоположный с помощью операторов ! и NOT.

```
SELECT 'TEXT'='text';
/* Выведет: 1 (истина) */
SELECT !('TEXT'='text');
/* Выведет: 0 (ложь) */
SELECT NOT ('TEXT'='text');
/* Выведет: 0 (ложь) */
```

Логические выражения следует заключать в круглые скобки, т. к. приоритет оператора отрицания выше приоритета других операторов.

6.7.4. Приоритет выполнения операторов

При составлении выражений следует учитывать приоритет выполнения операторов.

Перечислим операторы в порядке убывания приоритета:

1. BINARY, COLLATE.
2. NOT, !.
3. - (унарный минус), ~.
4. *, /, %, MOD, DIV.
5. +, - — сложение, вычитание.
6. <<, >> — двоичные сдвиги.
7. & — двоичное И.
8. | — двоичное ИЛИ.
9. =, <=>, >=, <=, >, <, <>, !=, IS, LIKE, REGEXP, IN.
10. BETWEEN.
11. &&, AND.
12. ||, OR, XOR.

С помощью круглых скобок можно изменить последовательность выполнения выражения:

```
SELECT 5 + 3 * 7;
/* Выведет: 26 */
SELECT (5 + 3) * 7;
/* Выведет: 56 */
```

6.7.5. Преобразование типов данных

В большинстве случаев преобразование типов осуществляется автоматически. В этом разделе мы рассмотрим результаты автоматического преобразования типов, а также встроенные функции для специального приведения типов.

Что будет, если к числу прибавить строку?

```
SELECT '5' + 3;
/* Выведет: 8 */
SELECT '5st' + 3;
/* Выведет: 8 */
```

В этом случае строка преобразуется в число, а затем выполняется операция сложения. Но что будет, если строку невозможно преобразовать в число?

```
SELECT 'str' + 3;
/* Выведет: 3 */
SELECT 3 + 'str';
/* Выведет: 3 */
```

Если строку невозможно преобразовать в число, то она приравнивается к нулю.

Для явного преобразования типов предназначены две функции:

- `CAST(<Выражение> AS <Тип>)`;
- `CONVERT(<Выражение>, <Тип>)`.

Параметр `<Тип>` может принимать следующие значения:

- `BINARY`;
- `CHAR`;
- `DATE`;
- `DATETIME`;
- `DECIMAL`;
- `SIGNED [INTEGER]`;
- `TIME`;
- `UNSIGNED [INTEGER]`.

6.8. Поиск по шаблону

Для поиска по шаблону предусмотрены два оператора:

- `LIKE` — соответствие шаблону SQL;
- `NOT LIKE` — несоответствие шаблону SQL.

В шаблоне SQL могут присутствовать следующие специальные символы:

- `%` — любое число символов;
- `_` — любой одиночный символ.

Если специальные символы не используются, то применение оператора `LIKE` эквивалентно оператору `=`, например:

```
SELECT 'строка для поиска' LIKE 'поиск';
/* Выведет: 0 */
SELECT 'поиск' LIKE 'поиск';
/* Выведет: 1 */
```

Следует помнить, что при поиске в текстовом поле регистр не учитывается. Чтобы учитывался регистр, необходимо указать ключевое слово `BINARY`:

```
SELECT 'PHP' LIKE 'php';
/* Выведет: 1 */
SELECT 'PHP' LIKE BINARY 'php';
/* Выведет: 0 */
```

Учет регистра русских букв зависит от кодировки:

```
SET NAMES cp866;
SELECT 'Поиск' LIKE 'поиск';
/* Выведет: 1 */
SET NAMES cp1251;
SELECT 'Поиск' LIKE 'поиск';
/* Выведет: 0 */
```

Специальные символы могут быть расположены в любом месте шаблона. Например, чтобы найти все вхождения, необходимо указать символ `%` в начале и в конце шаблона:

```
SELECT 'строка для поиска' LIKE '%поиск%';
/* Выведет: 1 */
```

Можно установить привязку или только к началу строки, или только к концу:

```
SELECT 'строка для поиска' LIKE 'строка%';
/* Выведет: 1 */
SELECT 'новая строка для поиска' LIKE 'строка%';
/* Выведет: 0 */
SELECT 'строка для поиска' LIKE '%поиска';
/* Выведет: 1 */
SELECT 'строка для поиска 2' LIKE '%поиска';
/* Выведет: 0 */
```

Шаблон для поиска может иметь очень сложную структуру:

```
SELECT 'строка для поиска' LIKE '%строк_%по_ск%';
/* Выведет: 1 */
SELECT 'строка для поиска' LIKE '%поиск%а';
/* Выведет: 1 */
```

Обратите внимание на последнюю строку поиска. Этот пример демонстрирует, что специальный символ `%` соответствует не только любому числу символов, но и полному их отсутствию.

Что же делать, если необходимо найти символы "%" и "_"? Ведь они являются специальными:

```
SELECT 'скидка 10%' LIKE '%10%';
/* Выведет: 1 */
SELECT 'скидка 10$' LIKE '%10%';
/* Выведет: 1 */
```

В этом случае специальные символы необходимо экранировать с помощью обратной косой черты:

```
SELECT 'скидка 10%' LIKE '%10\%';
/* Выведет: 1 */
SELECT 'скидка 10$' LIKE '%10\%';
/* Выведет: 0 */
```

Обратите внимание, что функция `mysql_real_escape_string()` не добавляет обратную косую черту перед символами `%` и `_` для их защиты. Сделать это в PHP позволяет функция `addcslashes()`:

```
$text = addcslashes($text, '_%');
```

В качестве примера рассмотрим поиск по шаблону. Для этого создадим таблицу `search` в базе данных tests:

```
CREATE TABLE `search` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `str` TEXT,
   PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим две записи:

```
SET NAMES cp866;
INSERT INTO `search` VALUES (NULL, 'Скидка 10%');
INSERT INTO `search` VALUES (NULL, 'Скидка 10$');
```

Исходный код с вариантами поиска по шаблону приведен в листинге 6.2.

Листинг 6.2. Поиск по шаблону

```
<?php
$str_search = '10%';
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db("tests");
   mysql_query("SET NAMES cp1251");
   // Добавляем защитные слэши
   $str_search = mysql_real_escape_string($str_search);

   echo 'Без добавления слэшей перед спецсимволами:<br>';
   $query = "SELECT `str` FROM `search` WHERE `str` LIKE '%";
```

```
    $query .= $str_search . "%'";
    $res = mysql_query($query);
    while ($pole = mysql_fetch_array($res)) {
        echo $pole[0] . '<br>';
    }

    echo '<br>После добавления слэшей перед спецсимволами:<br>';
    $str_search_add = addcslashes($str_search, '_%');
    $query_add = "SELECT `str` FROM `search` WHERE `str` LIKE '%";
    $query_add .= $str_search_add . "%'";
    $res_add = mysql_query($query_add);
    while ($pole_add = mysql_fetch_array($res_add)) {
        echo $pole_add[0] . '<br>';
    }

    mysql_close($db);
}
else {
    echo 'Не удалось установить подключение к базе данных';
}
?>
```

Открыв этот файл в Web-браузере, мы увидим:

```
Без добавления слэшей перед спецсимволами:
Скидка 10%
Скидка 10$

После добавления слэшей перед спецсимволами:
Скидка 10%
```

В этом примере предполагается, что значение переменной `$str_search` было получено через форму поиска. Поэтому, прежде чем подставить значение переменной в SQL-запрос, мы экранируем специальные символы. Если этого не сделать, то любой пользователь может видоизменить SQL-запрос.

6.9. Поиск с помощью регулярных выражений

Регулярные выражения дают возможность осуществить сложный поиск. Использовать регулярные выражения позволяют следующие операторы:

- `RLIKE` — соответствие регулярному выражению;
- `REGEXP` — соответствие регулярному выражению (синоним `RLIKE`);
- `NOT RLIKE` — несоответствие регулярному выражению;
- `NOT REGEXP` — несоответствие регулярному выражению (синоним `NOT RLIKE`).

При использовании регулярных выражений следует помнить, что такой поиск выполняется медленнее, чем поиск по шаблону, и отнимает у сервера больше системных ресурсов. Кроме того, при работе с многобайтовыми кодировками регулярные выражения могут давать некорректный результат.

6.9.1. Метасимволы, используемые в регулярных выражениях

Синтаксис регулярных выражений MySQL, PHP и JavaScript схож. В регулярных выражениях может встречаться ряд метасимволов:

- `^` — привязка к началу строки.
- `$` — привязка к концу строки.

  Привязки работают так:

```
SELECT '2' RLIKE '^[0-9]+$';
/* Выведет: 1 */
SELECT 'Строка2' RLIKE '^[0-9]+$';
/* Выведет: 0 */
```

  Если убрать привязку к началу и концу строки, то любая строка, содержащая цифру, вернет `1`:

```
SELECT 'Строка2' RLIKE '[0-9]+';
/* Выведет: 1 */
```

  Можно указать привязку только к началу или только к концу строки:

```
SELECT 'Строка2' RLIKE '[0-9]+$';
/* Есть цифра в конце строки, значит выведет: 1 */
SELECT 'Строка2' RLIKE '^[0-9]+';
/* Нет цифры в начале строки, значит выведет: 0 */
```

  Регулярное выражение `'^$'` соответствует пустой строке. Если необходимо найти значение `NULL`, то следует использовать не регулярные выражения, а операторы `IS NULL` и `IS NOT NULL`.

- `[[:<:]]` — привязка к началу слова.
- `[[:>:]]` — привязка к концу слова.

```
SET NAMES 'cp1251';
SELECT 'в середине строки' RLIKE '[[:<:]]середине[[:>:]]';
/* Выведет: 1 */
```

  При работе с русским языком необходимо правильно настроить кодировку. Если кодировка соединения будет `latin1`, то поиск вернет значение `0`, а не `1`.

- Квадратные скобки (`[]`) позволяют указать символы, которые могут встречаться на этом месте в строке. Можно перечислять символы подряд или указать диапазон через дефис:

- `[09]` — цифра 0 или 9;
- `[0-9]` — любая цифра от 0 до 9;
- `[абв]` — буквы "а", "б" или "в";
- `[а-г]` — буквы "а", "б", "в" или "г";
- `[а-яе]` — любая буква от "а" до "я";
- `[0-9а-яеа-z]` — любая цифра и любая буква независимо от языка.

Значение можно инвертировать, если после первой скобки указать символ `^`. Таким образом можно указать символы, которых не должно быть на этом месте в строке:

- `[^09]` — не цифра 0 и не цифра 9;
- `[^0-9]` — не цифра от 0 до 9;
- `[^а-яеа-z]` — не буква.

Поиск выполняется без учета регистра символов. Чтобы учитывался регистр, необходимо указать ключевое слово `BINARY`:

```
SELECT 'String' RLIKE '^[a-z]+$';
/* Выведет: 1 */
SELECT 'String' RLIKE BINARY '^[a-z]+$';
/* Выведет: 0 */
```

Следует заметить, что учет регистра русских букв зависит от настройки кодировки.

```
SET NAMES 'cp866';
SELECT 'Строка' RLIKE '^[а-яе]+$';
/* Выведет: 1 */
SET NAMES 'cp1251';
SELECT 'Строка' RLIKE '^[а-яе]+$';
/* Выведет: 0 */
SET NAMES 'latin1';
SELECT 'Строка' RLIKE '^[а-яе]+$';
/* Выведет: 0 */
```

Как видно из примера, с кодировкой cp1251 также возникла проблема. Поэтому в регулярных выражениях для русских букв желательно учитывать регистр:

```
SET NAMES 'cp1251';
SELECT 'Строка' RLIKE '^[А-ЯЕа-яе]+$';
/* Выведет: 1 */
```

Вместо перечисления символов можно использовать стандартные классы:

- `[[:alnum:]]` — алфавитно-цифровые символы;
- `[[:alpha:]]` — буквенные символы;
- `[[:lower:]]` — строчные буквы;

- `[[:upper:]]` — прописные буквы;
- `[[:digit:]]` — десятичные цифры;
- `[[:xdigit:]]` — шестнадцатеричные цифры;
- `[[:punct:]]` — знаки пунктуации;
- `[[:blank:]]` — символы табуляции и пробелов;
- `[[:space:]]` — символы пробела, табуляции, новой строки или возврата каретки;
- `[[:cntrl:]]` — управляющие символы;
- `[[:print:]]` — печатные символы;
- `[[:graph:]]` — печатные символы, за исключением пробельных;
- . (точка) — любой символ.

Что же делать, если нужно найти точку, ведь символ "точка" соответствует любому символу? Для этого перед специальным символом необходимо указать два символа \ (`\\.`).

