ЗАО "БелХард Групп"
Центр обучающих технологий

Михалькевич А.В.

PHP. Практика создания сайтов.

методическое пособие

Минск 2012

УДК 004.415.53
ББК 32.973.2-018.2я7

Рецензент:

Об авторе:
Михалькевич Александр Викторович, программист-практик.

А.В.Михалькевич
PHP. Практика создания сайтов.

А.В.Михалькевич, Закрытое акционерное общество «БелХард Групп» - Мн., 2012. – 183с.

ISBN

В книге отображены следующие аспекты программирования:

- Браузерное (HTML и HTML5, CSS, jQuery).
- Серверное (PHP, PHP+MySQL). Разработка сайта визитки, системы администрирования для сайта, поиска, обработка изображений, регистрация и аутентификация пользователей.
- Серверно-браузерное (ajax).
- Трехуровневая архитектура данных (логика приложения, логика данных и логика представления).
- Объектно-ориентированное программирования на PHP. PDO.
- XML.
- Защита PHP

Пособие рекомендовано к использованию слушателям курсов ЦОТ ЗАО "БелХард Групп".

УДК 004.415.53
ББК 32.973.2-018.2я7

ISBN

Оглавление

1. Основы HTML. Знать обязательно.

HTML - Это язык гипертекстовой разметки.

Структура тэга в HTML:

<имя тэга_параметр="значение">

Обязательный набор тэгов в документе:

Обязательный набор тегов в документе. Листинг 1.1

```
<html>
<head>
<title>...</title>(по последним стандартам - обязателен)
<head>
<body>
... ... ...
</body>
</html>
```

Помни: Браузер при прочтении кода игнорирует переводы строки (вводы), для него это пробел, причем несколько подряд пробелов воспринимается как один пробел. HTML позволяет разбивать текст на абзацы следующими тэгами:

** ** (по последним стандартам закрывающий тэг обязателен)

**
** (закрывающий тэг не требуется) Этот тэг не создает абзацев, а позволяет перейти на новую строку не доходя до конца строки.

Тэги оформления текста

курсив

курсив (более предпочтителен, хотя и идентичен предыдущему)

жирный шрифт

жирный шрифт (более предпочтителен)

<u>подчеркнутый текст</u>

<hr> разделитель, горизонтальная черта

<h1>**Заголовок первого уровня**</h1>

<h6>**Заголовок последнего уровня**</h6> (причем, тэги заголовков не могут находиться внутри абзацев, и наоборот)

<code> Вставляем фрагмент исходного текста программ. Этот тэг отвечает за отображаемые шрифты. Часто используется вместе с тэгом **<pre>**, т.к. он сохраняет вводы и пробелы.

<blockquote> блочный тэг, увеличивает отступ.

Параметры body

Цвет текста и фона страницы прописывается в параметрах **body**:

<body bgcolor="black" text="white"> (делаем фон черным, а буквы - белыми)

Значение параметров цвета можно также задавать в системе RGB:

<body bgcolor="#000000" text="#ffffff">

Прописываем цвет ссылок:

<body link="blue" vlink="red" alink="yellow">, где

link - цвет непосещенной страницы

vlink - цвет посещенной ссылки

alink - цвет ссылки, на которую наведен курсор

Вставка картинки-фона:

<body background="pic.jpg">

Ссылка

Тэг ссылки**<a>** автоматически подчеркивается и становится синего цвета.

****mikhalkevich.colony.by****

Кроме переходов на другие документы/ресурсы, с помощью этого тэга можно производить переходы внутри странички. Делается это следующим образом:

Сперва ставим метку в том месте, куда будет переходить ссылка:>

****

После чего прописываем саму ссылку перехода:

****Переходим в начало страницы****

Иногда требуется, чтобы ссылка открывалась в новом окне, для этого существует атрибут **target**. Пример:

 Эта ссылка будет открываться в новом окне

Спецсимволы, которые надо знать наизусть

< <

> >

** ** неразрывный пробел, служит для связки слов, которые нельзя разрывать переносами (Пример: 2007 г.)

© ©

« «

» »

" " (обычные двойные ковычки)

— — (длинное тире)

– – (тире покороче)

Вставка изображения

Следует помнить, что тэг **<img>** не может находиться непосредственно в контейнере <body> ... </body>, его нужно поместить внутрь любого блочного элемента, например ... . Для этого тэга имеется два обязательных атрибута: **src** (связывающий документ с изображением) и **alt** (альтернативное текстовое описание рисунка)

<img src="my.jpg" alt="изображение1"> Вставка картинки из того же каталога, где лежит документ

<img src="/my/my.jpg"> Если картинка лежит на уровень ниже, в поддиректории

<img src="../my.jpg"> Если картинка лежит на уровень выше, а документ находится в поддиректории

<img src="http://www.homepage.ru/my/my.jpg"> Если картинка находится на другом сайте, то путь прописывается полностью.

Чтобы заставить браузеры отображать всплывающие подсказки, текст последних следует заключить в атрибут **title**.

Задаем параметры изображению с помощью атрибутов **width** и **height**

<img src="my.jpg" width="200" height="100"> Однако, пользуясь этими тэгами следует помнить, что браузер будет сужать и растягивать изображение, а делать это он не очень-то умеет.

Центрируем изображение:

<**img src="my.jpg" align="right"**> по правому краю

<**img src="my.jpg" align="left"**> по левому краю

Добавляем/убираем рамку вокруг изображения:

<**img src="my.jpg" border="2"**> толщина этой рамки составляет 2 пикселя.

Рассмотрим пример, когда надо избавляться от рамки:

<img src="my.jpg">, мы прописали код ссылки-картинки, вокруг которой автоматически появится рамка, поэтому дописываем атрибут border со значением "0", чтобы избавиться от рамки.

<img src="my.jpg" border="0">

Отступ по горизонтали/вертикали задается атрибутами **hspace** и **vspace** соответственно.

<img src="my.jpg" hspace="2" vspace="4">

Списки

Списки не могут находиться внутри абзацев, но внутри списков можно использовать абзацы.

Два типа списков:

Нумерованные задаются тэгом**
**Пример:

<ol>

**
Первый
**

**
Второй
**

**
**

1. Первый
2. Второй

Параметры <ol>:

<ol type="1"> нумеруем список обычными цифрами

<ol type="a"> нумеруем список латинскими прописными буквами

<ol type="A"> нумеруем список латинскими большими буквами

<ol type="i"> нумеруем список римскими прописными цифрами

<ol type="I"> нумеруем список римскими большими цифрами

<ol start="10"> начинаем считать с 10

Маркированный список задается тэгом **
** Пример:

<ul>

**
Первый
**

**
Второй
**

**
**

- Первый
- Второй

Параметры

- **<ul type="disc">**
- **<ul type="square">**
- **<ul type="circle">**

Немного о таблице

<table> начинаем таблицу

<tr> прописываем строчку в таблице

<td>а эта ячейка таблицы**</td>**

</tr>закрываем строчку в таблице

</table> закрываем саму таблицу.

Создадим таблицу. Листинг 1.2

```
<table width="50%">
<tr><td>1</td><td>2</td><tr>
<tr><td>3</td><td>4</td><tr>
</table>
```

Браузер отобразит этот код так:

1	2
3	4

Параметры таблицы:

<table align="center"> центрируем таблицу относительно документа. Также можно использовать значения "**right**" и "**left**".

<table border="2"> добавляем рамку вокруг таблицы

<table bgcolor="red"> цвет фона

<table width="20%" height="100"> Прописываем высоту в пикселях и ширину в процентах

<table cellpadding="3"> отступ внутри ячейки между содержимым таблицы

<table cellspacing="2"> отступ или расстояние между ячейками таблицы

Содержимое внутри ячеек по умолчанию располагается в середине, чтобы изменить это расположение прописываем внутри тэга<td> атрибут **valign**

<td valign="top"> содержимое ячейки привязывается к верху

<td valign="bottom"> содержимое ячейки привязывается ко дну ячейки

<td valign="middle"> (по умолчанию) содержимое в середине ячейки.

Ячейки в таблице можно объединять:

<td colspan="2"> объединяем 2 ячейки в одну строку

<td rowspan="4"> объединяем 4 ячейки в один ряд

Внутрь тэга <td></td> можно еще вложить таблицу.

Подробнее о табличной верстке читай в следующей лекции.

В этом разделе мы ознакомились лишь с основами html. Но этих знаний должно быть достаточно, чтобы имея под рукой лишь справочник html создавать сайты. Отличный справочник можно найти по адресу: http://html.manual.ru/

Вся правда о табличной верстке

Существует предвзятое мнение: будто табличная верстка устарела. Но это не так, многие крупные сайты не чураются применять табличную верстку (например, яндекс).
Прежде чем приступать к табличной верстке страниц посредством html, необходимо изучить как создавать сами таблицы.

Для описания таблиц в html используется три основных тэга:<table>, <tr>, <td>.

Тэг **<table> </table>** собственно и определяет начало и конец таблицы. Тэг **<tr>** отвечает за формирование строк в таблице. Тэг **<td>** прописывает ячейки таблицы.

Каждая строка таблицы (элемент <tr> </tr>) может содержать в себе одну или несколько ячеек (<td> </td>).

Проще всего реализовать создание правильной таблицы, которая бы содержала одинаковое количество ячеек всех строках и во всех столбцах

Примитивная таблица с заданными параметрами ширины в 50%, границ 2px и выравниванию по центру страницы. Листинг 1.3

```
<table width="50%" border="2" align="center">
<tr>
<td>ячейка1</td><td>ячейка2</td>
</tr>
<tr>
<td>ячейка3</td><td>ячейка4</td>
</tr>
</table>
```

ячейка1	ячейка2
ячейка3	ячейка4

Но такие правильные таблицы на практики встречаются очень редко. Часто возникает необходимость растянуть некоторые ячейки на несколько столбцов или строк. Для этих целей в таблице предусмотрены атрибуты **rowspan** (число строк на которые растягивается данная ячейка вправо, по горизонтали) и **colspan** (число столбцов, на которые растягивается данная ячейка вниз, по вертикали). Причем, одна и та же ячейка может быть растянута как по вертикали, так и по горизонтали.

Использование атрибутов colspan и rowspan. Листинг 1.4

```
<table align="center" border="3" width="80%" bordercolor="#000000">
<tr>
<td colspan="2" width="70%">Технологии</td>
<td bgcolor="#CCCCCC">Год утверждения в качестве рекомендации</td>
</tr>
<tr>
<td rowspan="4">Языки разметки</td>
<td>HTML 4.01</td>
<td bgcolor="#CCCCCC">1999</td>
</tr>
<tr>
<td>XHTML 1.0</td>
<td bgcolor="#CCCCCC">2000</td>
</tr>
<tr>
<td>XHTML 1.1</td>
<td bgcolor="#CCCCCC">2001</td>
</tr>
<tr>
<td>HTML 5</td>
<td bgcolor="#CCCCCC">2012</td>
</tr>
<tr>
<td rowspan="2">Каскадные листы стилей</td>
<td>CSS2</td>
<td bgcolor="#CCCCCC">1998</td>
</tr>
<tr>
<td>CSS3</td>
<td bgcolor="#CCCCCC">Обладает статусом Candidate Recommenda-
tion</td>
</tr>
 </table>
```

Технологии		Год утверждения в качестве рекомендации
Языки разметки	HTML 4.01	1999

	XHTML 1.0	2000
	XHTML 1.1	2001
	HTML 5	2012
Каскадные листы стилей	CSS2	1998
	CSS3	Обладает статусом Candidate Recommendation

Атрибуты тэга <table>

Align..... задает параметры выравнивания самой таблицы в документе;

width..... задает рекомендуемую ширину таблицы;

Height..... атрибут указывающий рекомендуемую высоту таблицы;

Border..... управляет видимостью и толщиной рамки вокруг таблицы и видимостью рамки вокруг каждой ячейки;

Cellspacing ...указывает расстояние в пикселах между смежными ячейками (а точнее между их рамками) как по горизонтали, так и по вертикали;

Cellpadding ..указывает размер отступа (по горизонтали и по вертикали) в пикселах между рамкой и содержимым ячейки;

Bordercolor ..этот атрибут определяет цвет рамок в таблице;

Bgcolor..... задает цвет фона на всем пространстве, занимаемом таблицей — на одержимом ячеек и на свободном пространстве между ними;

id, class..... идентификаторы в пределах документа;

lang, dir..... информация о языке и направленности текста;

title ..... заголовок элемента (выводится браузером в качестве комментария при наведении курсора на содержимое таблицы);

style ..... встроенная информация о стиле;

onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup..... внутренние события.

С некоторыми атрибутами мы уже познакомились (см. код таблицы), а остальные прописываются аналогично.

Атрибуты тэга <tr>

Align, char, charoff, valign..... атрибуты выравнивания;

Bgcolor..... атрибут, задающий цвет фона ячеек, содержащей его строки;

id, class..... идентификаторы в пределах документа;

lang, dir..... информация о языке и направленности текста;

title ..... заголовок элемента (выводится браузером в качестве комментария при наведении курсора на содержимое документа);

style ..... встроенная информация о стиле;

onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup..... внутренние события.

Атрибуты тэга <td> и <th>

Rowspan.... атрибут, отвечающий за объединение соседних ячеек в столбце;

Colspan..... атрибут, отвечающий за объединение соседних ячеек в строке;

Nowrap..... логический атрибут, наличие которого запрещает разбивать содержимое ячейки на несколько строк;

Width..... атрибут, указывающий браузеру рекомендуемую ширину ячейки в пикселах;

Height..... атрибут, указывающий браузеру рекомендуемую высоту ячейки в пикселах;

Align, char, charoff, valign.....атрибуты выравнивания;

Bgcolor..... атрибут, задающий цвет фона внутри ячейки;

id, class..... идентификаторы в пределах документа;

lang, dir..... информация о языке и направленности текста;

title ..... заголовок элемента (выводится браузером в качестве комментария при наведении курсора на содержимое элемента);

style ..... встроенная информация о стиле;

onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup....внутренние события.

Теперь, когда мы знаем всё о таблицах в html, можно приступать и к табличной верстке страниц. Внимательно изучите код представленный ниже. Для удобства восприятия кода, каждая таблица выделена своим цветом.

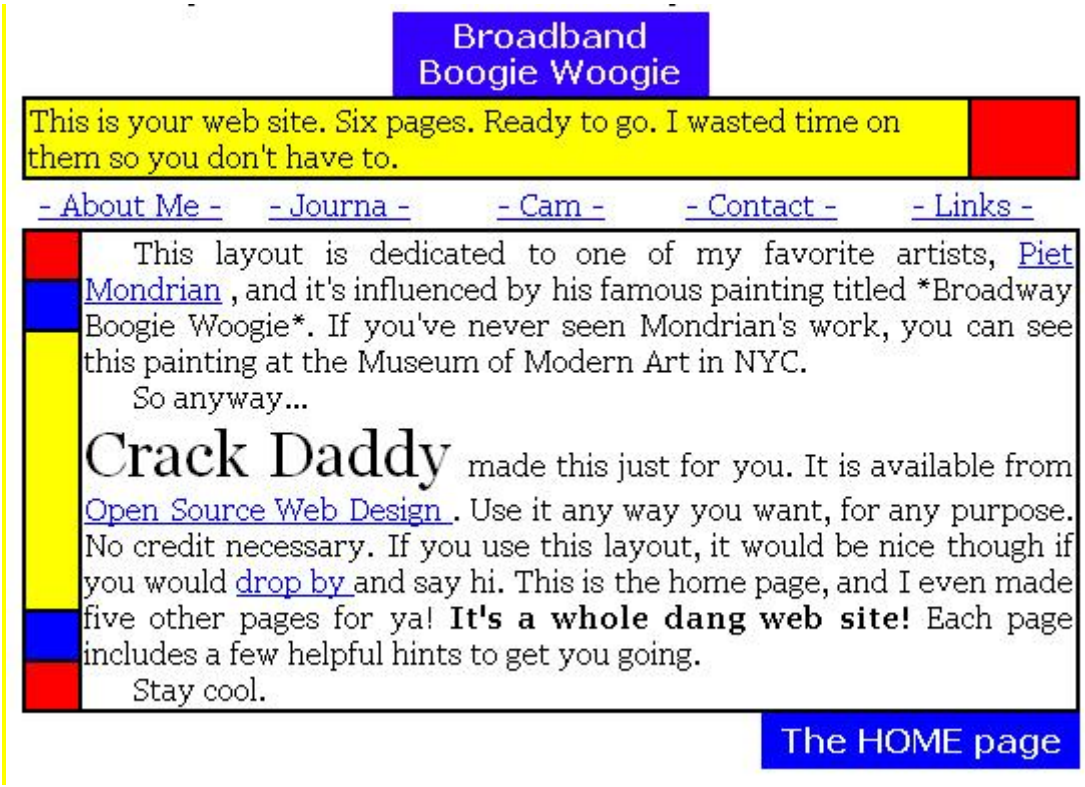
Пример табличной верстки. Листинг 1.5

```
<table width="70%" align="center">
<tr>
<td>
<table align="center" width="30%" bgcolor="#3300FF">
<tr>
<td><center> <font color="#FFFFFF" face="Verdana, Arial, Helvetica, sans-serif"><b>Broadband Boogie Woogie</b>
</font></center> </td>
</tr>
</table>
<table align="center" width="100%" border="0" bgcolor="#000000" cellspacing="2">
<tr>
<td bgcolor="#FFFF00" width="90%">This is your web site. Six pages. Ready to go. I wasted time on them soyou don't have to.</td>
<td bgcolor="#FF0000"></td>
</tr>
</table>
<table border="0" width="100%">
<tr>
<td width="20%" align="center"><a href="#">- About Me -
</a></td>
<td width="20%" align="center"><a href="#"> - Journa -
</a></td>
<td width="20%" align="center"><a href="#"> - Cam -</a></td>
<td width="20%" align="center"><a href="#"> - Contact -
</a></td>
<td align="center"><a href="#">- Links -</a></td>
</tr>
</table>
<table width="100%" border="0" bgcolor="#000000" cellspacing="2">
<tr height="20">
<td bgcolor="#FF0000" width="5%"> </td>
<td rowspan="5" bgcolor="#FFFFFF"><p>This layout is dedicated to one of my favorite artists, <a
href="http://www.artchive.com/artchive/M/mondrian.html">Piet Mondrian</a>
, and it's influenced by his famous painting titled *Broadway Boogie Woogie*. If you've never
```

```
    seen Mondrian's work, you can see this painting at the Museum
    of Modern Art in NYC.</p>
    <p>So anyway...<br />
    <font size="+3">Crack Daddy </font>made this just for you. It
    is available from <a
    href="http://www.oswd.org/index.phtml">Open Source Web Design
    </a>. Use it any way you want, for any purpose. No credit nec-
    essary. If you use this layout, it would be nice though if you
    would <a href="http://tinyplace.net/"> drop by </a> and say
    hi. This is the home page, and I even made five other pages
    for ya! <b>It's a whole dang web site!</b> Each page includes
    a few helpful hints to get you going.</p>
    <p>Stay cool.</p>
    </td>
    </tr>
    <tr height="20"><td bgcolor="blue" width="5%"></td></tr>
    <tr height="120"><td bgcolor="yellow" width="5%"></td></tr>
    <tr height="20"><td bgcolor="blue" width="5%"></td></tr>
    <tr height="20"><td bgcolor="red" width="5%"></td></tr>
    </table>
    <table align="right" width="100%" border="0" cellpadding="5"
    cellspacing="0"> <tr><td width="70%" bgcol-
    or="#FFFFFF"></td><td bgcolor="#0000FF" ><center> <font col-
    or="#FFFFFF" face="Verdana, Arial, Helvetica, sans-serif"><b>
    The HOME page</b> </font></center> </td></tr></table>
</td></tr>
</table>
```

Как видите, ничего нового и непонятного в этом коде мы не увидели, и у нас уже получилась достаточно симпатичная страничка созданная табличной верстkой:

!!! Тэги <font> и мы использовали только в учебных целях. В реальном проекте от них следует отказаться, т.к. они считаются устаревшими. Вместо стилистических тегов необходимо использовать стили css, которые будут рассмотрены позже**.**



Формы HTML

Форма — это инструмент, с помощью которого HTML-документ может послать некоторую информацию в некоторую заранее определенную точку внешнего мира, где информация будет некоторым образом обработана.

Рассказать о формах в книге, посвященной HTML, достаточно трудно. Причина очень простая: создать форму гораздо проще, чем ту "точку внешнего мира", в которую форма будет посылать информацию. В качестве такой "точки" в большинстве случаев выступает программа, написанная на Перл или Си. Программы, обрабатывающие данные, переданные формами, часто называют CGI-скриптами. Сокращение CGI (Common Gateways Interface) означает "общепринятый интерфейс шлюзов". Написание CGI-скриптов в большинстве случаев требует хорошего знания соответствующего языка программирования и возможностей операционной системы Unix.

В последнее время определенное распространение получил язык PHP/FI, инструкции которого можно встраивать прямо в HTML-документы (документы при этом сохраняются в виде файлов с расширением *.pht или *.php).

Формы передают информацию программам-обработчикам в виде пар [имя переменной]=[значение переменной]. Имена переменных следует задавать латинскими буквами. Значения переменных воспринимаются обработчиками как строки, даже если они содержат только цифры.

Как устроена форма

Форма открывается меткой **<FORM>** и заканчивается меткой **</FORM>**. HTML-документ может содержать в себе несколько форм, однако формы не должны находиться одна внутри другой. HTML-текст, включая метки, может размещаться внутри форм без ограничений.

Метка **<FORM>** может содержать три атрибута, один из которых является обязательным. Вот эти атрибуты:

ACTION
: Обязательный атрибут. Определяет, где находится обработчик формы.

METHOD
: Определяет, каким образом (иначе говоря, с помощью какого метода протокола передачи гипертекстов) данные из формы будут переданы обработчику. Допустимые значения: **METHOD=POST** и **METHOD=GET**. Если значение атрибута не установлено, по умолчанию предполагается **METHOD=GET.**

ENCTYPE
: Определяет, каким образом данные из формы будут закодированы для передачи обработчику. Если значение атрибута не установлено, по умолчанию предполагается ENCTYPE=application/x-www-form-urlencoded.

Простейшая форма

Для того, чтобы запустить процесс передачи данных из формы обработчику, нужен какой-то орган управления. Создать такой орган управления очень просто:

<INPUT TYPE=submit>

Встретив такую строчку внутри формы, браузер нарисует на экране кнопку с надписью Submit (читается "сабмит" с ударением на втором слоге, от английского "подавать"), при нажатии на которую все имеющиеся в форме данные будут переданы обработчику, определенному в метке **<FORM>.**

Надпись на кнопке можно задать такую, какая нравится, путем введения атрибута **VALUE="[Надпись]"** (читается "вэлью" с ударением на первом слоге, от английского "значение"), например:

<INPUT TYPE=submit VALUE="Поехали!">

Теперь мы знаем достаточно для того, чтобы написать простейшую форму . Она не будет собирать никаких данных, а просто вернет нас к тексту этой главы.

Основа формы. Листинг 1.6

```
<HTML>
<HEAD>
<TITLE>Пример 11</TITLE>
</HEAD>
<H1>Простейшая форма </H1>
<FORM ACTION="1.html"> <!--Это начало формы-->
<INPUT TYPE=submit VALUE="Назад, к Основам html">
</FORM> <!--Это конец формы-->
</BODY>
</HTML>
```

Надпись, нанесенную на кнопку, можно при необходимости передать обработчику путем введения в определение кнопки атрибута NAME=[имя] (читается "нэйм", от английского "имя"), например:
<INPUT TYPE=submit NAME=button VALUE="Поехали!">

При нажатии на такую кнопку обработчик вместе со всеми остальными данными получит и переменную button со значением Поехали!.

В форме может быть несколько кнопок типа submit с различными именами и/или значениями. Обработчик, таким образом, может действовать поразному в зависимости от того, какую именно кнопку submit нажал пользователь.

Как форма собирает данные

Существуют и другие типы элементов **<INPUT>**. Каждый элемент **<INPUT>** должен включать атрибут **NAME=[имя]**, определяющий имя элемента (и, соответственно, имя переменной, которая будет передана обработчику). Имя должно задаваться **только латинскими буквами**. Большинство элементов **<INPUT>** должны включать атрибут **VALUE="[значение]"**, определяющий значение, которое будет передано обработчику под этим именем. Для элементов **<INPUT TYPE=text>** и **<INPUT TYPE=password>**, однако, этот атрибут не обязателен, поскольку значение соответствующей переменной может вводиться пользователем с клавиатуры.

Основные типы элементов **<INPUT>**:
TYPE=submit

Кнопка отправляющая данные. Пример использования смотри выше.