```
SELECT '26,03.2008' RLIKE
'^[0-3][[:digit:]].[01][[:digit:]].[12][09][[:digit:]][[:digit:]]$';
/* Поскольку точка означает любой символ, выведет: 1 */
SELECT '26,03.2008' RLIKE
'^[0-3][[:digit:]]\\.[01][[:digit:]]\\.[12][09][[:digit:]][[:digit:]]$';
/* Поскольку перед точкой указаны символы "\\", выведет: 0 */
```

Если символ "точка" указан впутри квадратных скобок, то он теряет свое специальное значение:

```
SELECT '26.03.2008' RLIKE
'^[0-3][[:digit:]][.][01][[:digit:]][.][12][09][[:digit:]][[:digit:]]$';
/* Выведет: 1 */
SELECT '26,03.2008' RLIKE
'^[0-3][[:digit:]][.][01][[:digit:]][.][12][09][[:digit:]][[:digit:]]$';
/* Выведет: 0 */
```

Число вхождений предшествующего символа или выражения в строку задается с помощью *квантификаторов*:

- `{n}` — `n` вхождений символа (выражения) в строку:

  `[[:digit:]]{2}` — соответствует двум вхождениям любой цифры;

- `{n,}` — `n` или более вхождений символа в строку:

  `[[:digit:]]{2,}` — соответствует двум и более вхождениям любой цифры;

- `{n,m}` — не менее `n` и не более `m` вхождений символа в строку. Цифры указываются через запятую без пробела:

  `[[:digit:]]{2,5}` — соответствует двум, трем, четырем или пяти вхождениям любой цифры;

- `*` — любое число вхождений символа в строку, в том числе ни одного:

  `[[:digit:]]*` — цифры могут не встретиться в строке или встретиться много раз;

- `+` — как минимум одно вхождение символа в строку:

  `[[:digit:]]+` — цифра может встретиться один или много раз;

- `?` — одно или ни одного вхождения символа в строку:

  `[[:digit:]]?` — цифра может встретиться один раз или не встретиться совсем.

Для указания числа вхождений нескольких символов используются круглые скобки:

```
SELECT '121212' RLIKE '^(12){3}$';
/* Выведет: 1 */
```

Также можно искать одно из двух выражений:

- `n|m` — один из символов (выражений) `n` или `m`:

```
красн(ая)|(ое) — красная или красное, но не красный, например
SELECT 'красная' RLIKE 'красн(ая)|(ое)';
/* Выведет: 1 */
```

6.10. Режим полнотекстового поиска

Кроме поиска по шаблону и применения регулярных выражений для таблиц типа `MyISAM` можно задать режим полнотекстового поиска. Столбцы, используемые для поиска, должны быть проиндексированы с помощью специального индекса `FULLTEXT`. Индексации подлежат столбцы, имеющие тип `CHAR`, `VARCHAR` или `TEXT`.

6.10.1. Создание индекса *FULLTEXT*

Создать индекс `FULLTEXT` можно следующими способами:

- при создании таблицы посредством оператора `CREATE TABLE` с помощью ключевого слова

```
FULLTEXT INDEX <Название индекса> (<Столбцы через запятую>)
```

Например:

```
CREATE TABLE `search1` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `str` TEXT,
   FULLTEXT INDEX `index1` (`str`),
   PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

или, если требуется создать индекс на основе сразу нескольких полей,

```
CREATE TABLE `search2` (
   `id` INT NOT NULL AUTO_INCREMENT,
```

```
    `str1` TEXT,
    `str2` TEXT,
    FULLTEXT INDEX `index2` (`str1`, `str2`),
    PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

- с помощью оператора `ALTER TABLE`, например, создав таблицу:

```
CREATE TABLE `search3` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `str` TEXT,
    PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

можно добавить к ней полнотекстовый индекс так:

```
ALTER TABLE `search3` ADD FULLTEXT `index3` (`str`);
```

А к таблице с двумя текстовыми полями:

```
CREATE TABLE `search4` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `str1` TEXT,
    `str2` TEXT,
    PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

можно добавить индекс так:

```
ALTER TABLE `search4` ADD FULLTEXT `index4` (`str1`, `str2`);
```

- с помощью оператора `CREATE INDEX`, например, к созданной ранее таблице с одним текстовым полем можно добавить индекс так:

```
CREATE FULLTEXT INDEX `index5` ON `search3` (`str`);
```

а к таблице с двумя полями так:

```
CREATE FULLTEXT INDEX `index6` ON `search4` (`str1`, `str2`);
```

В индекс попадут слова длиной от 4 до 84 символов. Данные значения задаются переменными `ft_min_word_len` и `ft_max_word_len` соответственно. Изменить значения этих переменных можно через конфигурационный файл my.ini. После изменения значения переменных необходимо заново создать индекс `FULLTEXT`. Посмотреть текущие значения переменных позволяет SQL-команда:

```
SHOW VARIABLES LIKE 'ft%';
```

Следует отметить, что полнотекстовый поиск предназначен для поиска в большом объеме текста. Если содержимое поля состоит из нескольких слов, то оно может вообще не попасть в индекс.

6.10.2. Реализация полнотекстового поиска

Полнотекстовый поиск выполняется с помощью конструкции `MATCH(...) AGAINST(...)`, которая имеет следующий формат:

```
MATCH(<Поля через запятую>)
AGAINST('<Строка для поиска>' [<Модификатор>])
```

Необязательный параметр `<Модификатор>` может принимать следующие значения:

- `IN BOOLEAN MODE` — режим логического поиска;
- `WITH QUERY EXPANSION` — поиск с расширением запроса.

Для примера добавим три записи в таблицу `search1`:

```
INSERT INTO `search1` VALUES (NULL, 'В индекс попадут слова длиной от 4
до 84 символов. Данные значения задаются переменными ft_min_word_len и
ft_max_word_len соответственно. Изменить значения этих переменных можно через
конфигурационный файл my.ini. После изменения значения переменных необходимо
заново создать индексы FULLTEXT.');
INSERT INTO `search1` VALUES (NULL, 'Запись 2');
INSERT INTO `search1` VALUES (NULL, 'Строка 3');
```

> ***Внимание!***
> Для реализации полнотекстового поиска в таблице должно быть не менее трех записей.

А теперь найдем строку с помощью полнотекстового поиска:

```
SELECT * FROM `search1` WHERE MATCH(`str`) AGAINST('значения');
```

Результаты поиска сортируются по коэффициенту релевантности, который представляет собой число с плавающей точкой. Чтобы увидеть этот коэффициент для разных запросов, воспользуемся следующими SQL-запросами:

```
SELECT MATCH(`str`) AGAINST('конфигурационный файл') AS iq
FROM `search1` WHERE MATCH(`str`) AGAINST('конфигурационный файл');
/* Выведет: 0.99839779903687 */
SELECT MATCH(`str`) AGAINST('конфигурационный файл') AS iq
FROM `search1` WHERE MATCH(`str`) AGAINST('конфигурационный файл');
/* Выведет: 0.49919889951843 */
```

6.10.3. Режим логического поиска

Режим логического поиска позволяет использовать специальные символы, которые влияют на значение коэффициента релевантности. Чтобы применить режим логического поиска, необходимо в конструкции `MATCH(...) AGAINST(...)` указать модификатор `IN BOOLEAN MODE`. Перечислим специальные символы логического режима:

- `+` — слово обязательно должно присутствовать в результате:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('конфигурационный +файл' IN BOOLEAN MODE);
```

- `-` — слово не должно присутствовать в результате:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('конфигурационный -файл' IN BOOLEAN MODE);
```

- `<` — уменьшает вклад слова в коэффициент релевантности:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('конфигурационный <файл' IN BOOLEAN MODE);
```

- `>` — увеличивает вклад слова в коэффициент релевантности:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('конфигурационный >файл' IN BOOLEAN MODE);
```

- `()` — круглые скобки служат для группировки слов в подвыражения;
- `~` — символ для указания нежелательного слова. В отличие от символа `-` символ `~` не исключает слово из результата, а лишь уменьшает коэффициент релевантности;
- `*` — символ усечения. Указывается в конце слова;
- `""` — строка должна содержать точную фразу:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('"конфигурационный файл"' IN BOOLEAN MODE);
```

6.10.4. Поиск с расширением запроса

При поиске с расширением запроса поиск фактически выполняется дважды. При первом поиске отбираются все записи, содержащие искомые слова. А при выполнении второго поиска ищутся записи, что включают наиболее релевантные слова из записей, найденных при первом поиске, даже если самих искомых слов в них нет. Все записи, найденные в процессе выполнения обоих поисков, объединяются и возвращаются в качестве результата.

В качестве примера можно рассмотреть поиск записей, содержащих словосочетание "база данных". При первом поиске будут найдены две записи, первая из которых содержит это словосочетание, а второе — еще и слово "MySQL". Тогда при втором поиске будут отобраны все записи, содержащие слово "MySQL". Результатом такого поиска будет набор, включающий как записи со словосочетанием "база данных", так и записи, что содержат слово "MySQL".

Чтобы применить режим поиска с расширением запроса, необходимо в конструкции `MATCH(...) AGAINST(...)` указать модификатор `WITH QUERY EXPANSION`.

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('конфигурационный файл' WITH QUERY EXPANSION);
```

6.11. Функции MySQL

MySQL имеет множество встроенных функций, которые позволяют выполнять определенные действия с данными. Например, функция `NOW()` возвращает текущие дату и время, а функция `DATE_FORMAT()` преобразует формат вывода даты.

При использовании функций следует помнить, что между круглыми скобками и именем функции не должно быть пробела, а скобки нужно обязательно указывать, даже если в функцию не передаются аргументы. Рассмотрим функции MySQL более подробно.

6.11.1. Функции для работы с числами

Стандартные тригонометрические функции (аргументы должны задаваться в радианах):

- `SIN()` — синус;
- `COS()` — косинус;
- `TAN()` — тангенс;
- `COT()` — котангенс.

Обратные тригонометрические функции (возвращают значение в радианах):

- `ASIN()` — арксинус;
- `ACOS()` — арккосинус;
- `ATAN()` — арктангенс.

Округление чисел:

- `CEILING()` — значение, округленное до ближайшего большего целого:

```
SELECT CEILING(4.3);
/* Выведет: 5 */
```

- `CEIL()` — то же, что и `CEILING()`;
- `FLOOR()` — значение, округленное до ближайшего меньшего целого:

```
SELECT FLOOR(4.6);
/* Выведет: 4 */
```

- `ROUND(X[, Y])` — значение, округленное до ближайшего меньшего целого для чисел с дробной частью меньше 0.5 или до ближайшего большего целого для чисел с дробной частью равной или больше 0.5:

```
SELECT ROUND(4.49);
/* Выведет: 4 */
SELECT ROUND(4.5);
/* Выведет: 5 */
SELECT ROUND(4.51);
/* Выведет: 5 */
```

Вторым аргументом для функции можно указать число знаков после занятой, до которых пужно округлить число:

```
SELECT ROUND(4.49, 1);
/* Выведет: 4.5 */
SELECT ROUND(12.34321, 3);
/* Выведет: 12.343 */
```

- `TRUNCATE(X, Y)` — возвращает дробное число `X`, имеющее `Y` знаков после запятой. Если в качестве значения аргумента `Y` передать значение 0, то функция вернет число, округленное до меньшего целого:

```
SELECT TRUNCATE(4.55, 0);
/* Выведет: 4 */
SELECT TRUNCATE(4.55, 1);
/* Выведет: 4.5 */
SELECT TRUNCATE(4.55, 3);
/* Выведет: 4.550 */
```

Функции для преобразования чисел:

- `CONV(<Число>, <Исходная система>, <Нужная система>)` — преобразует число из одной системы счисления в другую:

```
SELECT CONV(255, 10, 16);
/* Выведет: FF */
SELECT CONV('FF', 16, 10);
/* Выведет: 255 */
```

- `BIN(<Число>)` — преобразует число из десятичной системы счисления в двоичную:

```
SELECT BIN(17);
/* Выведет: 10001 */
```

- `HEX(<Число>)` — возвращает значение аргумента в виде шестнадцатеричного числа:

```
SELECT HEX(255);
/* Выведет: FF */
```

- `OCT(<Число>)` — преобразует число из десятичной системы счисления в восьмеричпую:

```
SELECT OCT(10);
/* Выведет: 12 */
```

Прочие функции:

- `ABS()` — абсолютное значение:

```
SELECT ABS(-4.55);
/* Выведет: 4.55 */
```

- `EXP()` — экспонента;
- `LOG(X)` — натуральный логарифм;
- `LOG2(X)` — логарифм числа по основанию 2:

```
SELECT LOG2(128);
/* Выведет: 7 */
```

- `LOG10(X)` — логарифм числа по основанию 10:

```
SELECT LOG10(100);
/* Выведет: 2 */
```

- `LOG(<Основание>, X)` — логарифм числа `X` по основанию `<Основание>`:

```
SELECT LOG(2, 128);
/* Выведет: 7 */
SELECT LOG(10, 100);
/* Выведет: 2 */
```

- `POW(<Число>, <Степень>)` — возводит `<Число>` в `<Степень>`:

```
SELECT POW(5, 2);
/* Выведет: 25 */
```

- `SQRT()` — извлекает квадратный корень:

```
SELECT SQRT(25);
/* Выведет: 5 */
```

- `PI()` — возвращает число π:

```
SELECT PI();
/* Выведет: 3.141593 */
```

- `MOD(X, Y)` — определяет остаток от деления `X` на `Y`:

```
SELECT MOD(10, 2);
/* Выведет: 0 */
```

- `DEGREES()` — преобразует значение угла из радиан в градусы:

```
SELECT DEGREES(PI());
/* Выведет: 180 */
```

- `RADIANS()` — преобразует значение угла из градусов в радианы:

```
SELECT RADIANS(180);
/* Выведет 3.141592653589793 */
```

- `SIGN()` — возвращает `-1`, если число отрицательное, `1`, если число положительное, и `0`, если число равно нулю:

```
SELECT SIGN(-80);
/* Выведет: -1 */
```

```
SELECT SIGN(80);
/* Выведет: 1 */
```

- `LEAST()` — служит для определения минимального значения из списка:

```
SELECT LEAST(2, 1, 3);
/* Выведет: 1 */
```

- `GREATEST()` — позволяет определить максимальное значение из списка:

```
SELECT GREATEST(2, 1, 3);
/* Выведет: 3 */
```

- `FORMAT(<Число>, <Число знаков после запятой>[, <Локаль>])` — форматирует число в строку с заданным числом знаков после запятой в соответствии с заданной локалью. Если локаль не указана, используется американская (en_US):

```
SELECT FORMAT(56873.8732, 2);
/* Выведет: 56,873.87 */
SELECT FORMAT(56873.8732, 2, 'ru_RU');
 /* Здесь мы указали российскую локаль */
 /* Выведет: 56.873,87 */
```

- `RAND()` — возвращает случайное число в диапазоне от 0 до 1. Если в функцию передать параметр, то это настроит генератор на новую последовательность. Следует учитывать, что при передаче одного и того же параметра функция выдает одпу и ту же последовательность:

```
SELECT RAND();
/* Выведет: 0.35286363153985106 */
SELECT RAND();
/* Выведет: 0.7805252687824195 */
SELECT RAND(10);
/* Выведет: 0.6570515219653505 */
SELECT RAND(10);
/* Выведет: 0.6570515219653505 */
```

В качестве примера рассмотрим вывод записи из базы данных случайным образом. Предположим, что наш сайт — развлекательный портал, и в базе данных есть таблица, заполненная анекдотами. При каждом запросе страницы мы будем выводить один анекдот случайным образом. Для этого в базе данных tests создадим таблицу `anecdotes`:

```
SET NAMES cp866;
CREATE TABLE `anecdotes` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `anecdote` TEXT,
   PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим несколько записей:

```
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 1');
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 2');
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 3');
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 4');
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 5');
```

Исходный код для вывода анекдота случайным образом приведен в листинге 6.3.

Листинг 6.3. Вывод анекдота случайным образом

```
<?php
if ($db = @mysql_connect("localhost", "root", "123456")) {
   mysql_select_db('tests');
   mysql_query("SET NAMES cp1251");
   $query = 'SELECT * FROM `anecdotes` ORDER BY RAND() LIMIT 1';
   $res = mysql_query($query);
   if (mysql_num_rows($res) == 1) {
      echo mysql_result($res, 0, 'anecdote');
   }
   mysql_close($db);
}
?>
```

6.11.2. Функции даты и времени

Для получения текущей даты и времени предусмотрены следующие функции:

- `NOW()`, `LOCALTIME()` и `LOCALTIMESTAMP()` — возвращают текущие дату и время для временной зоны, установленной в системных настройках, в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`;

```
SELECT NOW();
/* Выведет: 2014-12-03 16:15:24 */
SELECT LOCALTIME();
/* Выведет: 2014-12-03 16:15:24 */
SELECT LOCALTIMESTAMP();
/* Выведет: 2014-12-03 16:15:24 */
```

- `UTC_TIMESTAMP()` — выводит текущие дату и время по Гринвичу в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`;

```
SELECT UTC_TIMESTAMP();
/* Выведет: 2014-12-03 15:15:24 */
```

- `SYSDATE()` — позволяет определить текущие дату и время в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:

```
SELECT SYSDATE();
/* Выведет: 2014-12-03 16:15:25 */
```

В отличие от функции NOW() и ее синонимов, SYSDATE() возвращает время, в которое она была вызвана, тогда как NOW() возвращает время начала выполнения запроса;

- CURDATE() и CURRENT_DATE() — возвращают текущую дату для временной зоны, установленной в системных настройках, в формате ГГГГ-ММ-ДД:

```
SELECT CURDATE();
/* Выведет: 2014-12-03 */
SELECT CURRENT_DATE();
/* Выведет: 2014-12-03 */
```

- UTC_DATE() — позволяет определить текущую дату по Гринвичу в формате ГГГГ-ММ-ДД:

```
SELECT UTC_DATE();
/* Выведет: 2014-12-03 */
```

- CURTIME() и CURRENT_TIME() — возвращают текущее время для временной зоны, установленной в системных настройках, в формате ЧЧ:ММ:СС;

```
SELECT CURTIME();
/* Выведет: 16:15:25 */
SELECT CURRENT_TIME();
/* Выведет: 16:15:25 */
```

- UTC_TIME() — сообщает текущее время по Гринвичу в формате ЧЧ:ММ:СС:

```
SELECT UTC_TIME();
/* Выведет: 15:15:25 */
```

- UNIX_TIMESTAMP() — подсчитывает число секунд, прошедших с полуночи 1 января 1970 г.:

```
SELECT UNIX_TIMESTAMP();
/* Выведет: 1417612779 */
```

Ряд функций позволяют получить следующие фрагменты даты и времени:

- DATE() — дата:

```
SELECT DATE('2014-12-03 16:15:24');
/* Выведет: 2014-12-03 */
```

- YEAR() — год:

```
SELECT YEAR('2014-12-03 16:15:24');
/* Выведет: 2014 */
```

- MONTH() — месяц:

```
SELECT MONTH('2014-12-03 16:15:24');
/* Выведет: 12 */
```

- `MONTHNAME()` — английское название месяца в виде строки:

```
SELECT MONTHNAME('2014-12-03 16:15:24');
/* Выведет: December */
```

- `DAY()` и `DAYOFMONTH()` — номер дня в месяце:

```
SELECT DAY('2014-12-03 16:15:24');
/* Выведет: 3 */
SELECT DAYOFMONTH('2014-12-03 16:15:24');
/* Выведет: 3 */
```

- `TIME()` — время:

```
SELECT TIME('2014-12-03 16:15:24');
/* Выведет: 16:15:24 */
```

- `HOUR()` — час:

```
SELECT HOUR('2014-12-03 16:15:24');
/* Выведет: 16 */
```

- `MINUTE()` — мипуты:

```
SELECT MINUTE('2014-12-03 16:15:24');
/* Выведет: 15 */
```

- `SECOND()` — секунды:

```
SELECT SECOND('c');
/* Выведет: 24 */
```

- `MICROSECOND()` — микросекунды:

```
SELECT MICROSECOND('2014-12-03 16:15:24.123456');
/* Выведет: 123456 */
```

Вместо перечисленных функций можно использовать функцию `EXTRACT()`. Формат функции:

```
EXTRACT(<Тип> FROM <Дата и время>)
```

Значения параметра `<Тип>`:

- `YEAR` — год:

```
SELECT EXTRACT(YEAR FROM '2014-12-03 16:15:24');
/* Выведет: 2014 */
```

- `YEAR_MONTH` — год и месяц:

```
SELECT EXTRACT(YEAR_MONTH FROM '2014-12-03 16:15:24');
/* Выведет: 201412 */
```

- `MONTH` — месяц:

```
SELECT EXTRACT(MONTH FROM '2014-12-03 16:15:24');
/* Выведет: 12 */
```

- DAY — день:

```
SELECT EXTRACT(DAY FROM '2014-12-03 16:15:24');
/* Выведет: 3 */
```

- DAY_HOUR — день и час:

```
SELECT EXTRACT(DAY_HOUR FROM '2014-12-03 16:15:24');
/* Выведет: 316 */
```

- DAY_MINUTE — день, час и минуты:

```
SELECT EXTRACT(DAY_MINUTE FROM '2014-12-03 16:15:24');
/* Выведет: 31615 */
```

- DAY_SECOND — день, час, минуты и секунды:

```
SELECT EXTRACT(DAY_SECOND FROM '2014-12-03 16:15:24');
/* Выведет: 3161524 */
```

- DAY_MICROSECOND — день, час, минуты, секунды и микросекунды:

```
SELECT EXTRACT(DAY_MICROSECOND FROM '2014-12-03 16:15:24.111111');
/* Выведет: 3161524111111 */
```

- HOUR — час:

```
SELECT EXTRACT(HOUR FROM '2014-12-03 16:15:24');
/* Выведет: 16 */
```

- HOUR_MINUTE — час и мипуты:

```
SELECT EXTRACT(HOUR_MINUTE FROM '2014-12-03 16:15:24');
/* Выведет: 1615 */
```

- HOUR_SECOND — час, мипуты и секунды:

```
SELECT EXTRACT(HOUR_SECOND FROM '2014-12-03 16:15:24');
/* Выведет: 161524 */
```

- HOUR_MICROSECOND — час, мипуты, секунды и микросекунды:

```
SELECT EXTRACT(HOUR_MICROSECOND FROM '2014-12-03 16:15:24.111111');
/* Выведет: 161524111111 */
```

- MINUTE — мипуты:

```
SELECT EXTRACT(MINUTE FROM '2014-12-03 16:15:24');
/* Выведет: 15 */
```

- MINUTE_SECOND — мипуты и секунды:

```
SELECT EXTRACT(MINUTE_SECOND FROM '2014-12-03 16:15:24');
/* Выведет: 1524 */
```

- `MINUTE_MICROSECOND` — минуты, секунды и микросекунды:

```
SELECT EXTRACT(MINUTE_MICROSECOND FROM '2014-12-03
16:15:24.111111');
/* Выведет: 1524111111 */
```

- `SECOND` — секунды:

```
SELECT EXTRACT(SECOND FROM '2014-12-03 16:15:24');
/* Выведет: 24 */
```

- `SECOND_MICROSECOND` — секунды и микросекунды:

```
SELECT EXTRACT(SECOND_MICROSECOND
FROM '2014-12-03 16:15:24.111111');
/* Выведет: 24111111 */
```

- `MICROSECOND` — микросекунды:

```
SELECT EXTRACT(MICROSECOND FROM '2014-12-03 16:15:24.111111');
/* Выведет: 111111 */
```

С помощью следующих функций можно получить дополнительные сведения о дате:

- `QUARTER()` — порядковый номер квартала в году (от 1 до 4):

```
SELECT QUARTER('2014-12-03');
/* Выведет: 4 */
```

- `WEEK(D[, M])` — порядковый номер недели. Необязательный второй параметр задает режим вычисления номера недели; все доступные режимы и их описание перечислены в табл. 6.14.

Таблица 6.14. *Режимы вычисления номера недели*

Режим	День, с которого начинается неделя	Диапазон номеров недель	Сколько дней должна включать первая неделя года
0	Воскресенье	0—53	Лишь воскресенье
1	Понедельник		Четыре или более дней
2	Воскресенье	1—53	Лишь воскресенье
3	Понедельник		Четыре или более дней
4	Воскресенье	0—53	
5	Понедельник		Понедельник
6	Воскресенье	1—53	Четыре или более дней
7	Понедельник		Понедельник

Если второй параметр не задан, для него устанавливается значение 0.

Получаем номер недели при условии, что она начинается с воскресенья, а первая неделя года должна включать лишь само воскресенье, в диапазоне 0—53:

```
SELECT WEEK('2014-12-03');
/* Выведет: 48 */
```

Получаем номер недели при условии, что она начинается с понедельника, а первая неделя года должна включать, по меньшей мере, четыре дня, в диапазоне 1—53:

```
SELECT WEEK('2014-12-03', 3);
/* Выведет: 49 */
```

- `WEEKOFYEAR()` — порядковый номер недели в году (от 1 до 53). Фактически выполняет ту же задачу, что вызов функции `WEEK()` с параметром режима 3:

```
SELECT WEEKOFYEAR('2014-12-03');
/* Выведет: 49 */
```

- `YEARWEEK(D[, M])` — число в формате `ГГГГНН`, где `ГГГГ` — год, а `НН` — порядковый номер недели в году. Второй параметр выполняет ту же задачу, что второй параметр функции `WEEK()`:

```
SELECT YEARWEEK('2014-12-03', 3);
/* Выведет: 201449 */
```

- `DAYOFYEAR()` — порядковый номер дня в году (от 1 до 366):

```
SELECT DAYOFYEAR('2014-12-03');
/* Выведет: 337 */
```

- `MAKEDATE(<Год>, <Номер дня в году>)` — дата в формате `ГГГГ-ММ-ДД` по номеру дня в году:

```
SELECT MAKEDATE(2014, 337);
/* Выведет: 2014-12-03 */
```

- `DAYOFWEEK()` — порядковый номер дня недели (1 — для воскресенья, 2 — для понедельника, ..., 7 — для субботы):

```
SELECT DAYOFWEEK('2014-12-03');
/* Выведет: 4 */
```

- `WEEKDAY()` — порядковый номер дня недели (0 — для понедельника, 1 — для вторника, ..., 6 — для воскресенья):

```
SELECT WEEKDAY('2014-12-03');
/* Выведет: 3 */
```

- `DAYNAME()` — название дня недели на английском языке:

```
SELECT DAYNAME('2014-12-03');
/* Выведет: Wednesday */
```

- `TO_DAYS(<Дата>)` — число дней, прошедших с нулевого года:

```
SELECT TO_DAYS('2014-12-03');
/* Выведет: 735935 */
```

- FROM_DAYS(<Число дней>) — дата в формате ГГГГ-ММ-ДД по числу дней, прошедших с нулевого года:

```
SELECT FROM_DAYS(735935);
/* Выведет: 2014-12-03 */
```

- TIME_TO_SEC(<Время>) — число секунд, прошедших с начала суток:

```
SELECT TIME_TO_SEC('16:15:24');
/* Выведет: 58524 */
```

- SEC_TO_TIME(<Число секунд>) — время в формате ЧЧ:ММ:СС по числу секунд, прошедших с начала суток:

```
SELECT SEC_TO_TIME(58524);
/* Выведет: 16:15:24 */
```

Для манинуляции датой и временем можно использовать следующие функции:

- ADDDATE(<Дата>, INTERVAL <Интервал> <Тип>) и DATE_ADD(<Дата>, INTERVAL <Интервал> <Тип>) — прибавляют к параметру <Дата> временной интервал;
- SUBDATE(<Дата>, INTERVAL <Интервал> <Тип>) и DATE_SUB(<Дата>, INTERVAL <Интервал> <Тип>) — вычитают из параметра <Дата> временной интервал.