TYPE=buttom

Кнопка без определенных действий. Пример:
<INPUT TYPE=submit VALUE="Кнопка без определенных действий произвольного размера"> Пример смотри ниже.

Это не значит, что кнопка совершенно бесполезна. Ее действия мы научимся задавать позже, в лекциях по PHP.

TYPE=file

Специальные поля с кнопкой для отправки фалов. Пример:
<INPUT TYPE=file NAME=button VALUE="Поехали!"> С тем, как выглядит кнопка в действии, можно познакомиться в примере в конце урока. Там же - все остальные типы элементов форм.

TYPE=text

Определяет окно для ввода строки текста. Может содержать дополнительные атрибуты SIZE=[число] (ширина окна ввода в символах) и MAXLENGTH=[число] (максимально допустимая длина вводимой строки в символах). Пример:

<INPUT TYPE=text SIZE=20 NAME=user VALUE="Иван">

Определяет окно шириной 20 символов для ввода текста. По умолчанию в окне находится текст Иван, который пользователь может редактировать. Отредактированный (или неотредактированный) текст передается обработчику в переменной user.

TYPE=password

Определяет окно для ввода пароля. Абсолютно аналогичен типу text, только вместо символов вводимого текста показывает на экране звездочки (*). Пример:

<INPUT TYPE=password NAME=pw SIZE=20 MAXLENGTH=10>

Определяет окно шириной 20 символов для ввода пароля. Максимально допустимая длина пароля — 10 символов. Введенный пароль передается обработчику в переменной pw.

TYPE=radio

Определяет радиокнопку. Может содержать дополнительный атрибут checked (показывает, что кнопка помечена). В группе радиокнопок с одинаковыми именами может быть только одна помеченная радиокнопка. Пример:

<INPUT TYPE=radio NAME=modem VALUE="9600" checked> 9600 бит/с
<INPUT TYPE=radio NAME=modem VALUE="14400"> 14400 бит/с
<INPUT TYPE=radio NAME=modem VALUE="28800"> 28800 бит/с

Определяет группу из трех радиокнопок, подписанных 9600 бит/с, 14400 бит/с и 28800 бит/с. Первоначально помечена первая из кнопок. Если пользователь не отметит другую кнопку, обработчику будет передана переменная modem со значением 9600. Если пользователь отметит другую кнопку, обработчику будет передана переменная modem со значением 14400 или 28800.

TYPE=checkbox

Определяет квадрат, в котором можно сделать пометку. Может содержать дополнительный атрибут checked (показывает, что квадрат помечен). В отличие от радиокнопок, в группе квадратов с одинаковыми именами может быть несколько помеченных квадратов. Пример:

<INPUT TYPE=checkbox NAME=comp VALUE="PC"> Персональные компьютеры
<INPUT TYPE=checkbox NAME=comp VALUE="WS" checked> Рабочие

станции

<INPUT TYPE=checkbox NAME=comp VALUE="LAN"> Серверы локальных сетей

<INPUT TYPE=checkbox NAME=comp VALUE="IS" checked> Серверы Интернет

Определяет группу из четырех квадратов. Первоначально помечены второй и четвертый квадраты. Если пользователь не произведет изменений, обработчику будут переданы две переменные: comp=WS и comp=IS.

TYPE=hidden

Определяет скрытый элемент данных, который не виден пользователю при заполнении формы и передается обработчику без изменений. Такой элемент иногда полезно иметь в форме, которая время от времени подвергается переработке, чтобы обработчик мог знать, с какой версией формы он имеет дело. Другие возможные варианты использования Вы вполне можете придумать сами. Пример:

<INPUT TYPE=hidden NAME=version VALUE="1.1">

Определяет скрытую переменную version, которая передается обработчику со значением 1.1.

TYPE=reset

Определяет кнопку, при нажатии на которую форма возвращается в исходное состояние. Поскольку при использовании этой кнопки данные обработчику не передаются, кнопка типа reset может и не иметь атрибута name. Пример:

<INPUT TYPE=reset VALUE="Очистить поля формы">

Определяет кнопку Очистить поля формы, при нажатии на которую форма возвращается в исходное состояние.

Помимо элементов **<INPUT>**, формы могут содержать меню **<SELECT>** и поля для ввода текста **<TEXTAREA>.**

Меню **<SELECT>** из n элементов выглядит примерно так:

<SELECT NAME="[имя]">
<OPTION VALUE="[значение 1]">[текст 1]
<OPTION VALUE="[значение 2]">[текст 2]
...
<OPTION VALUE="[значение n]">[текст n]
</SELECT>

Как Вы видите, меню начинается с метки **<SELECT>** и заканчивается меткой **</SELECT>**. Метка **<SELECT>** содержит обязательный атрибут **NAME**, определяющий имя переменной, которую генерирует меню.

Метка **<SELECT>** может также содержать атрибут **MULTIPLE**, присутствие которого показывает, что из меню можно выбрать несколько элементов. Большинство браузеров показывают меню **<SELECT MULTIPLE>** в виде окна, в котором находятся элементы меню (высоту окна в строках можно задать атрибутом SIZE=[число]). Меню **<SELECT>** в большинстве случаев показывается в виде выпадающего меню.

Метка **<OPTION>** определяет элемент меню. Обязательный атрибут **VALUE** устанавливает значение, которое будет передано обработчику, если выбран этот элемент меню. Метка **<OPTION>** может включать атрибут **checked**, показывающий, что данный элемент отмечен по умолчанию.

Разберем небольшой пример.

```
<SELECT NAME="selection">
<OPTION VALUE="option1" checked>Вариант 1
<OPTION VALUE="option2">Вариант 2
<OPTION VALUE="option3">Вариант 3
</SELECT>
```

Такой фрагмент определяет меню из трех элементов: Вариант 1, Вариант 2 и Вариант 3. По умолчанию выбран элемент Вариант 1. Обработчику будет передана переменная **selection** значение которой может быть **option1** (по умолчанию), **option2** или **option3**.

После всего, что мы уже узнали, элемент **<TEXTAREA>** может показаться совсем простым. Например:

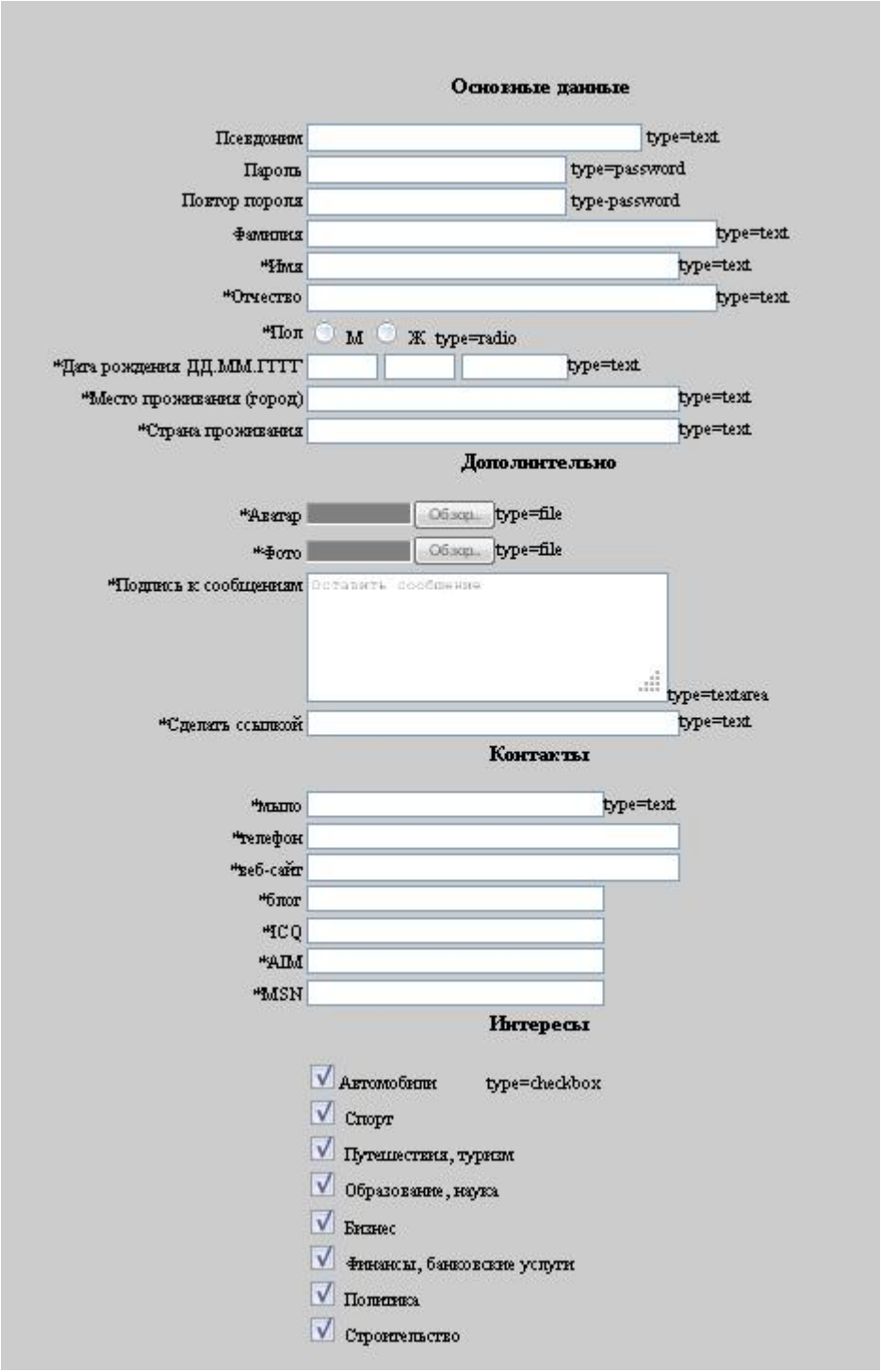
```
<TEXTAREA NAME=address ROWS=5 COLS=50>
А здесь - Ваш адрес...
</TEXTAREA>
```

Все атрибуты обязательны. Атрибут **NAME** определяет имя, под которым содержимое окна будет передано обработчику (в примере — address). Атрибут **ROWS** устанавливает высоту окна в строках (в примере — 5). Атрибут **COLS** устанавливает ширину окна в символах (в примере — 50).

Текст, размещенный между метками **<TEXTAREA>** и **</TEXTAREA>**, представляет собой содержимое окна по умолчанию. Пользователь может его отредактировать или просто стереть.

Важно знать, что русские буквы в окне **<TEXTAREA>** при передаче обработчику могут быть конвертированы в соответствующие им символьные объекты.

Пример, в котором можно ознакомиться со всеми типами элементов формы



В HTML 5 появились новые типы элементов форм (для ввода даты, цвета, ползунок прокрутки и т.д). Далее мы их рассмотрим. Но т.к. полноценную поддержку HTML5 всеми мултемедийными устройствами и браузерами планируется осуществить только к 2022 году, активно использовать новые элементы форм пока не рекомендуется.

Новые типы input в HTML5 формах

Обратите внимание: на данный момент новые типы input поддерживаются полностью только не во всех брузерах.

- **input type=email** определяет поле, которые должно содержать email адрес. Значение введенное в поле автоматически проверяется перед отправкой на сервер.
- **input type=url** определяет поле, которые должно содержать url адрес. Значение введенное в поле автоматически проверяется перед отправкой на сервер.
- **input type=tel** определяет поле для ввода телефонного номера. С помощью атрибута pattern Вы может задать формат принимаемого телефонного номера. Формат задается с помощью регулярных выражений.
- **input type=number** определяет поле, которое должно содержать числа. Вы можете ограничивать диапазон принимаемых чисел с помощью атрибутов min (*минимальное допустимое число*) и max (*максимальное допустимое число*). С помощью атрибута step Вы можете задать шаг допустимых чисел (*к примеру если шаг равен 2, то в поле могут вводиться числа 0,2,4,6 и т.д.*)
- **input type=range** определяет поле, которые может содержать значения в определенном интервале. Отображается как ползунок, который можно перетаскивать мышкой. Вы можете ограничивать диапазон принимаемых чисел с помощью атрибутов min (*минимальное допустимое число*) и max (*максимальное допустимое число*). С помощью атрибута step Вы можете задать шаг допустимых чисел (*к примеру если шаг равен 2, то в поле могут вводиться числа 0,2,4,6 и т.д.*)
- **input type=search** определяет поле поиска (*может использоваться например для создания поиска по сайту*).