Параметр <Тип> в функциях ADDDATE(), DATE_ADD(), SUBDATE() и DATE_SUB() может принимать следующие значения:

- YEAR — год:

```
SELECT ADDDATE('2014-12-03 16:15:24', INTERVAL 2 YEAR);
/* Выведет: 2016-12-03 16:15:24 */
```

- YEAR_MONTH — год и месяц (формат 'ГГ-ММ'):

```
SELECT ADDDATE('2014-12-03 16:15:24', INTERVAL '2-2' YEAR_MONTH);
/* Выведет: 2017-02-03 16:15:24 */
```

- MONTH — месяц:

```
SELECT ADDDATE('2014-12-03 16:15:24', INTERVAL 3 MONTH);
/* Выведет: 2015-03-03 16:15:24 */
```

- DAY — день:

```
SELECT ADDDATE('2014-12-03 16:15:24', INTERVAL 6 DAY);
/* Выведет: 2014-12-09 16:15:24 */
```

- DAY_HOUR — день и час (формат 'ДД ЧЧ'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '6 3' DAY_HOUR);
/* Выведет: 2014-12-09 19:15:24 */
```

- DAY_MINUTE — день, час и минуты (формат 'ДД ЧЧ:ММ'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '6 3:5' DAY_MINUTE);
/* Выведет: 2014-12-09 19:20:24 */
```

- DAY_SECOND — день, час, минуты и секунды (формат 'ДД ЧЧ:ММ:СС'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '6 3:5:15' DAY_SECOND);
/* Выведет: 2014-12-09 19:20:39 */
```

- DAY_MICROSECOND — день, час, минуты, секунды и микросекунды (формат 'ДД ЧЧ:ММ:СС.XXXXXX'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '6 3:5:15.10' DAY_MICROSECOND);
/* Выведет: 2014-12-09 19:20:39.100000 */
```

- HOUR — час:

```
SELECT ADDDATE('2014-12-03 16:15:24', INTERVAL 3 HOUR);
/* Выведет: 2014-12-03 19:15:24 */
```

- HOUR_MINUTE — час и минуты (формат 'ЧЧ:ММ'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '3:7' HOUR_MINUTE);
/* Выведет: 2014-12-03 19:22:24 */
```

- HOUR_SECOND — час, минуты и секунды (формат 'ЧЧ:ММ:СС'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '3:7:15' HOUR_SECOND);
/* Выведет: 2014-12-03 19:22:39 */
```

- HOUR_MICROSECOND — час, минуты, секунды и микросекунды (формат 'ЧЧ:ММ:СС.XXXXXX'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '3:7:15.10' HOUR_MICROSECOND);
/* Выведет: 2014-12-03 19:22:39.100000 */
```

- MINUTE — минуты:

```
SELECT ADDDATE('2014-12-03 16:15:24', INTERVAL 8 MINUTE);
/* Выведет: 2014-12-03 16:23:24 */
```

- MINUTE_SECOND — минуты и секунды (формат 'ММ:СС'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '3:7' MINUTE_SECOND);
/* Выведет: 2014-12-03 16:18:31 */
```

- MINUTE_MICROSECOND — минуты, секунды и микросекунды (формат 'MM:CC.XXXXXX'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '3:7.11' MINUTE_MICROSECOND);
/* Выведет: 2014-12-03 16:18:31.110000 */
```

- SECOND — секунды:

```
SELECT ADDDATE('2014-12-03 16:15:24', INTERVAL 15 SECOND);
/* Выведет: 2014-12-03 16:15:39 */
```

- SECOND_MICROSECOND — секунды и микросекунды (формат 'CC.XXXXXX'):

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL '15.123456' SECOND_MICROSECOND);
/* Выведет: 2014-12-03 16:15:39.123456 */
```

- MICROSECOND — микросекунды:

```
SELECT ADDDATE('2014-12-03 16:15:24',
INTERVAL 123456 MICROSECOND);
/* Выведет: 2014-12-03 16:15:24.123456 */
```

Кроме того, функции ADDDATE() и SUBDATE() можно применять в сокращенном формате, описанном далее;

- ADDDATE(<Дата>, <Интервал в днях>) — прибавляет к параметру <Дата> временной интервал в днях. Если указать перед интервалом знак -, то интервал вычитается из даты:

```
SELECT ADDDATE('2014-12-03', 10);
/* Выведет: 2014-12-13 */
SELECT ADDDATE('2014-12-03', -10);
/* Выведет: 2014-11-23 */
```

- SUBDATE(<Дата>, <Интервал в днях>) — вычитает из параметра <Дата> временной интервал в днях.

  Если указать перед интервалом знак -, то интервал прибавляется к дате:

```
SELECT SUBDATE('2014-12-03', 10);
/* Выведет: 2009-11-23 */
SELECT SUBDATE('2014-12-03', -10);
/* Выведет: 2014-12-13 */
```

- ADDTIME(<Дата>, <Время>) — прибавляет к параметру <Дата> временной интервал:

```
SELECT ADDTIME('2014-12-03 16:15:24', '12:52:35');
/* Выведет: 2014-12-04 05:07:59 */
```

- SUBTIME(<Дата>, <Время>) — вычитает из параметра <Дата> временной интервал:

```
SELECT SUBTIME('2014-12-03 16:15:24', '12:52:35');
/* Выведет: 2014-12-03 03:22:49 */
```

- DATEDIFF(<Конечная дата>, <Начальная дата>) — вычисляет число дней между двумя датами:

```
SELECT DATEDIFF('2014-12-03', '2014-10-20');
/* Выведет: 44 */
```

- TIMEDIFF(<Конечная дата>, <Начальная дата>) — вычисляет разницу между двумя временными значениями:

```
SELECT TIMEDIFF('16:15:24', '08:43:17');
/* Выведет: 07:32:07 */
```

- PERIOD_ADD(<Дата>, <Число месяцев>) — добавляет заданное <Число месяцев> к дате, заданной в формате ГГГГММ или ГГММ:

```
SELECT PERIOD_ADD(201412, 4);
/* Выведет: 201404 */
```

- PERIOD_DIFF(<Конечная дата>, <Начальная дата>) — вычисляет разницу в месяцах между двумя временными значениями, заданными в формате ГГГГММ или ГГММ:

```
SELECT PERIOD_DIFF(201412, 201410);
/* Выведет: 2 */
```

- CONVERT_TZ(<Дата>, <Часовой пояс 1>, <Часовой пояс 2>) — переводит дату из одного часового пояса в другой:

```
SELECT CONVERT_TZ('2014-12-03 16:15:24', '+00:00', '+4:00');
/* Выведет: 2014-12-03 20:15:24 */
```

- LAST_DAY(<Дата>) — возвращает дату в формате ГГГГ-ММ-ДД, в которой день выставлен на последний день текущего месяца:

```
SELECT LAST_DAY('2014-12-03 16:15:24');
/* Выведет: 2014-12-31 */
```

- MAKETIME(<Часы>, <Минуты>, <Секунды>) — возвращает время в формате ЧЧ:ММ:СС:

```
SELECT MAKETIME(12, 52, 35);
/* Выведет: 12:52:35 */
```

- TIMESTAMP(<Дата>, [<Время>]) — возвращает дату в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС:

```
SELECT TIMESTAMP('2014-12-03');
/* Выведет: 2014-12-03 00:00:00 */
SELECT TIMESTAMP('2014-12-03', '13:48:11');
/* Выведет: 2014-12-03 13:48:11 */
```

Помимо описанных функций, добавить или вычесть интервал времени можно с помощью операторов + и -, за которыми следует ключевое слово INTERVAL, значение и

тип интервала. Применимы те же типы интервалов, что и в функциях ADDDATE(), DATE_ADD(), SUBDATE() и DATE_SUB():

```
SELECT '2014-12-03 16:15:24' + INTERVAL '3:7:15' HOUR_SECOND;
/* Выведет: 2014-12-03 19:22:39 */
SELECT '2014-12-03 16:15:24' - INTERVAL '3:7:15' HOUR_SECOND;
/* Выведет: 2014-12-03 13:08:09 */
```

Для форматирования даты и времени также предназначено несколько функций:

- DATE_FORMAT(<Дата>, <Формат>) — форматирует дату в соответствии со строкой <Формат>:

  ```
  SELECT DATE_FORMAT('2014-12-03 16:15:24', '%d.%m.%Y');
  /* Выведет: 03.12.2014 */
  ```

- STR_TO_DATE(<Дата>, <Формат>) — возвращает дату в форматах ГГГГ-ММ-ДД ЧЧ:ММ:СС или ГГГГ-ММ-ДД по дате, соответствующей строке <Формат>:

  ```
  SELECT STR_TO_DATE('20.10.2014 13:48:11', '%d.%m.%Y %H:%i:%s');
  /* Выведет: 2014-10-20 13:48:11 */
  ```

- TIME_FORMAT(<Время>, <Формат>) — форматирует время в соответствии со строкой <Формат>:

  ```
  SELECT TIME_FORMAT('16:15:24', '%H %i %s');
  /* Выведет: 16 15 24 */
  ```

- FROM_UNIXTIME(<Дата>, [<Формат>]) — возвращает дату в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС или соответствующую строке <Формат> по числу секунд, прошедших с полуночи 1 января 1970 г.:

  ```
  SELECT FROM_UNIXTIME(1417147122);
  /* Выведет: 2014-11-28 06:58:42 */
  SELECT FROM_UNIXTIME(1417147122, '%d.%m.%Y');
  /* Выведет: 28.11.2014 */
  ```

- GET_FORMAT(<Тип времени>, '<Стандарт>') — возвращает строку форматирования для пяти стандартов отображения даты и времени. Параметр <Тип времени> может принимать следующие значения:

  - DATETIME — дата и время;
  - DATE — дата;
  - TIME — время.

  Параметр <Стандарт> может принимать такие значения:

  - ISO — стандарт ISO;
  - EUR — европейский стандарт;
  - USA — американский стандарт;

- JIS — японский стандарт;
- INTERNAL — внутренний формат MySQL.

Пример:

```
SELECT GET_FORMAT(DATE, 'EUR');
/* Выведет: %d.%m.%Y */
SELECT DATE_FORMAT('2014-12-03 16:15:24',
GET_FORMAT(DATE, 'EUR'));
/* Выведет: 03.12.2014 */
```

Параметр <Формат> в функциях форматирования может содержать следующие комбинации символов:

- %Y — год из 4-х цифр;
- %y — год из 2-х цифр;
- %m — номер месяца с предваряющим нулем (от 01 до 12);
- %c — номер месяца без предваряющего нуля (от 1 до 12);
- %b — аббревиатура месяца из 3-х букв по-английски;
- %M — полное название месяца по-английски;
- %d — номер дня с предваряющим нулем (от 01 до 31);
- %e — номер дня без предваряющего нуля (от 1 до 31);
- %w — номер дня недели (0 — для воскресенья и 6 — для субботы);
- %a — аббревиатура дня недели из 3-х букв по-английски;
- %W — полное название дня недели по-английски;
- %H — часы в 24-часовом формате (от 00 до 23);
- %h и %I — часы в 12-часовом формате (от 01 до 12);
- %i — минуты (от 00 до 59);
- %s — секунды (от 00 до 59);
- %f — микросекунды;
- %% — знак процента.

Настройки параметра <Формат> зависят от значения переменной lc_time_names. Выведем текущее значение переменной:

```
SELECT @@lc_time_names;
/* Выведет: en_US */
```

В качестве примера изменим значение переменной и выведем название месяца на русском языке:

```
SELECT DATE_FORMAT('2014-12-03 16:15:24', '%d %M %Y');
/* Выведет: 03 December 2014 */
```

```
SET lc_time_names = 'ru_RU';
SELECT DATE_FORMAT('2014-12-03 16:15:24', '%d %M %Y');
/* Выведет: 03 Декабря 2014 */
```

6.11.3. Функции для обработки строк

Перечислим основные функции для обработки строк:

- `CHAR_LENGTH(<Строка>)` и `CHARACTER_LENGTH(<Строка>)` — возвращают длину строки в символах. Функции корректно работают с многобайтовыми кодировками:

```
SELECT CHAR_LENGTH('String');
/* Выведет: 6 */
SELECT CHARACTER_LENGTH('String');
/* Выведет: 6 */
```

- `LENGTH(<Строка>)` — возвращает длину строки в байтах (вследствие чего длина строк, записанных в многобайтовых кодировках, определяется некорректно):

```
SELECT LENGTH('String');
/* Выведет: 6 */
```

- `BIT_LENGTH(<Строка>)` — возвращает длину строки в битах:

```
SELECT BIT_LENGTH('String');
/* Выведет: 48 */
```

- `CONCAT(<Строка 1>, <Строка 2>, ..., <Строка N>)` — объединяет все параметры в одну строку:

```
SELECT CONCAT('string1', 'string2', 'string3');
/* Выведет: string1string2string3 */
```

- `CONCAT_WS(<Разделитель>, <Строка 1>, ..., <Строка N>)` — объединяет все параметры в одну строку через разделитель, заданный в параметре `<Разделитель>`:

```
SELECT CONCAT_WS(' - ', 'string1', 'string2', 'string3');
/* Выведет: string1 - string2 - string3 */
```

- `TRIM([[<Откуда>] [<Символы для удаления>] FROM] <Строка>)` — удаляет из начала (и/или конца) строки символы, указанные в параметре `<Символы для удаления>`. Если параметр не указан, то удаляются пробелы. Необязательный параметр `<Откуда>` может принимать значения:
  - `BOTH` — символы удаляются из начала и конца строки (по умолчанию);
  - `LEADING` — только из начала строки;
  - `TRAILING` — только из конца строки.

  Примеры:

```
SELECT CONCAT("'", TRIM('   String   '), "'");
/* Выведет: 'String' */
```

```
SELECT CONCAT("'", TRIM(LEADING FROM '   String   '), "'");
/* Выведет: 'String   ' */
SELECT CONCAT("'", TRIM(TRAILING FROM '   String   '), "'");
/* Выведет: '   String' */
SELECT CONCAT("'", TRIM(BOTH 'm' FROM 'mmmmStringmmmm'), "'");
/* Выведет: 'String' */
SELECT CONCAT("'", TRIM(TRAILING 'ing' FROM 'Stringing'), "'");
/* Выведет: 'Str' */
SELECT CONCAT("'", TRIM(TRAILING 'gn' FROM 'String'), "'");
/* Выведет: 'String' */
```

- `LTRIM(<Строка>)` — удаляет пробелы в начале строки:

```
SELECT CONCAT("'", LTRIM('   String   '), "'");
/* Выведет: 'String   ' */
```

- `RTRIM(<Строка>)` — удаляет пробелы в конце строки:

```
SELECT CONCAT("'", RTRIM('   String   '), "'");
/* Выведет: '   String' */
```

- `LOWER(<Строка>)` и `LCASE(<Строка>)` — переводят все символы в нижний регистр:

```
SELECT LOWER('STRING');
/* Выведет: string */
SELECT LCASE('String');
/* Выведет: string */
```

- `UPPER(<Строка>)` и `UCASE(<Строка>)` — переводят все символы в верхний регистр:

```
SELECT UPPER('string');
/* Выведет: STRING */
SELECT UCASE('String');
/* Выведет: STRING */
```

- `REVERSE(<Строка>)` — возвращает строку в обратном порядке:

```
SELECT REVERSE('string');
/* Выведет: gnirts */
```

- `LEFT(<Строка>, <Число символов>)` — возвращает заданное число крайних символов слева:

```
SELECT LEFT('string', 2);
/* Выведет: st */
```

- `RIGHT(<Строка>, <Число символов>)` возвращает заданное число крайних символов справа:

```
SELECT RIGHT('string', 2);
/* Выведет: ng */
```

- `SUBSTRING(<Строка>, <Начальная позиция>, [<Длина>])`, `SUBSTR(<Строка>, <Начальная позиция>, [<Длина>])` и `MID(<Строка>, <Начальная позиция>, [<Длина>])` — по-

зволяют получить подстроку заданной длины, начиная с позиции <Начальная позиция>. Если параметр <Длина> не задан, то возвращаются все символы до конца строки:

```
SELECT SUBSTRING('string', 2, 2);
/* Выведет: tr */
SELECT SUBSTR('string', 2, 2);
/* Выведет: tr */
SELECT MID('string', 2);
/* Выведет: tring */
```

Первые две функции имеют альтернативный синтаксис:

```
SELECT SUBSTRING('string' FROM 2 FOR 3);
/* Выведет: tri */
SELECT SUBSTRING('string' FROM 2);
/* Выведет: tring */
```

- LPAD(<Строка>, <Длина>, <Подстрока>) — добавляет подстроку к исходной строке слева, доводя общую длину строки до величины <Длина>:

```
SELECT LPAD('string', 11, 'mp');
/* Выведет: mpmpmstring */
```

- RPAD(<Строка>, <Длина>, <Подстрока>) — добавляет подстроку к исходной строке справа, доводя общую длину строки до величины <Длина>:

```
SELECT RPAD('string', 10, 'mp');
/* Выведет: stringmpmp */
```

- REPEAT(<Строка>, <Число повторений>) — возвращает строку, содержащую заданное число повторений исходной строки:

```
SELECT REPEAT('str', 3);
/* Выведет: strstrstr */
```

- SPACE(<Число пробелов>) — возвращает строку, состоящую из заданного числа пробелов:

```
SELECT CONCAT("'", SPACE(3), 'String', "'");
/* Выведет: '   String' */
```

- ELT(<Номер из списка>, <Строка 1>, ..., <Строка N>) — позволяет получить одну строку из списка параметров, номер которой задается первым параметром:

```
SELECT ELT(2, 'string1', 'string2', 'string3');
/* Выведет: string2 */
```

- ASCII(<Строка>) — возвращает код ASCII первого символа строки:

```
SELECT ASCII('String');
/* Выведет: 83 */
```

- `ORD(<Строка>)` — дает возможность узнать код первого символа строки. Корректно работает с многобайтовыми кодировками. Если первый символ — однобайтовый, вернет то же значение, что и `ASCII()`:

```
SELECT ORD('String');
/* Выведет: 83 */
```

- `CHAR(<ASCII-код1>, <ASCII-код2>, ..., <ASCII-кодN>)` — возвращает строку, состоящую из последовательности символов, соответствующих ASCII-кодам:

```
SELECT CHAR(83, 116, 114, 105, 110, 103);
/* Выведет: String */
```

- `INSTR(<Строка>, <Подстрока>)` или `POSITION(<Подстрока> IN <Строка>)` — ищут подстроку в строке и возвращают позицию ее первого вхождения. Если вхождение не найдено, то возвращается 0:

```
SELECT INSTR('string', 'st');
/* Выведет: 1 */
SELECT POSITION('st' IN 'string');
/* Выведет: 1 */
SELECT POSITION('pt' IN 'string');
/* Выведет: 0 */
```

- `LOCATE(<Подстрока>, <Строка>, [<Начальная позиция>])` — возвращает позицию первого вхождения подстроки в строку, начиная с указанной начальной позиции. Если подстрока не найдена, то возвращается 0. Если начальная позиция не указана, то поиск производится с начала строки:

```
SELECT LOCATE('st', 'string_st');
/* Выведет: 1 */
SELECT LOCATE('st', 'string_st', 3);
/* Выведет: 8 */
```

- `FIELD(<Исходная строка>, <Строка 1>, ..., <Строка N>)` — позволяет определить номер строки из списка `<Строка1>, ..., <СтрокаN>`, которая совпадает с исходной строкой:

```
SELECT FIELD('st', 'string', 'st', 'st2');
/* Выведет: 2 */
```

- `FIND_IN_SET(<Исходная строка>, <Список строк через запятую>)` — возвращает номер строки из списка `<Список строк через запятую>`, которая совпадает с исходной строкой:

```
SELECT FIND_IN_SET('st', 'string,st,st2');
/* Выведет: 2 */
```

- `REPLACE(<Строка>, <Подстрока для замены>, <Новая подстрока>)` — заменяет все вхождения подстроки на новую подстроку и возвращает результат:

```
SELECT REPLACE('Привет, Петя', 'Петя', 'Вася');
/* Выведет: Привет, Вася */
```

- `SUBSTRING_INDEX(<Строка>, <Подстрока>, <Номер вхождения>)` — находит *N*-е вхождение подстроки в строку, где *N* задается параметром `<Номер вхождения>`, и возвращает часть строки, расположенную слева от подстроки:

```
SELECT SUBSTRING_INDEX('синий, красный, зеленый', ',', 1);
/* Выведет: синий */
SELECT SUBSTRING_INDEX('синий, красный, зеленый', ',', 2);
/* Выведет: синий, красный */
```

Если параметр `<Номер вхождения>` имеет отрицательное значение, то ищется *N*-е вхождение подстроки с конца строки и возвращается часть строки, расположенная справа от найденной подстроки:

```
SELECT CONCAT('"', SUBSTRING_INDEX('синий, красный, зеленый',
                    ',', -1), '"');
/* Выведет: " зеленый" */
SELECT CONCAT('"', SUBSTRING_INDEX('синий, красный, зеленый',
                    ',', -2), '"');
/* Выведет: " красный, зеленый" */
```

- `INSERT(<Строка>, <Начальная позиция>, <Длина>, <Подстрока>)` — заменяет фрагмент в строке с начальной позиции длиной `<Длина>` на значение параметра `<Подстрока>`:

```
SELECT INSERT('красный', 6, 2, 'ое');
/* Выведет: красное */
SELECT INSERT('красный', 6, 1, 'ое');
/* Выведет: красноей */
```

- `QUOTE(<Строка>)` — экранирует все специальные символы в строке:

```
SELECT QUOTE("Д'Артаньян и три мушкетера");
/* Выведет: Д\'Артаньян и три мушкетера */
```

- `UNHEX(<Строка>)` — переводит строку из шестнадцатеричных цифр в обычную строку. Каждая пара символов в исходной строке воспринимается как шестнадцатеричное число, которое преобразуется в символ:

```
SELECT UNHEX('537472696E67');
/* Выведет: String */
```

- `COMPRESS(<Строка>)` — архивирует строку. Сжатую строку следует хранить в полях, имеющих бинарный тип данных;
- `UNCOMPRESS(<Строка>)` — разархивирует строку, сжатую функцией `COMPRESS()`;
- `UNCOMPRESSED_LENGTH(<Строка>)` — позволяет узнать длину строки, которую она будет иметь после разархивирования:

```
SELECT UNCOMPRESSED_LENGTH(COMPRESS('Строка'));
/* Выведет: 6 */
```

- `CHARSET(<Строка>)` — возвращает название кодировки для строки:

```
SET NAMES 'cp866';
SELECT CHARSET('Строка');
/* Выведет: cp866 */
```

- `COLLATION(<Строка>)` — возвращает порядок сортировки для строки:

```
SET NAMES 'cp866';
SELECT COLLATION('Строка');
/* Выведет: cp866_general_ci */
```

- `STRCMP(<Строка 1>, <Строка 2>)` — сравнивает две строки и возвращает:
  - `0` — если строки идентичны;
  - `-1` — если `<Строка1>` больше `<Строка2>`;
  - `1` — если `<Строка1>` меньше `<Строка2>`.

  Примеры:

```
SELECT STRCMP('Строка', 'Строка');
/* Выведет: 0 */
SELECT STRCMP('Строка1', 'Строка2');
/* Выведет: -1 */
SELECT STRCMP('Строка2', 'Строка1');
/* Выведет: 1 */
```

  Сравнение строк чувствительно к регистру;

- `LOAD_FILE(<Путь к файлу>)` — возвращает содержимое файла в виде строки. Часто используется для заполнения бинарных полей. В качестве примера создадим текстовый файл с названием test.txt в папке C:\Apache2. Затем запишем в файл строку "Content". Теперь получим содержимое файла с помощью функции `LOAD_FILE()`:

```
SELECT LOAD_FILE('C:/Apache2/test.txt');
/* Выведет: Content */
```

6.11.4. Функции для шифрования строк

Функции для необратимого шифрования:

- `MD5(<Строка>)` — кодирует строку по алгоритму MD5. Возвращает шестнадцатеричное число, содержащее 32 шестнадцатеричные цифры:

```
SELECT MD5('Пароль');
/* Выведет: 4a9866c3070171aa5a9faab83e61d887 */
```

  Этот алгоритм часто используется для кодирования паролей, т. к. не существует алгоритма для дешифровки. Для сравнения введенного пользователем пароля

с сохраненным в базе данных необходимо зашифровать введенный пароль, а затем провести сравнение;

- `PASSWORD(<Строка>)` — предназначена для шифрования паролей в таблице привилегий MySQL:

```
SELECT PASSWORD('Пароль');
/* Выведет: *63D191EAB71A4289F2473F5DE7E19E3C29DE9786 */
```

- `SHA(<Строка>)` и `SHA1(<Строка>)` — возвращают 40-разрядное шестнадцатеричное число:

```
SELECT SHA('Пароль');
/* Выведет: 8affbaf9a316d8b5500236c3daa1ce54a5a0385d */
SELECT SHA1('Пароль');
/* Выведет: 8affbaf9a316d8b5500236c3daa1ce54a5a0385d */
```

- `SHA2(<Строка>, <Длина результата>)` — возвращает шестнадцатеричное число, длина которого указана вторым параметром. В качестве длины результата можно указать значения 224, 256, 384, 512 или 0 (то же самое, что 256);

```
SELECT SHA2('Пароль', 0);
/* Выведет: b3fccb99c8d045f21cac8bb8851e277fa0120488465616c18 ↳
45d7ce81604a957 */
SELECT SHA2('Пароль', 512);
/* Выведет: b59a85e635c0582d8d50840379ae787f960303b2c38b5f05d ↳
ea3182b626db09d21e5d0d5d1ba6679ad5c7a56d8ac3c2969dbd54f91f81d ↳
368a8c87d2614bf20b */
```

- `ENCRYPT(<Строка>, [<Ключ>])` — использует для шифрования строки системную функцию `cript()`, имеющуюся в операционных системах UNIX. Если параметр `<Ключ>` не указан, то функция каждый раз будет возвращать разный результат. В операционной системе Windows функция всегда возвращает значение `NULL`.

Функции для симметричного шифрования:

- `AES_ENCRYPT(<Строка>, <Ключ>)` — принимает строку и секретный ключ и возвращает бинарную строку, зашифрованную по алгоритму AES;
- `AES_DECRYPT(<Зашифрованная строка>, <Ключ>)` — служит для расшифровки строк, зашифрованных функцией `AES_ENCRYPT()`:

```
SELECT AES_DECRYPT(AES_ENCRYPT('Пароль', 'Ключ'), 'Ключ');
/* Выведет: Пароль */
```

- `ENCODE(<Строка>, <Ключ>)` — принимает строку и секретный ключ и возвращает зашифрованную строку;
- `DECODE(<Зашифрованная строка>, <Ключ>)` — служит для расшифровки строк, зашифрованных функцией `ENCODE()`:

```
SELECT DECODE(ENCODE('Пароль', 'Ключ'), 'Ключ');
/* Выведет: Пароль */
```

Внимание!

Настоятельно не рекомендуется пользоваться функциями `ENCODE()` и `DECODE()`, т. к. реализуемый ими алгоритм шифрования крайне нестоек.

- `DES_ENCRYPT(<Строка>, [<Номер ключа>] | [<Ключ>])` — принимает строку и секретный ключ (или номер записи в ключевом DES-файле сервера). Возвращает зашифрованную строку. Если в MySQL не включена поддержка SSL, то функции `DES_ENCRYPT()` и `DES_DECRYPT()` возвращают `NULL`;
- `DES_DECRYPT(<Зашифрованная строка>, <Ключ>)` — служит для расшифровки строк, зашифрованных функцией `DES_ENCRYPT()`.

6.11.5. Информационные функции

Информационные функции:

- `VERSION()` — выводит информацию о версии сервера MySQL:

```
SELECT VERSION();
/* Выведет: 5.5.40 */
```

- `USER()` — позволяет узнать имя пользователя и имя хоста текущего пользователя:

```
SELECT USER();
/* Выведет: root@localhost */
```

- `CURRENT_USER()` — выдает имя пользователя и имя хоста текущего пользователя в сессии:

```
SELECT CURRENT_USER();
/* Выведет: root@localhost */
```

- `DATABASE()` — возвращает название текущей базы данных:

```
USE tests;
SELECT DATABASE();
/* Выведет: tests */
```

- `CONNECTION_ID()` — возвращает идентификатор соединения:

```
SELECT CONNECTION_ID();
/* Выведет: 200 */
```

- `DEFAULT(<Имя поля>)` — позволяет узнать значение по умолчанию для указанного поля:

```
USE tests;
CREATE TABLE `new_table` (
`id` INT AUTO_INCREMENT,
`count` INT DEFAULT 25,
PRIMARY KEY(`id`)
) ENGINE=MyISAM;
```

```
INSERT INTO `new_table` VALUES(NULL, 50);
SELECT DEFAULT(`count`) FROM `new_table` LIMIT 1;
/* Выведет: 25 */
```

- `LAST_INSERT_ID()` — служит для определения последнего автоматически сгенерированного значения для поля с атрибутом `AUTO_INCREMENT`. Значение возвращается только в том случае, если перед вызовом функции было сгенерировано новое значение:

```
INSERT INTO `new_table` VALUES(NULL, 80);
SELECT LAST_INSERT_ID();
/* Выведет: 2 */
```

Вывести последнюю добавленную запись можно следующими способами:

```
INSERT INTO `new_table` VALUES(NULL, 30);
SELECT `count` FROM `new_table` WHERE `id` = LAST_INSERT_ID();
/* Выведет: 30 */
INSERT INTO `new_table` VALUES(NULL, 90);
SELECT `count` FROM `new_table` WHERE `id` IS NULL;
/* Выведет: 90 */
```

- `FOUND_ROWS()` — позволяет узнать число строк, которое возвратил бы оператор `SELECT` без инструкции `LIMIT`.