Новые типы форм в html5. Листинг 1.7

```
<input name='email' type='email' value='He email' />
<input name='url' type='url' value='He url' />
<input name='tel1' type='tel' pattern='8[0-9]{10}' />
<input name='tel2' type='tel' pattern='[0-9]{2,3}-[0-9]{2}-[0-
9]{2}' />
<input name='number' type='number' value='10' />
<input name='range' type='range' min='1' max='5' />
<input name='search' type='search' value='Поиск по сайту' />
<input name='color' type='color' />
```

Новые элементы в HTML5 формах

- **datalist** позволяет привязать список к полям формы. Значения списка будут выводится как поисковые подсказки во время ввода информации в поле связанное с ним.
- **keygen** позволяет генерировать открытые и закрытые ключи, которые используются для безопасной связи с сервером.
- **output** может использоваться для вывода различной информации. С помощью атрибута for можно указать связанные поля.

Новые элементы форм в html5. Листинг 1.8

```
<input name='city' type="url" list="city" />
<datalist id="city">
<option label="Москва" value="Москва, Россия" />
<option label="Санкт-Петербург" value="Санкт-Петербург, Россия" />
<option label="Новосибирск" value="Новосибирск, Новосибирская об-
ласть, Россия" />
</datalist>
<keygen name='keygen' />
<output name='result' for='first second'>100</output>
```

Новые атрибуты в HTML5 формах

- **autofocus** делает поле активным после загрузки страницы (*может использоваться со всеми типами input*).
- **form** указывает форму, которой принадлежит данное поле (*может использоваться со всеми типами input*).
- **multiple** указывает, что данное поле может принимать несколько значений одновременно (*может использоваться с input типов email и file*).
- **novalidate** указывает, что данное поле не должно проверяться перед отправкой (*может использоваться с form и input*).
- **placeholder** отображает текст-подсказку в поле (*может использоваться с input следующих типов: text, search, url, tel, email и password.*).
- **required** указывает, что данное поле должно быть обязательно заполнено перед отправкой.

Новые атрибуты в формах в html5. Листинг 1.9

```
<input type='text' autofocus="autofocus" /><br /><br />
<input type='file' multiple='multiple' /><br /><br />
<input type='text' form='form1' /><br /><br />
<form action='html5.php' novalidate='novalidate'>
<input type='email' placeholder='Введите Ваш email' /><br /><br />
<input type='text' required="required" />
```

Выбор даты

В HTML5 были добавлены новые элементы ввода позволяющие удобно выбирать дату и время.

Обратите внимание:

- **date** позволяет выбрать дату в формате год-месяц-день_месяца.
- **time** позволяет выбрать время.
- **datetime** позволяет выбрать дату в формате год-месяц-день_месяцаТвремяZ (*отчет ведется по глобальному времени*).
- **datetime-local** позволяет выбрать дату в формате год-месяц-день_месяцаТвремя (*отчет ведется по местному времени*).
- **month** позволяет выбрать дату в формате год-месяц.
- **week** позволяет выбрать дату в формате год-Wнеделя.

Новые атрибуты в формах в html5. Листинг 1.10

```
<input type='date' /><br />
<input type='time' /><br />
<input type='datetime' /><br />
<input type='datetime-local' /><br />
<input type='month' /><br />
<input type='week' /><br />
```

HTML5 видео

HTML5 вводит специальный тэг **<video>** позволяющий встраивать в веб-страницы видео файлы.

Обратите внимание: поддержку видео имеют браузеры: Internet Explorer 9+, Opera 10.50+, Firefox 3.5+, Chrome 3.0+, Safari 3.1+.

Атрибут src тэга <video> позволяет указать путь к видео файлу, который будет воспроизведен.

Атрибут controls отображает в плеере кнопки управления видео, а атрибуты height (*высота*) и width (*ширина*) задают размеры плеера.

Добавление видео. Листинг 1.11

```
<video src="mountvideo.ogv" width='300' height='200' con-
trols='controls'></video>
```

Видео из предыдущего примера будет нормально проигрываться только в браузерах Opera, Firefox и Chrome. Это происходит из-за того, что в данный момент разные браузеры поддерживают разные форматы.

Opera и Firefox поддерживают формат Ogg (*с видео кодеком Theora и аудио кодеком Vorbis*) и WebM (*с видео кодеком VP8 и аудио кодеком Vorbis*), а Internet Explorer и Safari поддерживают MPEG4 (*с видео кодеком H.264 и аудио кодеком AAC*). Браузер Chrome имеет поддержку всех перечисленных форматов.

Таким образом для того, чтобы видео нормально воспроизводилось во всех браузерах необходимо добавить источники в формате Ogg и MPEG4 (*или WebM и MPEG4*).

С помощью тэга **<source>** Вы можете предоставить несколько источников видео в разных форматах для воспроизведения. Браузер будет использовать первый поддерживаемый им формат.

Использование тега <source> для видео. Листинг 1.12

```
<video width="300" height="200" controls="controls">
<source src="mountvideo.ogv" type="video/ogg" />
<source src="mountvideo.mp4" type="video/mp4" />
<source src="mountvideo.webm" type="video/webm" />
</video>
```

Текст помещенный в содержимое элемента video будет отображен в случае если браузер пользователя не поддерживает тэг видео.

Поддержка старых браузеров. Листинг 1.13

```
<video src='mountvideo.ogv' width="300" height="200" con-
trols="controls">Ваш браузер
не поддерживает элемент video.</video>
```

HTML5 аудио

С помощью нового HTML5 элемента **<audio>** Вы можете добавить на Ваш веб-сайт музыкальный трек или подкаст с помощью обычного разметочного тэга.

Обратите внимание: поддержку аудио имеют следующие браузеры: Internet Explorer 9+, Opera 10.50+, Firefox 3.5+, Chrome 3.0+, Safari 3.0+.

Атрибут src тэга <audio> позволяет указать путь к аудио файлу, а атрибут control добавляет кнопки управления.

Добавление аудио. Листинг 1.14

```
<audio src="ghost_k-stop.ogv" controls='controls'></audio>
```

В предыдущем примере используется только формат Ogg, поэтому пример будет работать только в Firefox, Opera и Chrome. Для того, чтобы файл нормально проигрывался в Safari и Internet Explorer необходимо добавить также файл в формате MP3.

С помощью элемента source Вы можете предоставить несколько источников в разных форматах для воспроизведения. Браузер будет использовать первый поддерживаемый им формат.

Текст находящийся в содержимом элемента будет отображен в случае если браузер пользователя не поддерживает элемент audio.

Использование тега <source> для аудио. Листинг 1.15

```
<audio controls="controls">
<source src="ghost_k-stop.ogv" type="audio/ogg" />
<source src="ghost_k-stop.mp3" type="audio/mpeg" />
Данный текст будет выведен если браузер пользователя не поддержива-
ет элемент audio.
</audio>
```

С помощью атрибута autoplay Вы можете заставить трек автоматически проигрываться после загрузки страницы.

Автоматическое проигрывание после загрузки страницы. Листинг 1.16

```
<audio autoplay='autoplay'>
<source src="ghost_k-stop.ogv" type="audio/ogg" />
<source src="ghost_k-stop.mp3" type="audio/mpeg" />
Данный текст будет выведен если браузер пользователя не поддержива-
ет элемент audio.
</audio>
```

Конвертация аудио в форматы поддерживаемые HTML5

С помощью программы Miro Video Converter Вы можете также конвертировать в необходимые форматы (*MP3 и Ogg*) аудио файлы.

Семантические теги

В HTML4 благодаря использованию CSS Вы могли создавать страницы с очень понятной для пользователей структурой, но были ли эти страницы также понятны для поисковых систем или браузеров?

Например, как поисковый робот мог отличить содержимое документа от футера или навигационного меню если все они размечены с помощью элементов div?

Семантические тэги добавленные в HTML5 позволяют сделать страницы такими же понятными для поисковых систем и браузеров как и для пользователей.

Структура документа

Тэг	Описание
footer	Определяет футер.
header	Определяет заголовочный блок сайта.
nav	Определяет навигационное меню.

Так выглядела общая структура документа в HTML4.

Так выглядит общая структура документа в HTML5 с использованием новых тэгов разметки.

Группировка содержимого

С помощью тэга **<section>** Вы можете сгруппировать логически связанное содержимого в документе.

Если логически связанное содержимое является автономным (*может использоваться в других документах независимо от остального содержимого на странице*) необходимо использовать вместо <section> тэг **<article>**.

Группировка содержимого с использованием тегов <section> и <article>. Листинг 1.17

```
<article>
<h1>Титаник</h1>
<p>«Титаник» — британский пароход компании «Уайт Стар Лайн». Во
время первого рейса 14
апреля 1912 года столкнулся с айсбергом и через 2 часа 40 минут за-
тонул. На борту
находилось 1316 пассажиров и 908 членов экипажа, всего 2224 челове-
ка. <p>
<section>
<h2>Постройка</h2>
<p>Заложен 31 марта 1909 года на верфях судостроительной компании
«Харланд энд Вольф»
в Куинс-Айленд (Белфаст, Северная Ирландия), спущен на воду 31 мая
1911 года, прошёл
ходовые испытания 2 апреля 1912 года.>
</section>
<section>
<h2>Местонахождение</h2>
<p>1 сентября 1985 года экспедиция под руководством директора Ин-
ститута океанологии
города Вудс-Холл, штат Массачусетс, доктора Роберта Балларда (Rob-
ert D. Ballard)
обнаружила место залегания «Титаника» на дне Атлантического океана
на глубине
3750 метров.</p>
</section>
</article>
```

Обратите внимание: тэгом article выделяется статья о Титанике целиком так как данная часть может использоваться в других документах без остального содержимого, которое может находится на данной странице. Тэгом section выделяется логически связанные разделы о постройке и текущем местонахождении титаника т.к. они не являются автономными и не смогут использоваться в других документах без остальной информации данной статьи.

Тэг aside

Тэг aside используется для выделения элементов, которые не являются частью содержимого, но косвенно с ним связаны.

Данным тэгом могут выделяться: цитаты, дополнительная информация к статье, словарь с терминами, список ссылок и т.д.

Выделение элементов, которые не являются частью содержимого, но косвенно связаны с ним 1.18

```
<article>
<h1>Титаник</h1>
<p>«Титаник» — британский пароход компании «Уайт Стар Лайн». Во
время первого рейса 14
апреля 1912 года столкнулся с айсбергом и через 2 часа 40 минут за-
тонул. На борту
находилось 1316 пассажиров и 908 членов экипажа, всего 2224 челове-
ка. <p>
<aside>Гибель Титаника - это одно из самых крупных кораблекрушений
```

```
в истории человечества.</aside>
</article>
```

Подсветка текста

С помощью тэга **<mark>** Вы можете выделить "важную" часть в тексте.

Подсветка текста 1.19

```
 <p>Я считаю, что <mark>HTML5</mark> облегчает жизнь веб-
разработчиков.</p>
```

Подписи для иллюстраций

В журналах и газетах иллюстрации часто сопровождаются подписями. В HTML4 невозможно было создавать подписи не прибегая к использованию CSS.

В HTML5 это проблема решена добавлением новых тэгов: **<figure>** и **<figcaption>**.

Подписи для иллюстраций 1.20

```
 <figure>
<img src='mountimg3.jpg' width='300' height='230' />
<figcaption>Скала "Братья", Западный Саян </figcaption>
</figure>
```

Web-хранилища

Веб хранилища представляют собой более функциональную альтернативу cookie.

Преимущества веб хранилищ:

- Можно хранить неограниченные объемы информации;
- Информация сохраненная в хранилищах доступна даже без подключения к интернету;
- Данные находящиеся в хранилище не отсылаются при каждом запросе страниц;
- Информацию более удобно сохранять и извлекать;
- Хранилища более безопасны чем cookie.

Веб хранилища поддерживаются в следующих браузерах: IE8+, Safari 4+, Chrome 4+, Firefox 3.5+, Opera 10.50+.

Сохранение данных

Вы можете сохранять данные используя два специальных объекта: **sessionStorage** и **localStorage**.

Обращаться к данным объектом можно с помощью JavaScript и других клиентских скриптов.

Обратите внимание: каждый веб-сайт имеет доступ только к своим данным в хранилище и поэтому не может обращаться к данным, которые были сохранены другими сайтами.

Использование sessionStorage

Объект sessionStorage сохраняет данные в течении пользовательской сессии. Когда браузер пользователя будет закрыт данные сохраненные в объекте будут удалены.

Web-хранилища 1.21

```
 //Сохраняем данные
sessionStorage.html='HTML5 предоставляет множество интересных воз-
можностей.';
sessionStorage.wisdomweb='Вы изучаете HTML5';
/* В течении пользовательской сессии мы можем извлечь сохраненные
ранее данные
следующим образом: */
document.write(sessionStorage.html+'<br />');
document.write(sessionStorage.wisdomweb);
```

Данные в хранилище доступны со всех страниц сайта, а не только с той с которой они были сохранены.

Использование localStorage

Объект localStorage сохраняет данные на неограниченный период времени.

Данные сохраненные в данном объекте будут доступны даже без подключения к интернету.

Доступ к данным без подключения к интернету на неограниченный период времени 1.22

```
<script>
function save(){
localStorage.data='Веб хранилища - это более функциональная альтер-
натива cookie.';
}
function load(){
alert(localStorage.data);
}
</script>
<input type='button' value='Сохранить данные' onclick='save()' />
<input type='button' value='Извлечь данные' onclick='load()' />
<p><b>Обратите внимание:</b> Вы сможете извлечь сохраненные данные
даже после перезагрузки браузера.</p>
```

HTML5 Canvas

Элемент **<canvas>** позволяет рисовать на веб-страницах произвольные фигуры с помощью JavaScript (*или других клиентских скриптов*).

При создании элемента canvas необходимо задать атрибут id (*определяет имя элемента для доступа к нему из скриптов*) и его размеры с помощью атрибутов width (*ширина*) и height (*высота*).

Содержимое помещенное между тэгами будет отображено если браузер пользователя не поддерживает элемент canvas.

Canvas 1.23

```
<canvas id='draw' width='300' height='200'>Вы видите это сообщение,
потому что Ваш браузер не поддерживает canvas.</canvas>
```

Обратите внимание: сам по себе canvas ничего не рисует. Он представляет собой холст, который предоставляет возможности для рисования клиентским скриптам.

Рисование прямоугольников

Метод	Описание
fillRect(x,y,ширина,высота)	Рисует закрашенный прямоугольник.
strokeRect(x,y,ширина,высота)	Рисует не закрашенный прямоугольник.
clearRect(x,y,ширина,высота)	Очищает указанную зону делая ее полностью прозрачной.

Обратите внимание: параметры x и y задают величину смещения прямоугольника по горизонтали (x) и вертикали (y) от верхнего левого угла холста в пикселях.

Теперь попробуем нарисовать, что-нибудь холсте.

Рисование прямоугольников 1.24

```
var canvas=document.getElementById("draw");
var x=canvas.getContext("2d");
x.fillRect(50,40,55,55);
x.strokeRect(150,70,55,55);
x.clearRect(68,57,20,20);
```

Объяснение примера:

1. var canvas=document.getElementById("draw") - находим нужный холст;
2. var x=canvas.getContext("2d") - обращаемся ко встроенному объекту, который содержит различные методы для рисования (*первые два шага являются стандартными для рисования любого объекта в canvas*);
3. x.fillRect(50,40,55,55) - рисуем закрашенный прямоугольник;

4. x.strokeRect(150,70,55,55) - рисуем не закрашенный прямоугольник;
5. x.clearRect(68,57,20,20) - очищаем зону в закрашенном прямоугольнике.

Составные фигуры состоят из нескольких соединенных простых объектов (таких как линии, круги и т.д.).

Объекты для рисования составных фигур 1.25

```
beginPath();
/* Простые объекты помещаются здесь */
closePath(); // Автоматически завершает фигуру (соединяет конечную точку с начальной)
//Теперь необходимо вызвать один из методов для рисования фигуры определенной выше
stroke(); //нарисует фигуру не закрашенной
fill(); //нарисует фигуру закрашенной
```

Обратите внимание: метод closePath() может отсутствовать.

Рисование составных фигур 1.26

```
var canvas=document.getElementById("draw");
var x=canvas.getContext("2d");
x.beginPath();
x.moveTo(20,20);
x.lineTo(70,70);
x.lineTo(20,70);
x.closePath();
x.fill();
```

Таблица простых объектов.

Объекты	Описание
moveTo(x,y)	Устанавливает координаты точки, из которой начнется рисование следующего объекта.
lineTo(x,y)	Рисует прямую линию.
arc(x,y,радиус,нач_угол,конеч_угол)	Рисует круг. Угол необходимо задавать в радианах, а не в градусах (*радианы=(Math.PI/180)*градусы*).
rect(x, y, ширина, высота)	Рисует прямоугольник.

Пример использования простых объектов 1.27

```
var x1=(Math.PI/180)*0;
var x2=(Math.PI/180)*360;
var canvas=document.getElementById("draw")
var x=canvas.getContext("2d");
x.beginPath();
x.arc(30,30,20,x1,x2);
x.moveTo(100,100);
x.rect(70,50,70,70);
```

```
x.fill();
```

Цвета

Для раскрашивания нарисованных в canvas фигур предусмотрены свойства: **fillStyle** и **strokeStyle**.

Свойство **fillStyle** используется для применения цвета к закрашенным, а strokeStyle для применения цвета к не закрашенным фигурам.

Раскрашивание нарисованных фигур 1.28

```
var canvas=document.getElementById("draw")
var x=canvas.getContext("2d");
x.fillStyle="green";
x.fillRect(10,40,65,65);
x.strokeStyle="#FF45FF"
x.strokeRect(100,40,65,65);
x.fillStyle="rgb(255,73,73)"
x.fillRect(190,40,65,65);
```

Помимо этих способов в canvas цвет может задавать с помощью rgba (*задается цвет и прозрачность элемента*).

Цвет и прозрачность элемента 1.29

```
var canvas=document.getElementById("draw")
var x=canvas.getContext("2d");
x.fillStyle="rgba(0,0,0,0.5)";
x.fillRect(10,40,65,65);
x.fillStyle="rgba(0,0,0,0.1)";
x.fillRect(100,40,65,65);
x.fillStyle="rgba(0,0,0,1)";
x.fillRect(190,40,65,65);
```

Оформление линий

В HTML5 имеются несколько свойств для оформления линий нарисованных с помощью метода lineTo(): **lineWidth**, **lineCap**, **lineJoin**.

С помощью свойства **lineWidth** Вы можете установить толщину линии (*по умолчанию линии имеют толщину 1 пиксель*).

Толщина линии 1.30

```
var canvas=document.getElementById("draw")
var x=canvas.getContext("2d");
//Рисуем линию толщиной 3 пикселя
x.beginPath()
x.moveTo(10,10);
x.lineWidth=3;
x.lineTo(200,10);
x.stroke();
```

С помощью свойства **lineCap** Вы можете оформлять кончики линии.

Оформление кончиков линий 1.31

```
//Нарисуем линию с круглым кончиком
x.beginPath()
x.moveTo(10,10);
x.lineWidth=10;
x.lineCap='round';
x.lineTo(200,10);
x.stroke();
```

С помощью свойства **lineJoin** Вы можете сглаживать стыки двух линий.

Сглаживание стыков двух линий 1.32

```
//Сгладим стыки линий
x.beginPath()
x.moveTo(10,10);
x.lineWidth=15;
x.lineJoin='round';
x.lineTo(50,50);
x.lineTo(100,10);
x.lineTo(170,80);
x.lineTo(210,40);
x.stroke();
```

Градиент

С помощью метода **createLinearGradient(x1,y1,x2,y2)** Вы можете создать линейный градиент. Параметры x1 и y1 устанавливают координаты начальной, а x2 и y2 конечной точки градиента.

После того, как градиент создан необходимо указать цвета перехода. Это можно сделать с помощью метода **addColorStop(точка,цвет)**.

Например если Вы хотите создать градиент, который плавно изменяет цвет с черного на белый необходимо установить черный цвет в точке 0 (начальная точка градиента) и белый цвет в точке 1 (конечная точка градиента).

Градиент 1.33

```
<canvas id='draw' width='240' height='200' style='border:1px sol-
id;'></canvas>
<script>
var canvas=document.getElementById("draw")
var x=canvas.getContext("2d");
/* Создаем градиент */
var grad = x.createLinearGradient(0,0,0,150);
/* Добавляем точки цветового перехода */
grad.addColorStop(0.0,'black');
grad.addColorStop(1.0,'white');
/* Устанавливаем получившийся градиент как свойство заливки */
x.fillStyle=grad;
/* Рисуем фигиру, к которой будет применен созданный градиент */
x.fillRect(20,20,200,200);
</script>
```

Вы можете создавать более сложные градиенты, которые изменяют цвет в нескольких точках.

Сложный градиент 1.34

```
<canvas id='draw' width='240' height='200' style='border:1px sol-
id;'></canvas>
<script>
var canvas=document.getElementById("draw")
var x=canvas.getContext("2d");
/* Создаем градиент */
var grad = x.createLinearGradient(50,50,150,150);
/* Добавляем точки цветового перехода */
grad.addColorStop(0.0,'#d2006b');//Розовый цвет в начале
grad.addColorStop(0.5,'#00a779');//Бирюзовый цвет в середине
grad.addColorStop(1.0,'#ffe800');//Желтый цвет в конце
/* Устанавливаем получившийся градиент как свойство заливки */
x.fillStyle=grad;
/* Рисуем фигиру, к которой будет применен созданный градиент */
x.fillRect(20,20,190,150);
</script>
```

Создание теней

Свойства	Описание
shadowOffsetX	Смещение тени от объекта по горизонтали (*может принимать отрицательные значения*).
shadowOffsetY	Смещение тени от объекта по вертикали (*может принимать отрицательные значения*).
shadowBlur	Величина размытия тени.
shadowColor	Цвет тени (*по умолчанию черный*).

Создание теней 1.35

```
var canvas=document.getElementById("draw")
var x=canvas.getContext("2d");
x.shadowOffsetX=3;
x.shadowOffsetY=3;
x.shadowBlur=5;
x.shadowColor='black';
x.fillStyle='#ffaa00';
x.fillRect(50,40,55,55);
```

Вот еще некоторые возможности HTML5:

- WebGL – интерфейс для работы с трехмерной графикой
- Включение новых обработчиков событий: мультисенсорного ввода и высокоуровневое событие жестами.

- Использование геолокационной информации: GoogleMap, GPS, IP, MAC-адреса WiFi, номера телефонов GSM. В случае с мобильным телефоном, мы можем определить местоположение с точностью до 10 м. Причем, отслеживая изменения расстояний во времени, мы можем определить скорость движения мобильного телефона, т.е. HTML5 можно использовать в качестве шагомера. Также мы можем найти себя на карте Google

- Кроссдоменный обмен сообщений.

- Поддержка технологии Ajax, в том числе и в кроссдоменных запросах.

- Web Storage на замену cookie. Решена проблема ограничения размеров хранимой информации на стороне клиента (в cookie хранилось не более 4 кб), повышена безопасность хранения данных (файлы web storage не только не видимы в сети, но и зашифрованы).

- Включены функции кэширования целых приложений с возможностью последующих их обновлений.

- Возможность обеспечения двухсторонней связи между сервером и браузером в режиме реального времени!!! Технология WebSocket. При изучении HTML5 уделите данной технологии особое внимание, т.к. со временем она вытеснит Ajax

- Поддержка всех международных языков

- Поддержка всеми доступными браузарми, платформами и мультимедийными устройствами (планшеты, смартфоны и т.д), чем не может похвастаться flash.

Я думаю, это серьезное подспорье, для того, чтобы начать углубленно изучать технологию HTML5.

Все о теге META

Тег **<META>**. Закрывающий тег **</META> запрещен** !

Специальная группа инструкций **<META>**, предназначена в основном для описания и индексирования документа поисковыми машинами. Все тэги META не видны при просмотре документа. Команды **<META>** вносятся в "шапку" гипертекстового документа - внутри блока <HEAD>...</HEAD>.

<META HTTP-EQUIV="Expires" CONTENT="Mon, 25 Sep 2001 00:02:01 GMT">

Используется для того, чтобы в нужное время браузер при просмотре документа брал не версию, хранящуюся в кэше, а свежую версию прямо с Вашего сайта.

<META HTTP-EQUIV="Content-Type" CONTENT="text/html; CHARSET=Windows-1251">

Используется для того, чтобы браузер мог правильно отобразить содержимое страницы и для определения поисковой машиной языка, на котором написана страница

<META HTTP-EQUIV="Refresh" CONTENT="x; URL=http://mikhalkevich.colony.by/">

Используется для того, чтобы браузер автоматически переключался на новый адрес.