Чтобы получить значение, необходимо в операторе `SELECT` указать опцию `SQL_CALC_FOUND_ROWS`:

```
INSERT INTO `new_table` VALUES(NULL, 6), (NULL, 70), (NULL, 50);
SELECT COUNT(*) FROM `new_table`;
/* Выведет: 7 */
SELECT SQL_CALC_FOUND_ROWS `count` FROM `new_table` LIMIT 0, 3;
/* Выведет три записи: 50, 80, 30 */
SELECT FOUND_ROWS();
/* Выведет: 7 */
```

- `BENCHMARK(<Число повторений>, <SQL-запрос>)` — выполняет SQL-запрос заданное число раз. Функция всегда возвращает значение 0. Применяется для определения быстродействия SQL-запроса:

```
SELECT BENCHMARK(1000000, MD5('строка'));
/* Выведет: 0
1 row in set (2.89 sec) */
```

6.11.6. Прочие функции

Также в SQL-запросах можно использовать следующие функции:

- `IF(<Условие>, <Если Истина>, <Если Ложь>)` — функция для логического выбора. Если `<Условие>` истинно, то возвращается значение выражения `<Если Истина>`, в противном случае возвращается значение выражения `<Если Ложь>`;

```
SELECT IF(5>6, 'Больше', 'Меньше');
/* Выведет: Меньше */
```

- CASE() — функция для логического выбора. Имеет две формы записи. Первая форма:

```
CASE <Переменная или выражение>
WHEN <Значение 1> THEN <Выражение 1>
[WHEN <Значение 2> THEN <Выражение 2>]
...
[ELSE <Выражение>] END
```

В зависимости от значения переменной (или выражения) выполняется один из блоков WHEN, в котором указано это значение.

Если ни одно из значений не описано в блоках WHEN, то выполняется блок ELSE:

```
SELECT CASE 3 + 5 WHEN 8 THEN 'Равно 8'
WHEN 7 THEN 'Равно 7' ELSE 'Не смогли определить' END;
/* Выведет: Равно 8 */
```

Вторая форма:

```
CASE WHEN <Условие 1> THEN <Выражение 1>
[WHEN <Условие 2> THEN <Выражение 2>]
...
[ELSE <Выражение>] END
```

Пример:

```
SELECT CASE WHEN 5>6 THEN 'Больше' ELSE 'Меньше' END;
/* Выведет: Меньше */
```

- IFNULL(<Выражение1>, <Выражение2>) — позволяет заменить значения NULL другими значениями. Если <Выражение1> не равно NULL, то функция возвращает <Выражение1>. В противном случае функция возвращает <Выражение2>:

```
SELECT IFNULL(5, 3);
/* Выведет: 5 */
SELECT IFNULL(NULL, 3);
/* Выведет: 3 */
```

- NULLIF(<Выражение1>, <Выражение2>) — функция для логического выбора. Если <Выражение1> равно <Выражение2>, возвращается значение NULL, в противном случае возвращается <Выражение1>:

```
SELECT NULLIF(5, 5);
/* Выведет: NULL */
SELECT NULLIF(5, 3);
/* Выведет: 5 */
```

- `INET_ATON(<IP-адрес>)` — представляет IP-адрес в виде целого числа:

```
SELECT INET_ATON('127.0.0.1');
/* Выведет: 2130706433 */
```

- `INET_NTOA(<IP-адрес в виде числа>)` — принимает IP-адрес в виде целого числа и возвращает IP-адрес в виде строки, состоящей из четырех цифр, разделенных точкой:

```
SELECT INET_NTOA(2130706433);
/* Выведет: 127.0.0.1 */
```

- `GET_LOCK(<Имя>, <Время ожидания ответа сервера>)` — устанавливает блокировку с указанным именем. Функция возвращает `1` в случае успешной блокировки и `0`, если время ожидания ответа сервера превысило величину, заданную в секундах параметром `<Время ожидания ответа сервера>`. Если произошла ошибка, то функция возвращает `NULL`. Блокировка снимается тремя способами:
  - с помощью функции `RELEASE_LOCK()`;
  - при повторном вызове функции `GET_LOCK()`;
  - при разрыве соединения с сервером.

  Например:

```
SELECT GET_LOCK('mylock', 5);
/* Выведет: 1 */
```

- `IS_FREE_LOCK(<Имя блокировки>)` — проверяет, свободна ли блокировка с указанным именем. Функция возвращает `1`, если блокировка свободна, и `0`, если она занята:

```
SELECT IS_FREE_LOCK('mylock');
/* Выведет: 0 */
```

- `IS_USED_LOCK(<Имя блокировки>)` — проверяет, установлена ли блокировка с указанным именем. Если блокировка установлена, то возвращается идентификатор соединения клиента, который установил блокировку:

```
SELECT IS_USED_LOCK('mylock');
/* Выведет: 1 */
SELECT CONNECTION_ID();
/* Выведет: 1 */
```

  Если блокировка не установлена, то возвращается значение `NULL`;

- `RELEASE_LOCK(<Имя блокировки>)` — снимает блокировку с указанным именем. Если блокировка успешно снята, то функция возвращает `1`. Если блокировка не может быть снята, то возвращается `0`. Если блокировка с указанным именем не существует, то функция возвращает `NULL`:

```
SELECT RELEASE_LOCK('mylock');
/* Выведет: 1 */
```

```
SELECT IS_USED_LOCK('mylock');
/* Выведет: NULL */
```

- `UUID()` — возвращает универсальный уникальный идентификатор — 128-разрядное уникальное число в виде строки, состоящее из пяти шестнадцатеричных чисел, разделенных символом -;

```
SELECT UUID();
/* Выведет: 9884721ded-2010-2ba9-71be-e337690000 */
SELECT UUID();
/* Выведет: a413be1fed-2010-2ba9-71be-e337690000 */
```

Используемый алгоритм гарантирует глобальную уникальность возвращенного идентификатора;

- `UUID_SHORT()` — возвращает сокращенный уникальный идентификатор — 64-разрядное уникальное число в виде строки;

```
SELECT UUID_SHORT();
/* Выведет: 23781672256274433 */
```

- `GROUP_CONCAT()` — объединяет отдельные значения в одну строку. Формат функции:

```
GROUP_CONCAT([DISTINCT] <Поле1> [, <ПолеN>]
[ORDER BY <Поле> [ASC | DESC]]
[SEPARATOR <Разделитель>]
)
```

Для примера создадим таблицу `concat_table` в базе данных tests.

```
CREATE TABLE `concat_table` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `counter` INT,
   PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим несколько записей:

```
INSERT INTO `concat_table` VALUES (NULL, 10);
INSERT INTO `concat_table` VALUES (NULL, 20);
INSERT INTO `concat_table` VALUES (NULL, 30);
INSERT INTO `concat_table` VALUES (NULL, 40);
INSERT INTO `concat_table` VALUES (NULL, 20);
```

Теперь продемонстрируем возможности функции `GROUP_CONCAT()`. Выведем все значения поля `counter`:

```
SELECT GROUP_CONCAT(`counter`) FROM `concat_table`;
/* Выведет: 10,20,30,40,20 */
```

А теперь выведем только уникальные значения, отсортированные в порядке убывания:

```
SELECT GROUP_CONCAT(DISTINCT `counter` ORDER BY `counter` DESC)
FROM `concat_table`;
/* Выведет: 40,30,20,10 */
```

И, наконец, выведем все значения поля `counter` больше 10 через разделитель "; ":

```
SELECT GROUP_CONCAT(DISTINCT `counter`
ORDER BY `counter` ASC SEPARATOR '; ')
FROM `concat_table` WHERE `counter` > 10;
/* Выведет: 20; 30; 40 */
```

6.12. Переменные SQL

Результат текущего запроса можно сохранить в переменной и использовать в последующих запросах в рамках одного сеанса. Присвоить значение переменной можно следующими способами:

- с помощью оператора `SELECT`:

```
SELECT @time := NOW();
/* Выведет: 2014-12-04 16:08:34 */
SELECT @time;
/* Выведет: 2014-12-04 16:08:34 */
```

- с помощью оператора `SET`:

```
SET @time = NOW();
SELECT @time;
/* Выведет: 2014-12-04 16:08:34 */
```

Объявление переменной начинается с символа `@`, а сохранить значение в переменной позволяет оператор `:=`. Обратите внимание, что в случае применения оператора `SET` вместо оператора `:=` можно использовать оператор `=`. Следует также помнить, что имя переменной зависит от регистра символов.

В качестве примера создадим таблицу `var_table` в базе данных tests:

```
CREATE TABLE `var_table` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `name_tovar` VARCHAR(255),
   `count` INT,
   PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Поле `name_tovar` предназначено для хранения названия товара, а поле `count` служит для обозначения количества товара на складе.

Добавим несколько записей:

```
INSERT INTO `var_table` VALUES (NULL, 'Монитор', 10);
INSERT INTO `var_table` VALUES (NULL, 'Клавиатура', 20);
```

```
INSERT INTO `var_table` VALUES (NULL, 'Мышь', 30);
INSERT INTO `var_table` VALUES (NULL, 'Тюнер', 40);
INSERT INTO `var_table` VALUES (NULL, 'HDD', 20);
```

Сохраним в переменной минимальное количество товара на складе, а затем выведем название товара с минимальным количеством:

```
SELECT @min := MIN(`count`) FROM `var_table`;
/* Выведет: 10 */
SELECT `name_tovar` FROM `var_table` WHERE `count` = @min;
/* Выведет: Монитор */
```

Если запрос вернет более одного варианта, то в переменной сохранится только последнее значение:

```
SELECT @min := `count` FROM `var_table`;
/* Выведет:
  10
  20
  30
  40
  20 */
SELECT @min;
/* Выведет: 20 */
```

6.13. Временные таблицы

Временные таблицы создаются с помощью оператора `CREATE TEMPORARY TABLE`, синтаксис которого ничем не отличается от оператора `CREATE TABLE`. Заполнить временную таблицу можно обычным способом, но чаще всего это делают с помощью вложенных запросов. Например, временные таблицы используются для реализации дополнительного поиска в результатах выполнения запроса. Следует помнить, что имя временной таблицы действительно только в течение текущего соединения с сервером. После завершения соединения с сервером временная таблица автоматически удаляется.

В качестве примера создадим таблицу `user_table` в базе данных tests.

```
CREATE TABLE `user_table` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(255),
   PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

В поле `name` мы будем хранить фамилию и имя пользователя. Добавим в таблицу несколько записей:

```
INSERT INTO `user_table` VALUES (NULL, 'Иванов Сергей');
INSERT INTO `user_table` VALUES (NULL, 'Иванов Николай');
```

```
INSERT INTO `user_table` VALUES (NULL, 'Иванов Иван');
INSERT INTO `user_table` VALUES (NULL, 'Петров Александр');
INSERT INTO `user_table` VALUES (NULL, 'Петров Николай');
INSERT INTO `user_table` VALUES (NULL, 'Иванов Максим');
```

А теперь инсценируем ситуацию поиска в найденном с помощью временных таблиц. Предположим, что первоначальный запрос пользователя выводит клиентов с фамилией Иванов:

```
SELECT `id`, `name` FROM `user_table` WHERE `name` LIKE '%Иванов%';
```

Сохраним результат запроса во временной таблице, а затем выведем клиентов только с именем Николай:

```
CREATE TEMPORARY TABLE `temp`
SELECT `id`, `name` FROM `user_table` WHERE `name` LIKE '%Иванов%';
SELECT `name` FROM `temp` WHERE `name` LIKE '%Николай%';
/* Выведет: Иванов Николай */
```

Обратите внимание: при использовании вложенных запросов не нужно определять структуру временной таблицы. По умолчанию структура временной таблицы будет такой же, как и в результирующей таблице. Посмотреть структуру временной таблицы можно с помощью оператора `DESCRIBE`:

```
CREATE TEMPORARY TABLE `temp2`
SELECT `id`, `name` FROM `user_table` WHERE `name` LIKE '%Иванов%';
DESCRIBE `temp2`;
```

Удалить временную таблицу можно следующими способами:

- с помощью оператора `DROP TABLE`:

  ```
  DROP TABLE <Имя временной таблицы>;
  ```

- по завершении соединения с сервером временная таблица будет удалена автоматически.

6.14. Вложенные запросы

Для изучения вложенных запросов создадим в базе данных tests следующие таблицы:

- `users_table` — для хранения данных о клиентах:

  ```
  CREATE TABLE `users_table` (
     `id_user` INT NOT NULL AUTO_INCREMENT,
     `name` VARCHAR(255),
     PRIMARY KEY (`id_user`)
  ) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
  ```

- `product_table` — для хранения данных о товарах:

  ```
  CREATE TABLE `product_table` (
     `id_product` INT NOT NULL AUTO_INCREMENT,
  ```

```
    `name_product` VARCHAR(255),
    PRIMARY KEY (`id_product`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

- `orders_table` — для хранения сведений о покупках:

```
CREATE TABLE `orders_table` (
    `id_orders` INT NOT NULL AUTO_INCREMENT,
    `id_product` INT,
    `id_user` INT,
    `count` INT,
    PRIMARY KEY (`id_orders`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Добавим в таблицы несколько записей:

```
INSERT INTO `users_table` VALUES (1, 'Иванов');
INSERT INTO `users_table` VALUES (2, 'Петров');

INSERT INTO `product_table` VALUES (1, 'Монитор');
INSERT INTO `product_table` VALUES (2, 'Клавиатура');
INSERT INTO `product_table` VALUES (3, 'Мышь');

INSERT INTO `orders_table` VALUES (1, 1, 1, 2);
INSERT INTO `orders_table` VALUES (2, 3, 2, 5);
INSERT INTO `orders_table` VALUES (3, 2, 1, 1);
```

6.14.1. Заполнение таблицы с помощью вложенного запроса

При изучении временных таблиц мы уже использовали вложенный запрос для заполнения временной таблицы. Заполнять можно не только временные таблицы, но и обычные таблицы, создаваемые посредством оператора `CREATE TABLE`. Создадим таблицу `orders_item_table` с помощью вложенного запроса:

```
CREATE TABLE `orders_item_table` (
    `id_orders` INT NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (`id_orders`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251
SELECT `orders_table`.`id_orders` AS `id_orders`,
`users_table`.`name` AS `user`,
`product_table`.`name_product` AS `name`,
`orders_table`.`count` AS `count`
FROM `users_table`, `product_table`, `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user` AND
`orders_table`.`id_product` = `product_table`.`id_product`;
```

Выведем структуру созданной таблицы с помощью SQL-запроса:

```
DESCRIBE `orders_item_table`\G
```

Эта команда SQL выведет:

```
*************************** 1. row ***************************
Field: id_orders
Type: int(11)
Null: NO
Key: PRI
Default: NULL
Extra: auto_increment
*************************** 2. row ***************************
Field: user
Type: varchar(255)
Null: YES
Key:
Default: NULL
Extra:
*************************** 3. row ***************************
Field: name
Type: varchar(255)
Null: YES
Key:
Default: NULL
Extra:
*************************** 4. row ***************************
Field: count
Type: int(11)
Null: YES
Key:
Default: NULL
Extra:
```

Как видно из результата, столбцы, не определенные в операторе `CREATE TABLE`, но имеющиеся в результирующей таблице, добавляются в новую таблицу. Если столбцы определены в операторе `CREATE TABLE`, но отсутствуют во вложенном запросе, то они получают значение по умолчанию.

Использовать вложенные запросы можно и в операторе `INSERT`. Создадим таблицу `orders_item2_table` обычным образом:

```
CREATE TABLE `orders_item2_table` (
   `id_orders` INT NOT NULL AUTO_INCREMENT,
   `user` VARCHAR(255),
   `name` VARCHAR(255),
   `count` INT,
   PRIMARY KEY (`id_orders`)
) ENGINE=MyISAM DEFAULT CHARSET=cp1251;
```

Затем добавим записи с помощью оператора `INSERT` и вложенного запроса:

```
INSERT IGNORE INTO `orders_item2_table`
SELECT `orders_table`.`id_orders` AS `id_orders`,
```

```
`users_table`.`name` AS `user`,
`product_table`.`name_product` AS `name`,
`orders_table`.`count` AS `count`
FROM `users_table`, `product_table`, `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user` AND
`orders_table`.`id_product` = `product_table`.`id_product`;
```

Ключевое слово `IGNORE` сообщает серверу, что записи с повторяющимися значениями первичного ключа и уникальных индексов должны отбрасываться. (Если не указать это ключевое слово, то при попытке вставить в таблицу повторяющуюся запись возникнет ошибка.)

Если мы хотим, чтобы записи с повторяющимися значениями ключа заменяли собой уже существующие в таблице записи, то применим оператор `REPLACE`:

```
REPLACE INTO `orders_item2_table`
SELECT `orders_table`.`id_orders` AS `id_orders`,
`users_table`.`name` AS `user`,
`product_table`.`name_product` AS `name`,
`orders_table`.`count` AS `count`
FROM `users_table`, `product_table`, `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user` AND
`orders_table`.`id_product` = `product_table`.`id_product`;
```

6.14.2. Применение вложенных запросов в инструкции *WHERE*

Выведем имя пользователя, сделавшего заказ под номером 2, с помощью вложенного запроса:

```
SELECT `name` FROM `users_table`
WHERE `id_user` = (SELECT `id_user` FROM `orders_table`
WHERE `id_orders` = 2);
/* Выведет: Петров */
```

В данном примере мы объединили два запроса в один. Внутренний запрос возвращает идентификатор пользователя, сделавшего заказ с номером 2, а внешний запрос по этому идентификатору получает имя пользователя. Как видно из примера, вложенный запрос всегда заключается в круглые скобки.

Уровень вложенности запроса может быть более двух. Однако на практике запросы с уровнем вложенности более трех нецелесообразны, т. к. это приводит к увеличению времени выполнения запроса.

Если вложенный запрос возвращает более одного значения, то MySQL генерирует ошибку. Обойти эту проблему можно следующими способами:

- указать ключевые слова `IN` или `NOT IN`:

```
SELECT `name` FROM `users_table`
WHERE `id_user` IN (SELECT `id_user` FROM `orders_table`);
```

Этот пример выведет

```
Иванов
Петров
```

- использовать ключевые слова ANY или SOME:

```
SELECT `name` FROM `users_table`
WHERE `id_user` > ANY (SELECT `id_user` FROM `orders_table`);
/* Выведет: Петров */
```

- задать ключевое слово ALL:

```
SELECT `name` FROM `users_table`
WHERE `id_user` <= ALL (SELECT `id_user` FROM `orders_table`);
/* Выведет: Иванов */
```

При указании ключевого слова IN проверяется совпадение с одним из значений, возвращаемых вложенным запросом. При использовании ключевого слова NOT IN, наоборот, проверяется отсутствие совпадения со списком значений. Если применяется ключевые слова ANY или SOME, то проверяемое значение поочередно сравнивается с каждым элементом, и если хотя бы одно сравнение возвращает значение Истина, то результат попадает в итоговую таблицу. Если задано ключевое слово ALL, то в результирующую таблицу попадут значения, только если все сравнения вернут значение Истина.

С помощью ключевого слова EXISTS можно проверить, имеется ли хоть одна строка в результирующей таблице. Если вложенный запрос дает ненустой результат, то ключевое слово EXISTS возвращает 1 (истина). В противном случае возвращается значение 0 (ложь). Получить противоположные значения позволяет ключевое слово NOT EXISTS.

В качестве примера выведем фамилии всех клиентов, сделавших хотя бы один заказ. Для наглядности добавим в таблицу users_table еще одного клиента:

```
INSERT INTO `users_table` VALUES (3, 'Сидоров');
```

Теперь выполним такой запрос:

```
SELECT `name` FROM `users_table`
WHERE EXISTS (SELECT * FROM `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user`);
```

В результате мы получим:

```
Иванов
Петров
```

А теперь выведем фамилии клиентов, не сделавших ни одного заказа:

```
SELECT `name` FROM `users_table`
WHERE NOT EXISTS (SELECT * FROM `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user`);
```

Этот запрос вернет

```
Сидоров
```

Обратите внимание, внутри вложенного запроса мы указываем поле таблицы `users_table.id_user` из внешнего запроса. Такая связь называется *внешней ссылкой*, а сам запрос называется *коррелированным вложенным запросом*.

6.14.3. Применение вложенных запросов в инструкции *FROM*

Вложенные запросы можно использовать и в инструкции `FROM`. При этом мы фактически выполним выборку данных, возвращенных вложенным запросом.

Для примера давайте рассмотрим созданную в *разд. 6.13* таблицу `user_table`, хранящую список пользователей. Ранее для выполнения сложного поиска мы сначала перенесли предварительно отобранные записи во временную таблицу, в которой потом выполняли окончательный поиск. Но ту же операцию можно реализовать гораздо проще — указав в инструкции `FROM` вложенный запрос:

```
SELECT `name` FROM
(SELECT `name` FROM `user_table` WHERE `name` LIKE '%Иванов%') AS `users`
WHERE `name` LIKE '%Николай%';
/* Выведет: Иванов Николай */
```

Здесь вложенный запрос выполняет предварительную выборку записей, включающих слово "Иванов". Основной запрос уже производит окончательный поиск записей, найденных вложенным запросом и хранящих слово "Николай".

Отметим, что для вложенного запроса всегда следует указывать псевдоним, применив инструкцию `AS`.

6.15. Внешние ключи

При эксплуатации реляционной базы данных время от времени необходимо изменять ее структуру или удалять устаревшие данные. Например, для увеличения быстродействия можно удалить учетные записи клиентов, которые не совершали покупок в течение определенного срока. Если просто удалить этих клиентов из одной таблицы, то это может привести к нарушению ссылочной целостности базы данных, т. к. кто-нибудь из удаляемых клиентов наверняка совершал ранее покупки, а значит, сведения о покупке заносились в таблицу заказов. По этой причине при удалении клиента необходимо предварительно изъять все записи о совершенных им покупках из таблицы заказов. Сделать это можно с помощью двух SQL-запросов. Первый запрос удаляет записи из таблицы заказов, а второй — удаляет запись о клиенте.

Для таблиц типа `InnoDB` предусмотрена возможность автоматического контроля над ссылочной целостностью базы данных с помощью внешних ключей.

Внешний ключ указывает, что поле или комбинация полей текущей таблицы содержат ссылку на другую таблицу. Его можно добавить при создании таблицы с помощью оператора `CREATE TABLE`, а оператор `ALTER TABLE` позволяет добавить внешний ключ в уже существующую таблицу.

Добавляется внешний ключ с помощью конструкции `FOREIGN KEY`. Формат конструкции:

```
FOREIGN KEY [<Имя ключа>] (<Список полей в текущей таблице>)
REFERENCES <Имя внешней таблицы> (<Список полей во внешней таблице>)
[ON DELETE <Действие>]
[ON UPDATE <Действие>]
```

В параметре `<Действие>` могут быть указаны значения:

- `CASCADE` — удаление или изменение записи, содержащей первичный ключ, приведет к автоматическому удалению или изменению соответствующих записей в таблице-потомке;
- `SET NULL` — при удалении или изменении записи, содержащей первичный ключ, соответствующие записи в таблице-потомке получат значение `NULL`;
- `RESTRICT` — нельзя удалить или изменить запись, пока в таблице-потомке существуют ссылающиеся записи;
- `NO ACTION` — то же самое, что и `RESTRICT`.

Если действие не указано, это равносильно указанию действия `RESTRICT`.

Для примера создадим в базе данных tests следующие таблицы:

- `users_foreign` — для хранения данных о клиентах:

```
CREATE TABLE `users_foreign` (
   `id_user` INT NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(255),
   PRIMARY KEY (`id_user`)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

- `product_foreign` — для хранения данных о товарах:

```
CREATE TABLE `product_foreign` (
   `id_product` INT NOT NULL AUTO_INCREMENT,
   `name_product` VARCHAR(255),
   PRIMARY KEY (`id_product`)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

- `orders_foreign` — для хранения сведений о покупках:

```
CREATE TABLE `orders_foreign` (
   `id_orders` INT NOT NULL AUTO_INCREMENT,
   `id_product` INT,
   `id_user` INT,
   `count` INT,
```

```
    PRIMARY KEY (`id_orders`),
    FOREIGN KEY (`id_user`) REFERENCES `users_foreign` (`id_user`)
    ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (`id_product`)
    REFERENCES `product_foreign` (`id_product`)
    ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

Добавим в таблицы несколько записей:

```
INSERT INTO `users_foreign` VALUES (1, 'Иванов');
INSERT INTO `users_foreign` VALUES (2, 'Петров');

INSERT INTO `product_foreign` VALUES (1, 'Монитор');
INSERT INTO `product_foreign` VALUES (2, 'Клавиатура');
INSERT INTO `product_foreign` VALUES (3, 'Мышь');
INSERT INTO `orders_foreign` VALUES (1, 1, 1, 2);
INSERT INTO `orders_foreign` VALUES (2, 3, 2, 5);
INSERT INTO `orders_foreign` VALUES (3, 2, 1, 1);
```

Теперь попробуем удалить господина Иванова из таблицы `users_foreign`:

```
DELETE FROM `users_foreign` WHERE `id_user`=1;
SELECT `name` FROM `users_foreign`;
/* Выведет: Петров */
```

Посмотрим, сколько заказов осталось в таблице `orders_foreign`:

```
SELECT `id_orders` FROM `orders_foreign`;
/* Выведет: 2 */
```

Как видно из этого примера, удаление господина Иванова привело к автоматическому удалению его заказа за счет применения ключевого слова `CASCADE`. Теперь попробуем добавить заказ на имя уже не существующего в базе данных господина Иванова:

```
INSERT INTO `orders_foreign` VALUES (NULL, 1, 1, 2);
```

В итоге получим ошибку:

```
#1452 - Cannot add or update a child row: a foreign key constraint fails
```

Попробуем удалить товар с номером 3 из таблицы `product_foreign`.

```
DELETE FROM `product_foreign` WHERE `id_product`=3;
```

В итоге также получим ошибку:

```
#1451 - Cannot delete or update a parent row: a foreign key constraint fails
```

Иными словами, пока мы не удалим заказ с номером 2 из таблицы `orders_foreign`, мы не сможем удалить товар с номером 3 из таблицы `product_foreign`. Это достигается за счет применения ключевого слова `RESTRICT`.

6.16. Транзакции

При работе с базами данных очень часто бывает необходимо выполнять какие-либо сложные действия, включающие несколько операций по добавлению, изменению и удалению записей. И эти операции обязательно должны быть либо выполнены все и полностью, либо, если в процессе их обработки произойдет сбой, не выполнены все и полностью — в противном случае нарушится целостность данных, хранящихся в базе. Действия подобного рода называются *транзакциями*.

MySQL предлагает развитую поддержку транзакций для таблиц типа `InnoDB`. Для таблиц `MyISAM` транзакции не поддерживаются.

6.16.1. Запуск, подтверждение и отмена транзакций

Запустить транзакцию можно командой `START TRANSACTION`, `BEGIN` или `BEGIN WORK`:

```
START TRANSACTION;
```

Все операции выборки, добавления, изменения и удаления записей, следующие после этой команды, будут выполняться в составе транзакции.