<META name="author" content="Alex Mikhalkevich">

Используется для указания имени автора. Поисковые системы могут найти нужный сайт по имени автора (или найти самого автора).

<META name="copyright" content="© 2001 SOFT BANANAS">

Полезно также указать и авторские права название фирмы почти наверняка будет проиндексировано поисковой машиной.

<META http-equiv="PICS-Label" content=' (PICS-1.1 "http://www.gcf.org/v2.5" labels on "1994.11.05T08:15-0500" until "1995.12.31T23:59-0000" for "http://w3.org/PICS/Overview.html" ratings (suds 0.5 density 0 color/hue 1)) '>

Еще одна интересная штучка отсечение нежелательных пользователей от указанной страницы (например, детей от секс-серверов), при помощи введения рейтинга - т.н. "красной лампочки". Некоторые браузеры позволяют "по-

весить замок" на содержимое определенных сайтов, запрещая их просмотр. Имеется несколько признанный рейтинговых систем, распознаваемых браузерами. Сам браузер, естественно, можно подстроить под использование рейтинга, а профиль пользователя браузера защитить паролем. Как правило, текст в этот тэг вставляется в строгом соответствии с текстом, имеющемся на рейтинговом сервере.

<META name="keywords" content="META, HTML, WWW, Web, паутина, поиск, определение, рекомендации, примеры использования, учебник, руководство, информация, справка, Netscape, Microsoft Internet Explorer">

Список терминов и ключевых слов это то, что является самым главным при индексировании Вашего сайта поисковой машиной! Длина содержимого тегов META "keywords" не должна превышать 1000 символов

<META name="description" content="The best homepage">

Краткое описание Вашего сайта, используемое поисковым сервером для индексирования, и, как правило, вставляемое в текст страницы найденных совпадений в качестве описания Вашего сайта. Длина содержимого тегов META "description" не должна превышать 200 символов

<META HTTP-EQUIV="imagetoolbar" CONTENT="no">

Отключает Панели управления изображениями. Обычно используют при выводе баннеров, фона картинки, карты изображения и др., когда вывод данной панели нежелателен.

Основы CSS. Знать обязательно

Основным понятием CSS является стиль – т. е. набор правил оформления и форматирования, который может быть применен к различным элементам страницы. В стандартном HTML для присвоения какому-либо элементу определенных свойств (таких, как цвет, размер, положение на странице и т. п.) приходилось каждый раз описывать эти свойства, даже если на одной страничке должны располагаться 10 или 110 таких элементов, ничуть не отличающихся один от другого. Вы должны были десять или сто десять раз вставить один и тот же кусок HTML-кода в страничку, увеличивая размер файла и время загрузки на компьютер просматривающего ее пользователя.

CSS действует другим, более удобным и экономичным способом. Для присвоения какому-либо элементу определенных характеристик вы должны один раз описать этот элемент и определить это описание как стиль, а в дальнейшем просто указывать, что элемент, который вы хотите оформить соответствующим образом, должен принять свойства стиля, описанного вами. Удобно, не правда ли?

Более того, вы можете сохранить описание стиля не в тексте вашей странички, а в отдельном файле – это позволит использовать описание стиля на любом количестве Web-страниц. И еще одно, связанное с этим, преимущество – возможность изменить оформление любого количества страниц, исправив лишь описание стиля в одном (отдельном) файле.

Кроме того, CSS позволяет работать со шрифтовым оформлением страниц на гораздо более высоком уровне, чем стандартный HTML, избегая излишнего утяжеления страниц графикой.

Давайте рассмотрим, как мы можем воплотить столь замечательные возможности в жизнь.

Практическое освоение CSS

Как вам уже известно, информация о стилях может располагаться либо в отдельном файле, либо непосредственно в коде Web-странички. Расположение описания стилей в отдельном файле имеет смысл в случае, если вы планируете применять эти стили к большему, чем одна, количеству страниц. Для этого нужно создать обычный текстовый файл, описать с помощью языка CSS необходимые стили, разместить этот файл на Web-сервере, а в коде Web-страниц, которые будут использовать стили из этого файла, нужно будет сделать ссылку на него. Делается это с помощью тега <LINK>, располагающегося внутри тега <BODY> ваших страниц:

<LINK REL=STYLESHEET TYPE="text/css" HREF="URL">

Первые два параметра этого тега являются зарезервированными именами, требующимися для того, чтобы сообщить браузеру, что на этой страничке будет использоваться CSS. Третий параметр – HREF= «URL» – указывает на файл, который содержит описания стилей. Этот параметр должен содержать либо относительный путь к файлу – в случае, если он находится на том же сервере, что и документ, из которого к нему обращаются – или полный **URL («http://...»)** в случае, если файл стилей находится на другом сервере.

Второй вариант, при котором описание стилей располагается в коде Web--странички, внутри тега <BODY>, в теге **<STYLE type="text/css">... </STYLE>**. В этом случае вы можете использовать эти стили для элементов,

располагающихся в пределах странички. Параметр **type="text/css"** является обязательным и служит для указания браузеру использовать CSS.

И третий вариант, когда описание стиля располагается непосредственно внутри тега элемента, который вы описываете. Это делается с помощью параметра **STYLE**, используемого при применении CSS с большинством стандартных тегов HTML. Этот метод нежелателен, и понятно почему: он приводит к потере одного из основных преимуществ CSS – возможности отделения информации от описания оформления информации. Впрочем, если необходимо описать лишь один элемент, этот вариант расположения описания стилей также вполне применим.

Давайте рассмотрим механизм, с помощью которого стили присваиваются элементам Webстраниц. Самый простой случай присвоения какому-либо элементу определенного стиля выглядит так:

НАЗВАНИЕ_ЭЛЕМЕНТА {свойство: значение;},

Где НАЗВАНИЕ_ЭЛЕМЕНТА – имя HTML-тега (H1, P, TD, A и т. д.), а параметры в фигурных скобках – список свойств элемента и присвоенных им значений. Более подробно команды языка CSS мы рассмотрим чуть позже.

Пример1:

H1 {font-size: 30pt; color: blue;}

В этом примере всем заголовкам на странице, оформленным тегом H1, присваивается размер шрифта 30 пунктов и синий цвет.

Также элементы страниц, созданные с использованием CSS, используют механизм наследования: т. е. если вы располагаете изображение внутри тега <P>...</P>, оформленного с помощью CSS, с отступами, так, чтобы параграф занимал только определенную часть ширины страницы, изображение также унаследует значения отступов, указанные для этого параграфа.

CSS реализует возможность присваивать стили не всем одинаковым элементам страницы, а избирательно – для этого используется параметр **CLASS = "имя класса"** или идентификатор **ID=«имя элемента»**, присваивающиеся любому элементу страницы. Рассмотрим эти возможности подробнее.

Пример (в данном примере, сперва в **body** мы идентифицировали блок **div**, назначив ему имя, а затем прописали его стили в **head**:

Использование таблицы стилей в head . Листинг 1.36

```
<head>
<STYLE type="text/css">
#nameofstyle {font-size: 14px;}
</STYLE>
</head>
<body>
<div id="nameofstyle"> текст </div>
<body>
```

Параметр **CLASS** применяется в случае, если необходимо создать одинаковый стиль для нескольких, но не всех элементов страницы (одинаковых или разных).

Пример:

.b-c {font-weight: bold; text-align: center}
– описание стиля для класса b-c

Все элементы класса b-c будут отображаться жирным шрифтом с выравниванием по центру страницы (или ячейки таблицы).

<P CLASS="b-c">Текст параграфа</P>
– параграфу присвоен стиль класса b-c.

<TD CLASS="b-c">текст</TD>
– ячейке таблицы присвоен стиль класса b-c.

Содержащийся в ячейке текст будет отображаться согласно описанию класса.

Таким образом, вы можете присвоить описанный стиль любым текстовым элементам страниц. Обратите внимание, что при написании названия классов необходимо соблюдать регистр символов, согласно тому, как вы назвали класс в описании стиля!

Присвоение стилей с помощью идентификаторов применяется в случае, если данному идентификатору соответствует только один элемент на странице. Если элементов, которым необходимо присвоить такой стиль, несколько – это уже класс.

Свойства элементов, управляемых с помощью CSS

СВОЙСТВА ШРИФТА	
font-family	Используется для указания шрифта или шрифтового семейства, которым будет отображаться элемент. P {font-family: Times New Roman, sans-serif;}
font-weight	Определяет степень жирности шрифта с помощью трех параметров: lighter, bold, bolder

	B {font-weight: bolder;}
font-size	Устанавливает размер шрифта. Параметр может указываться как в относительной (проценты), так и абсолютной величине (пункты, пикселы, сантиметры) H1 {font-size: 200%;} H2 {font-size: 150px;} H3 {font-size: 400pt;}
font-style	Стиль шрифта. Два вариатна отображения: normal (обычный) и italic (курсив) H1 {font-style: normal;} H2 {font-style: italic;}
РАСПОЛОЖЕНИЕ	
display	display:block - Элемент показывается как блочный. Применение этого значения для встроенных элементов, например тега , заставляет его вести подобно блокам — происходит перенос строк в начале и в конце содержимого. display:inline - Элемент отображается как встроенный. Использование блочных тегов, таких как и <p>, автоматически создает перенос и показывает содержимое этих тегов с новой строки. Значение inline отменяет эту особенность, поэтому содержимое блочных элементов начинается с того места, где окончился предыдущий элемент. display:none – скрывает элемент. display:table Определяет, что элемент является блочной таблицей подобно использованию тега <table>. display:table-cell: ll Указывает, что элемент представляет собой ячейку таблицы (тег <td> или <th>).
float	Устанавливает размер шрифта. Параметр может указываться как в относительной (проценты), так и абсолютной величине (пункты, пикселы, сантиметры) img1 {float: left;} img2 {float: right;} img3 {float: none;}
position	Параметр управления расположением блока. H1 {position: static;} Обычное расположение блока. H2 {position: absolute;} Блок находится на странице не зависимо от других блочных элементов. H3 {position: relative;} Позиция блока отсчитывается от исходного H4 {position: fixed;} Блок не прокручивается ползунком, всегда находится в одном и том же месте экрана.
visibility	Параметр отображения блока. H1 {visibility: visible;} Блок виден

	H2 {position: hidden;} Блок невиден.
z-index	Наложение одного блока на другой. H1 {z-index: 2;} Чем больше число тем выше окажется блок.
ФОН И РАМКА	
color	Определяет цвет элемента H1 {color: yellow;}
background-image	Устанавливаем на фон картинку. Причем, прописываем путь не от страницы, где она будет находится, а от файла стиля. P {background-image:url(/photo.jpg);}
background-attachment	Параметр отвечающий за прокручивание или фиксацию фона. P {background-attachment: scroll;} Прокручивание. P {background-attachment: fixed;} Фиксация.
background-color	Цвет фона P {background-color: green;}
background-position	Выравнивание изображения P {background-position: left-top;} P {background-position: center-bottom;} P {background-position: 20px 50px;}
background-repeat	Размножение изображения P {background-repeat: repeat;} Размножение по горизонтали и вертикали. P {background-repeat: no-repeat;} Не размножать. P {background-repeat: repeat-x;} Размножать только по горизонтали. P {background-repeat: repeat-y;} Размножать только по вертикали.
border	Цвет, толщина и стиль границы. Все три значения обязательные. P {border: red 1px solid;} Сплошная рамка красного цвета толщиной в 1 px. P {border: red 1px dotted;} Штрихпунктирная рамка красного цвета толщиной в 1 px. P {border: red 1px dashed;} Точечная. P {border: red 1px hidden;} Спрятать линии. Можно записать так: P {border-color: red;} P {border-width: 1px;} P {border-style: solid;} Рамка с одной стороны: P {border-top: green 3px solid;}
СВОЙСТВА АБЗАЦА	
text-decoration	Устанавливает эффекты оформления шрифта, такие, как подчеркивание или зачеркнутый текст H4 {text-decoration: underline;} A {text-decoration: none;} .wrong {text-decoration: line-through;}

text-align	Определяет выравнивание элемента. P {text-align: justify} H1 {text-align: center}
text-indent	Устанавливает отступ первой строки текста. Чаще всего используется для создания параграфов с табулированной первой строкой. P {text-indent: 50pt;}
line-height	Управляет интервалами между строками текста. P {line-height: 50px;}
letter-spacing	Расстояние между буквами по горизонтали. Значение - в единицах измерения или normal P {letter-spacing: 5px;}
white-space	Параметр отвечающий за восприятие пробелов. P {white-space: pre;} воспринимается каждый пробел, т.е. если в коде стоит три пробела, то и броузер будет отображать три пробела. P {white-space: normal;} Обычное восприятие пробелов, т.е. любое количество пробелов броузер отобразит как один. P {white-space: norap;} Неразрывные пробелы.
margin	Отступ внешний, т.е. от рамки до содержимого обтекающего рамку. Значения - в единицах измерения. P {margin:10px;} со всех сторон одинаковый отступ P {margin:10px 3px 5px 5px;} Сверху - 10px, справа - 3px, снизу - 5px и слева - 5px. Тоже самое можно записать так P {margin-top:10px;} P {margin-right:3px;} P {margin-bottom:5px;} P {margin-left:5px;}
padding	Внутренний отступ, от рамки до ее содержимого. Значения параметров те же, что у margin. Если рамка невидимая, то лучше использовать этот параметр, т.к. он лучше воспринимается броузерами.
ЕДИНИЦЫ ИЗМЕРЕНИЯ	
px	Пикселы
cm	Сантиметры
mm	Миллиметры
pt	Пункты (типограф.)
%	Проценты
em	Размер, привязанный к стандартному размеру броузера

Система приоритетов

Наиболее приоритетными являются стили для конкретных тэгов

Пример: **<p style="color: black">**

Потом идет приоритет блоков **div**.

Пример: **#name {color: black;}**

Потом - приоритет классов

Пример: **.nameclass {color: black;}**

Потом - приоритеты общие для тэгов.

Пример: **p {color: black;}**

Но любые стили из блоков, классов или общих стилей для тэгов можно сделать приоритетными по отношению к конкретным тэгам. Для этого, после описания стилей, достаточно вставить **!impotent**, и этот стиль станет приоритетным.

Отображаем содержимое не помещающееся в блок

- **overflaw: visible** (содержимое заходит за пределы блока, но IE растянет сам блок.

  Чтобы этого не произошло, используем:

- **overflaw: hidden** (выступающая часть прячется за пределами блока)
- **overflaw: scroll** (появляется полоса прокрутки).

Псевдоклассы

Чаще всего используются при работе с ссылками:

a.link (не посещенная)

a.visited (посещенная ссылка)

a.hover (на ссылку наведен курсор мыши)

a.active (активная ссылка)

Псевдоклассы для ссылки. Листинг 1.5

```
<head><style type="text/css">
a.lin {color: #993300;}
a:hover {color: #5C743D;}
a:visited { color: green;}
a:active {color:black;}
</style>
</head>
<body>
```

```
<p>
<a class="lin" href="#"> ссылка </a>
</p>
</body>
```

Тип носителя

В одном документе может быть несколько стилей для разных носителей.

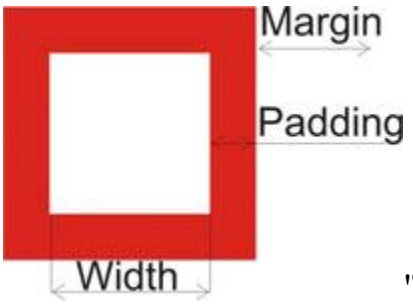
@media scrin { p {...}} данный тип применяется для экрана

@media print { p {...}} применяется для печати

@media handed { p {...}} применяется для КПК.

Особенности задания размеров элементов

Разные браузеры по разному воспринимают стилевые параметры.



"Правильные" браузеры должны воспринимать элемент ** ** так как показано на рисунке. Но **IE**, причем все версии, включает в ширину рабочей области и двухсторонний **padding**. Для того, чтобы избавиться от неприятных багов, которые могут возникнуть от этой неприятной особенности IE, достаточно прописать декларацию **Doctype** (желательно xhtml) в начале документа. Если документ, даже при наличии Doctype начинается с <? xml...>, то IE опять перейдет в свой режим отображения.

Избавляемся от зазора между и картинкой

По умолчанию любой встраиваемый в html элемент, в том числе и картинка - это инлайновый элемент. Поэтому между картинкой и текстом появляется зазор равный зазору между строками текста. Чтобы избавиться от этого зазора, переводим этот элемент в блочный: **display: block**

Вставка изображения

По сравнению с html, в css намного больше средств по управлению изображениями.

За отображение фонового рисунка отвечают 4 свойства:

background-image

background-repeat

background-attachment

background-position

В **background-image** мы прописываем путь до фонового изображения. Пример:

body {background-image: url(picter.jpg)}

- В **background-repeat** прописываем алгоритм повторения фонового изображения. Возможны следующие значения:

  repeat - изображение повторяется по вертикали и по горизонтали, заполняя всю площадь отведенного пространства

  repeat-x - изображение повторяется только по горизонтали

  repeat-y - изображение повторяется только по вертикали

  no-repeat - изображение не повторяется

- **background-attachment** отвечает за прокрутку изображения. Значения:

  fixed - изображение не прокурчивается вслед за скролингом

  scroll - изображение прокурчивается

- Значение **background-position** отвечает за расположение фонового изображения. Обозначается цифрами или ключевыми значениями. **Цифры**: используются любые, абсолютные или относительные единицы. Если задано одно число, например, 10%, то подразумевается горизонтальное смещение. Два числа, например 10% 20% означают смещение по горизонтали на 10%, а по вертикали на 20%. **Ключевые слова**: left, right, top, bottom, center. Позиция по умолчанию: **0% 0%** или **top left**

Суммируя все вышесказанное, вставим изображение:

Вставка изображения . Листинг 1.37

```
<style type="text/css">
#fon { background-image: url(picters/dog.jpg);
background-attachment:scroll;
background-position:bottom;
background-repeat:no-repeat; height:461px;}
</style>
</head>
<body>
<div id="fon"> text</div>
</body>
```

Обратите внимание на то, что помимо вышерассмотренных значений свойств, в стилях присутствует значение **height**. Этот параметр (вместе с **width**) желательно прописывать, чтобы все браузеры одинаково воспринимали размеры рисунка и не подгоняли под заданные по умолчанию.

В CSS существует обобщенное свойство **background**, благодаря которому данную характеристику картинки можно записать более кратко:

Свойство background . Листинг 1.38

```
#fon {background: url(picters/dog.jpg) no-repeat fixed bottom height:461px;}
```

Трюки с позиционированием

Выравниваем блок (класс) по центру.

Только в режиме **strict** (см. Doctype-декларацию). В стилях нужно прописать следующее **{margin-left:auto; margin-right:auto;}**

Еще один вариант: предположим нам надо выровнять блок шириной 200px. Можно это сделать прописав в стилях: **{position: absolute; left: 50%; margin-left:-100px;}**

Позиционируем один блок по отношению к другому.

В **body** прописываем стандартное

* *

* текст *

* *

В стилях пишем следующее:

#block1 **{position: relative;}**

#block2 **{position: absolute;}** (прописать координаты)

Таким образом, мы сообщаем браузеру, что все координаты второго блока должны отсчитываться от первого. Точка второго блока с координатами {0, 0} совпадет с верхней левой точкой первого блока.

3-колоночная верстка с резиновым центром.

Создаем три колонки, причем левую и правую с фиксированными значениями. Причем важно, чтобы первой шла левая колонка, второй - правая, последней - резиновая колонка.

Итак, в body прописываем

* *

* *

* *

В head прописываем позиционирование блоков:

#left **{width: 200px; float: left;}**

#right **{width: 200px; float: right;}**

#main **{margin-left: 210px; margin-right: 210px;}**

Полностью резиновая верстка.

#t1 **{widght: 33%; float:left;}**

#t2 **{widght: 33%; float:left;}**

#t3 **{widght: 33%; float:left;}**

Получим три резиновые колонки, занимающие 99% ширины страницы. Оставшийся процент зальем цветом, сходным с цветом колонки. Для более-менее адекватного восприятия браузерами страницы, желательно по ширине всегда оставлять пару пикселей или процентов не занятыми блоками.

Выравниваем колонки.

Чтобы выровнять плавающие колонки по высоте иногда достаточно все колонки взять в один большой блок (**)**, а внутрь этого блока, в конце, вставить еще один ** **

* *

В стилях пропишем следующие:

#clear **{clear: left;}** Таким образом все плавающие левые колонки выровняются.

Блочная верстка

В довершение главы акцентирую Ваше внимание на еще одном способе верстки сайта, ранее в учебнике я приводил примеры верстки страницы при помощи таблиц, теперь же имея накопленный багаж знаний, настало время приступить к верстке сайта, используя блоки div и свойства CSS.

Разберём классический макет верстки сайта из трёх колонок, а так же "шапки" и "подвала".. в котором правая и левая колонки имеют фиксированную ширину, а центральная колонка "резиновая" т.е. занимает собой всё оставшееся место.

Взгляните на пример:

Блочная верстка web-странички . Листинг 1.39

```
<html>
<head>
<title>Блочная вёрстка</title>
</head>
<body style="background: #cc0; margin:0;">
<div style="clear:both; background: #0c0; padding:
5px;">Логотип</div>
<div style="float: left; background: #c0c; padding: 5px;
width: 170px">Меню</div>
<div style="float: right; background: #c0c; padding: 5px;
width: 170px">Реклама</div>
<div style="margin:0 180px; background: #0cc; padding:
5px;">Основное содержание<br><br><br><br><br> И ещё куча
текста..</div>
<div style="clear:both; background: #0c0; padding:
5px;">Подвал</div>
</body>
</html>
```

Пояснения:

<body style="background: #cc0; margin:0;"> - Используем margin:0 для того чтобы обнулить поля в окне браузера.

Логотип - Создаём контейнер с будущим логотипом и запрещаем его обтекание с обеих сторон, используя clear:both, теперь что бы не случилось, последующие блоки будут идти снизу, а шапка сайта как ей и положено будет располагаться сверху.

Меню - Левая колонка с "Меню" выровнена по левому краю свойством float: left и имеет фиксированный размер в 170 пикселей.

Реклама - Правая колонка с рекламой выровнена по правому краю свойством float: right; и тоже имеет фиксированный размер 170 пикселей.

Основное содержание - Центральная колонка никак не выравнивается, но занимает своё место по центру, так как имеет широкие поля слева и справа margin:0 180px, тем самым не накладываясь на правую и левую колонки. Почему поля 180 пикселей, а не 170 в соответствии с ширенной колонок "Меню" и "Реклама"? Отвечаю: потому что кроме ширины width: 170px эти колонки имеют ещё и отступы со всех сторон padding: 5px складываем 170+5+5=180.

Подвал - Ну и "подвал" блок, в котором, как правило, располагаются контактные данные и авторские права, так же как логотип запрещает на всякий случай обтекание слева и справа clear:both и тем самым устремляется вниз страницы.

Практические советы

1. Используй <!doctype html> **DOCTYPE** (прописывается первой строчкой в любом документе). Это необходимо, чтобы : браузер понял с каким типом документов он имеет дело; валидатор смог протестировать страницу.
2. Используй тег **META**. Задавай язык и кодировку документа (см. предыдущую лекцию).
3. **Кодировка самого файла** должна совпадать с кодировкой указанной в META.
4. Каждая страница должна иметь элемент **<title></title>** (содержимое тэга отображается в строке заголовка браузера), причем в нем должно отображаться заглавие именно этой страницы, а не документа в целом. Больше всего поисковые машины любят именно title.
5. Каждая страница, помимо title должна иметь свой **description** (META тег)

6. Для разметки документа используй те элементы, которые описывают структуру документа, а не его внешний вид. Например, для отступа нужно использовать элемент **<margin>** или **<padding>**, но ни как не элемент **<blockquote**>, который предназначен для выделения цитат. Используй элементы по назначению. Такие теги, как , <font>, <color> считаются устаревшими. HTML5 их уже не поддерживает. Устаревают теги форматирования, зато появляются новые теги структурные (<header>, <footer>, <nav>…). Иди в ногу со временем.
7. Избегай большого количества пробелов и вводов: при грамотной разметки документа не должно возникнуть необходимости более чем в одном пробеле или вводе.
8. Используй по необходимости комментарии *<! -текст комментарий ->*.
9. Безопасных цветов не существует. Связано это с тем, что ограничение в цветопередачи зависит только от принимающего устройства (например, монитора). Поэтому можно использовать абсолютно любые цвета.
10. Избегай пользоваться теми элементами, значение которых можно заменить атрибутами. Пример: вместо элемента ** ** предпочтительнее использовать **<align="center">**, и т.д. И вообще, избегай использовать устаревшие элементы. Даже align=”center” можно заменить стилями.
11. Для работы над внешним видом документа используй **CSS**, а с помощью **HTML** осуществляй только разметку.
12. Ограничь использование табличной верстки **HTML**. Используй верстку на основе **CSS.**
13. Использование фреймов - признак дурного тона.
14. Тестируй сайт как минимум в трех браузерах: **Mozilla, IE, Opera** причем для тестирования не используй последние версии браузеров.
15. Ты выбрал ту профессию, в которой, чтобы остаться на плаву, каждый день необходимо узнавать что-то новое, и быть способным применить эти знания в борьбе с конкурентами. Поэтому, даже если ты достиг вершины мастерства, **продолжай совершенствоваться.**
16. Просто делать хорошие сайты, и просто быть хорошим кодером – недостаточно. Старайся превзойти себя. Если ты будешь постоянно прилагать усилия в этом направлении, ты совершишь массу ошибок. Но в этом и есть

Путь web-мастера. Тот, кто не совершает ошибок, ничему не учится.