Для завершения транзакции следует отдать команду `COMMIT` или `COMMIT WORK`:

```
COMMIT;
```

При получении этой команды сервер выполняет все команды изменения записей, что находились между ней и предыдущей командой зануска транзакции, и подтверждает все сделанные этими командами изменения. Поэтому такое действие называется *подтверждением транзакции*.

Если же в процессе обработки включенных в состав транзакции команд возникнет ошибка, действия, выполненные всеми этими командами, будут отменены, и база данных вернется в изначальное состояние, предшествующее моменту запуска транзакции.

Иногда возникает потребность принудительно отменить транзакцию и, соответственно, изменения, которые произвели с данными все заключенные в нее команды (выполнить *откат транзакции*). Для этого используются идентичные команды `ROLLBACK` и `ROLLBACK WORK`:

```
ROLLBACK;
```

Для примера создадим в базе данных tests две таблицы. Первая (`inv_goods`) будет хранить список товаров и их количество, имеющееся на складе, а вторая (`inv_cns`) — список расходных ведомостей, документирующих передачу товаров со склада заказчикам:

```
CREATE TABLE `inv_goods` (
   `id_good` INT NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(10),
   `count` SMALLINT,
   PRIMARY KEY (`id_good`)
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

```
CREATE TABLE `inv_cns` (
   `id_cn` INT NOT NULL AUTO_INCREMENT,
   `id_good` INT,
   `count` SMALLINT,
   PRIMARY KEY (`id_cn`),
   FOREIGN KEY (`id_good`) REFERENCES `inv_goods` (`id_good`)
   ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

Добавим пару записей в таблицу `inv_goods`:

```
INSERT INTO `inv_goods` VALUES (NULL, 'Мыло', 20);
INSERT INTO `inv_goods` VALUES (NULL, 'Шило', 10);
```

Выясним, какой идентификатор получил товар "мыло":

```
SELECT `id_good` FROM `inv_goods` WHERE `name` = "Мыло";
/* Выведет: 1 */
```

Теперь давайте попробуем выписать расходную ведомость на отпуск трех единиц этого товара. Для этого нам придется:

- добавить в таблицу `inv_cns` запись, представляющую саму расходную ведомость;
- уменьшить значение поля `count` таблицы `inv_goods` на три, чтобы показать уменьшение хранящегося на складе товара.

Поскольку обе эти операции обязательно должны быть выполнены все и полностью (в противном случае, если случится сбой, мы получим недостачу товара на складе), мы заключим их в транзакцию и не забудем подтвердить ее:

```
START TRANSACTION;
INSERT INTO `inv_cns` VALUES (NULL, 1, 3);
UPDATE `inv_goods` SET `count`=`count`-3 WHERE `id_good`=1;
COMMIT;
```

Посмотрим содержимое таблицы `inv_cns`:

```
SELECT * FROM `inv_cns`;
```

Мы увидим, что в таблице появилась одна запись — представляющая только что выписанную нами расходную ведомость.

И проверим, произошла ли выдача товара со склада:

```
SELECT `count` FROM `inv_goods` WHERE `name`="Мыло";
/* Выведет: 17 */
```

Как видим, товар был успешно выдан.

А теперь сымитируем возникновение ошибки при выполнении заключенных в транзакцию операций, вставив в представленный код команду, не поддерживаемую MySQL:

```
START TRANSACTION;
INSERT INTO `inv_cns` VALUES (NULL, 1, 3);
LJUUHG;
UPDATE `inv_goods` SET `count`=`count`-3 WHERE `id_good`=1;
COMMIT;
```

В этом случае мы получим сообщение об ошибке. А проверив содержимое таблиц `inv_goods` и `inv_cns`, убедимся, что никаких изменений в него внесено не было.

И, наконец, предположим, что мы собирались выписать расходную ведомость на две единицы товара "шило", но в последний момент передумали и решили отменить выдачу, выполнив откат транзакции:

```
START TRANSACTION;
SELECT @inv_id:=`id_good` FROM `inv_goods` WHERE `name`="Шило";
INSERT INTO `inv_cns` VALUES (NULL, @inv_id, 2);
UPDATE `inv_goods` SET `count`=`count`-2 WHERE `id_good`=@inv_id;
ROLLBACK;
```

И в этом случае все операции, заключенные в транзакцию, не будут выполнены. В чем мы убедимся, просмотрев содержимое таблиц `inv_goods` и `inv_cns`.

Отметим, что в последнем примере для временного хранения идентификатора товара, выдаваемого со склада, мы использовали переменную SQL.

6.16.2. Изоляция транзакций

Поскольку к базе данных MySQL одновременно могут подключаться сразу несколько пользователей, в ней могут выполняться сразу несколько транзакций, запущенных разными пользователями. И эти транзакции могут изменять содержимое одной и той же таблицы.

Введение в изоляцию транзакций

В связи с этим возникает вопрос: как сделанные в таблице изменения отслеживаются командами `SELECT`, находящимися в составе транзакции? В этом случае MySQL соблюдает следующие правила:

- При выполнении первой команды `SELECT`, присутствующей в транзакции, в памяти компьютера создается своего рода "снимок" данных, являющихся актуальными на момент выполнения этой операции. Все последующие команды `SELECT`, что включены в транзакцию, оперируют именно этим "снимком".
- "Снимок" отслеживает все действия команд `INSERT`, `UPDATE` и `DELETE`, находящихся в той же транзакции, где он был создан.
- Действия команд `INSERT`, `UPDATE` и `DELETE`, присутствующих в других транзакциях, "снимок" не отслеживает.

Как видим, каждая транзакция работает со своей копией данных, хранящейся в оперативной памяти. Такой подход называется *изоляцией транзакций*.

Проиллюстрируем сказанное на примере. Пусть у нас имеется пустая таблица `tbl` с двумя полями. К базе данных, хранящей эту таблицу, подключились два пользователя и запустили две отдельные транзакции. Результат их выполнения можно увидеть в табл. 6.15.

Таблица 6.15. *Пример, иллюстрирующий принцип изоляции транзакций*

Транзакция 1	Транзакция 2
`START TRANSACTION;`	`START TRANSACTION;`
`SELECT COUNT() FROM `tbl`;` `/* Выведет: 0 */`	
	`INSERT INTO `tbl` VALUES (1, 1);`
`SELECT COUNT() FROM `tbl`;` `/* Выведет: 0 */`	
	`COMMIT;`
`SELECT COUNT() FROM `tbl`;` `/* Выведет: 0 */`	
`COMMIT;`	
`SELECT COUNT() FROM `tbl`;` `/* Выведет: 1 */`	

Здесь сразу видно, что команды выборки записей, присутствующие в первой транзакции, "живут" будто бы в своем собственном мире, оперируя сделанным самой первой командой "снимком".

Если в транзакции присутствует команда выборки данных `SELECT`, ее рекомендуется поставить самой первой, сразу же после команды `START TRANSACTION`:

```
START TRANSACTION;
SELECT . . .
/* Остальные команды транзакции */
COMMIT | ROLLBACK;
```

В этом случае "снимок" данных будет создан сразу же после запуска транзакции.

Как вариант можно использовать команду `START TRANSACTION WITH CONSISTENT SNAPSHOT` — она указывает MySQL создать "снимок" данных непосредственно при запуске транзакции:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
/* Команды транзакции */
COMMIT | ROLLBACK;
```

Уровни изоляции транзакций

MySQL поддерживает четыре уровня изоляции транзакции, характеризующие степень доступности таблиц, вовлеченных в транзакцию, для других транзакций, ко-

торые запущены параллельно с текущей. Для указания уровня изоляции применяется команда SET TRANSACTION ISOLATION LEVEL:

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED |
SERIALIZABLE
```

Рассмотрим все доступные нам уровни изоляции транзакций:

- REPEATABLE READ — первая команда SELECT в транзакции создает "снимок" данных, который используется всеми последующими командами такого же типа. Транзакции изолированы друг от друга. Это уровень изоляции по умолчанию.
- READ COMMITTED — каждая команда SELECT в транзакции создает свой собственный "снимок" данных, которым и оперирует. Транзакции изолированы друг от друга.
- READ UNCOMMITTED — транзакции не изолированы друг от друга; при этом команда SELECT может прочитать данные, в которые были внесены изменения параллельно работающими транзакциями. Вследствие чего может возникпуть ситуация, когда одна транзакция может прочитать данные, измененные другой транзакцией, после чего, в случае отката последней первая транзакция будет оперировать данными, которых фактически нет в таблице.
- SERIALIZABLE — то же самое, что и REPEATABLE READ, но, если автозавершение транзакций отключено (как это сделать, мы узнаем чуть позже), каждая команда SELECT временно блокирует таблицы, из которых выбирает данные, с тем, чтобы никакая другая транзакция не смогла получить к ним доступ.

Пример:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
/* Команды транзакции */
COMMIT;
```

Кроме того, мы можем указать, на какие транзакции будет распространяться действие команды SET TRANSACTION ISOLATION LEVEL:

- если не указаны ключевые слова GLOBAL и SESSION, действие команды распространится лишь на следующую транзакцию. Все транзакции, запущенные после нее, получат уровень изоляции по умолчанию:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
/* Эта транзакция использует уровень изоляции READ COMMITTED */
COMMIT;
START TRANSACTION;
/* Эта транзакция будет запущена с уровнем изоляции по умолчанию -
REPEATABLE READ */
COMMIT;
```

- если указано ключевое слово SESSION, действие команды распространится на все транзакции, запущенные в течение текущей сессии;

```
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
/* Все последующие транзакции, запущенные в пределах текущей сессии,
используют уровень изоляции READ UNCOMMITTED */
```

- если указано ключевое слово GLOBAL, действие команды распространится на все транзакции, запущенные в течение текущей и последующих сессий.

6.16.3. Автозавершение транзакций и его отключение

MySQL, как и все прочие серверы баз данных, каждую операцию запускает в составе транзакции, даже если мы не указали это явно. Такие создаваемые по умолчанию транзакции подтверждаются автоматически (*автозавершение транзакций*):

```
/* Запускается транзакция по умолчанию */
SELECT * from `inv_cns`;
/* Транзакция по умолчанию подтверждается*/
. . .
/* Запускается транзакция по умолчанию */
INSERT INTO `inv_cns` VALUES (NULL, 1, 2);
/* Транзакция по умолчанию подтверждается*/
```

Как видим, это относится, в том числе, и к операциям выборки данных.

Но как только мы отдадим команду на запуск транзакции, автозавершение отключается. И мы должны сами завершить транзакцию, подтвердив или откатив ее:

```
START TRANSACTION;
/* Автозавершение транзакций отключается */
INSERT INTO `inv_cns` VALUES (NULL, 1, 2);
/* Выполняем завершение транзакции вручную */
COMMIT;
```

Мы также можем отключить автозавершение транзакций самостоятельно, присвоив системной переменной MySQL autocommit значение 0:

```
SET autocommit=0;
/* Автозавершение транзакций отключено */
INSERT INTO `inv_cns` VALUES (NULL, 1, 2);
/* Выполняем завершение транзакции */
COMMIT;
```

Чтобы вновь включить автозавершение, следует присвоить переменной autocommit значение 1:

```
SET autocommit=1;
/* Автозавершение транзакций включено */
INSERT INTO `inv_cns` VALUES (NULL, 1, 2);
/* Завершать транзакцию вручную нет необходимости */
```

6.16.4. Поддержка транзакций библиотекой php_mysqli.dll

Библиотека php_mysqli.dll предлагает три функции и три метода для управления транзакциями. Они позволят нам несколько упростить программный код.

> ***Внимание!***
>
> Библиотека php_mysql.dll не предоставляет никаких инструментов для работы с транзакциями. Нам придется посылать соответствующие команды серверу непосредственно, с помощью функции `mysql_query()`.

При использовании процедурного стиля доступны следующие функции:

- `mysqli_commit(<Идентификатор>)` — подтверждает транзакцию:

```
if (@$db = mysqli_connect("localhost", "root", "123456", "tests")) {
   mysqli_query($db, 'START TRANSACTION');
   mysqli_query($db, 'INSERT INTO `inv_cns` VALUES (NULL, 1, 2)');
   mysqli_commit($db);
   mysqli_close($db);
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `mysqli_rollback(<Идентификатор>)` — отменяет транзакцию;
- `mysqli_autocommit(<Идентификатор>, true|false)` — включает (`true`) или отключает (`false`) автозавершение транзакций.

Поклонники объектного стиля будут пользоваться вот такими методами:

- `commit()` — подтверждает транзакцию:

```
@$db = new mysqli("localhost", "root", "123456", "tests");
if (!mysqli_connect_errno()) {
   $db->query('START TRANSACTION');
   $db->query('INSERT INTO `inv_cns` VALUES (NULL, 1, 2)');
   $db->commit();
   $db->close();
}
else {
   echo "Не удалось установить подключение к базе данных";
}
```

- `rollback()` — отменяет транзакцию;
- `autocommit(true|false)` — включает (`true`) или отключает (`false`) автозавершение транзакций.

Как видим, отдельной функции или метода для запуска транзакции php_mysqli.dll не предусматривает, и нам приходится самим посылать на сервер команду `START TRANSACTION`.

ПРИЛОЖЕНИЕ

Описание электронного архива

По ссылке **ftp://ftp.bhv.ru/9785977531306.zip** можно скачать электронный архив со следующими материалами:

- глава 7 "Публикация сайта";
- описание процесса установки и настройки специализированных редакторов;
- дополнительные руководства;
- описание процесса разработки полнофункционального Web-сайта с использованием всех изученных технологий;
- все листинги, встречающиеся в тексте книги;
- видеоуроки к книге.

Ссылка доступна и со страницы книги на сайте **www.bhv.ru**.

Предметный указатель

U

W

X

A

Б

В

Г

Д

Е

З

И

К

Л

М

H

O

П

P

C

T

У

Ф

Х

Ц

Ш

Э

Я