2. Инструментарий web-мастера.

- Блакнот Notepad++
- XAMPP
- Firefox, который по-праву считается браузером для разработчиков
- Firebug – плагин для firefox
- Ftp-клиент, например, плагин для firefox – fireFTP

После установки XAMPP, необходимо зайти в директорию c://xampp/htdocs. В этой дирректории создадим папку, например, kurs, где будут размещаться файлы и каталоги сайта.

Файл index.php должен располагаться в корне директории kurs.

Вот примерная стуркура любой cms (системы управления сайтом):

- templates - папка для шаблонов
- class – папка для классов
- utils – папка для функций
- config – папка для конфигурационных файлов (например, соединение с базой данных)
- img – папка для изображений
- styles – папка для стилей css
- adminka – папка для файлов, отвечающих за администрирование (текствое наполнение, редактирование, удаление контента, блока меню, стилей и изображений сайта).
- другие папки, например, для форума, плагинов, прайсов, электронных книг и т.д.

После настройки необходимых папок и файлов, необходимо настроить базу данных, и связать нашу базу данных с сайтом через конфигурационный файл config.php

Прежде чем это сделать, давайте разберемся что такое MySQL и как пользоваться этой базой данных.

3. MySQL + PHP.

MySQL – реляционная база данных.

MySQL - это система управления реляционными базами данных. В реляционной базе данных данные хранятся в отдельных таблицах, благодаря чему достигается выигрыш в скорости и гибкости. Таблицы связываются между собой, есть возможность объединять при выполнении запроса данные из нескольких таблиц. Связь осуществляется посредством естественного (номер паспорта, в/у…) или суррогатного (auto_increment) ключа. Никакие записи таблицы не могут иметь двух одинаковых записей ключа.

Основные типы данных MySQL:

- Числовые: tinyint (возраст человека, кол-во детей у одних родителей, кол-во учеников в одном классе), smallint (кол-во страниц в книге, кол-во учеников в школе), int (кол-во людей на Земле), bigint (кол-во молекул во Вселенной) , bool (0 или 1), float (число с точкой).
- Дата и время: date, time, datatime, timestamp, year.
- Строковые значения: tinytext (путь к файлу, фамилия, название книги), text (новостной блок), longtext (большая статья), enum (‘value1’, ‘value2’, ‘value3’)

Создание, удаление, редактирование таблиц, столбцов в таблице происходит в самом phpMyAdmin, вкладка «Структура»

Необходимый минимум, для работы с базой MySQL

- Вставка строковых значений:

INSERT INTO catalog VALUES (1, ‘Школа ПХП’); //либо

INSERT INTO catalog VALUES (1, «Школа ПХП»);

- Вставка числовых значений в таблицу:

INSERT INTO inttable VALUES (10, 20); //либо

INSERT INTO inttable (id_cat, id) VALUES (10, 20);

- Вставка календарных значений в таблицу:

INSERT INTO catalog VALUES (1, «2010-06-9»); //ГГГГ-ММ-ДД

- Механизм AUTO_INCREMENT – автоматическая генерация ключа
- Обновление записей

UPDATE catalogs SET name = ‘Школа ПХП’

WHERE name = ‘ШКОЛА’

- Выборка данных.

SELECT * FROM catalogs;

SELECT * FROM catalogs WHERE id_catalog > 2;

SELECT * FROM catalogs WHERE id_catalog > 2 AND id_catalog <= 4;

SELECT * FROM catalogs WHERE id_catalog BETWEEN 2 AND 4;

SELECT * FROM catalogs WHERE id_catalog NOT BETWEEN 2 AND 4;

SELECT * FROM catalogs WHERE id_catalog IN (1, 2, 5);

SELECT * FROM catalogs WHERE id_catalog NOT IN (1, 2, 5);

SELECT * FROM catalogs WHERE name = ‘школа’;

SELECT * FROM catalogs WHERE name LIKE ‘%ы’;

- Сортировка записей

SELECT id_catalog, name FROM catalogs ORDER BY id_catalog DESC;

SELECT * FROM tbl ORDER BY id_catalog DESC, putdate DESC;

- Вывод записей в случайном порядке

SELECT id_catalog, name FROM catalogs ORDER BY RAND();

- Ограничение выборки

SELECT id_catalog, name FROM catalogs ORDER BY RAND() LIMIT 1;

SELECT id_catalog, name FROM catalogs ORDER BY id_catalog DESC LIMIT 2, 2;

- Вывод уникальных значений

SELECT DISTINCT id_catalog FROM tbl ORDER BY id_catalog;

- Математические функции (практически не используется)
- Округление результатов вычисления

SELECT ROUND (A*B*SIN(RADIANS(angle))/2/0, 3) AS scool FROM triangle;

SELECT CEILING (0,49), CEILING (1,51), CEILING (-0,49), CEILING (-49,9);

SELECT FLOOR(0,49), FLOOR(1,51), FLOOR (-0,49), FLOOR (-49,9);

- Вычисления

SELECT fio, (TO_DAYS(NOW()) – TO_DAYS(putdate))/365.25

- Преобразование кодировки

SELECT CONVERT (‘Школа PHP’ USING koi8r);

- Первые несколько символов строки.

SELECT SUBSTRING(fio, 1, 5) FROM tbl;

SELECT CONCAT (family, ‘ ‘, SUBSTRING (name, 1,1), ‘.’, SUBSTRING (patronymic, 1,1), ‘.’)

- Перевод первого символа в верхний регистр

UPDATE tbl SET

family = CONCAT(UPPER(SUBSTRING(family, 1, 1)), SUBSTRING(family,2)),

name = CONCAT(UPPER(SUBSTRING(name, 1, 1)), SUBSTRING(name,2)),

patronymic = CONCAT(UPPER(SUBSTRING(patronymic, 1, 1)), SUBSTRING(patronymic,2)),

SELECT * FROM tbl;

- Обратимое и необратимое шифрование

SELECT AES_ENCRYPT (‘MySQL’, ‘ключ’);

SELECT AES_DECRYPT (‘*1VASF|)*)ADSF ADSF_+_*’, ‘ключ’);

SELECT MD5(‘MySQL’);

- Агрегатные функции

SELECT COUNT(id), COUNT(value) FROM tbl;

SELECT COUNT(*) FROM tbl WHERE value < 300;

SELECT MIN(price) FROM products;

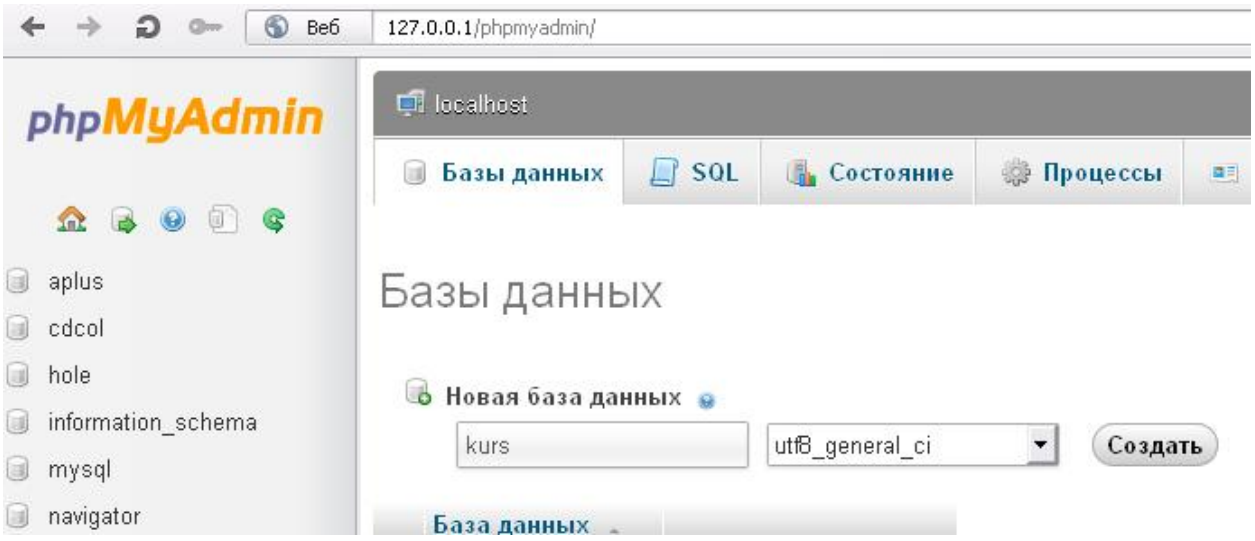
SELECT SUM(price) FROM products;

Приступим к созданию базы данных.

После запуска XAMPP, откроем firefox и наберем в адресной строке 127.0.0.1

Находим PHPMyAdmin.

Переходим во вкладку Базы данных, в поле «Новая база данных» пишем имя нашей базы. Затем выбираем кодировку (желательно utf8-general-ci, т.к. utf8 считается универсальной кодировкой). Нажимаем кнопку «Создать». Таким образом мы создали пустую базу данных. Сейчас необходимо ее заполнить данными.



Далее мы можем либо импортировать уже готовую базу (файл с расширением .sql), либо создавать таблицы и наполнять их значениями вручную.

Весь курс PHP+MySQL можно свести к следующим действиям:

- Выбор из базы данных:
  - Одной записи.
  - Всех записей.
  - Множества записей, постранично.
- Удаление.
- Редактирование.
- Вставка.

После создания базы данных и заполнения таблиц значениями, необходимо ее подключить к PHP.

В папке config создадим файл config.php, который будет отвечать за соединение с базой данных.

Реализация файла config.php. Подключение базы данных. Листинг 3.1

```
// сейчас выставлен сервер локальной машины
  $dblocation = "localhost";
  // Имя базы данных, на хостинге или локальной машине
  $dbname = "hole";
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";
// Устанавливаем соединение с базой данных
  $dbcnx = mysql_connect($dblocation,$dbuser,$dbpasswd);
  if(!$dbcnx)
    exit("<P>В настоящий момент сервер базы данных не
          доступен, поэтому корректное отображение
          страницы невозможно.</P>" );
  // Выбираем базу данных
  if(! @mysql_select_db($dbname,$dbcnx))
    exit("<P>В настоящий момент база данных не доступна,
          поэтому корректное отображение страницы
          невозможно.</P>" );
  @mysql_query("SET NAMES 'utf8'");
```

Функция mysql_connect отвечает за соединение с сервером баз данных. Принимает три входящих параметра: путь к серверу (как правило, это localhost), имя пользователя (локально – это root - суперпользователь) и пароль. Если по каким-то причинам, соединение не установилось (неправильный пользователь, пароль либо недоступен сервер), то функция exit() останавливает дальнейшее выполнение сценария и выводит на экран ошибку *"<P>В настоящий момент сервер базы данных не доступен, поэтому корректное отображение страницы невозможно.</P>*". После подключения к базе данных, функция mysql_select_db выбирает нужную базу данных. Если по каким-то причинам база не доступна, то exit() выведет другую ошибку "*<P>В настоящий момент база данных не доступна, поэтому корректное отображение страницы невозможно.</P>*".

После соединения с базой данных, можно использовать функцию mysql_query, через которую можно выполнять любые sql-запросы. Знак собачки @ перед функцией подавляет вывод ошибки на экран, если она не критичная.

Создание шаблона

Любую web-страничку можно визуально разбить на три части: шапку, подложку и основное текстовое содержимое.

Шапку сайта – все то, что располагается выше контента, поместим в файл top.php

Все что ниже контента – уходит в bottom.php. Сам контент – в index.php

Напомню, что индексирование сайта начинается с файла index.php. Поэтому к индексному файлу нужно подключить и top.php и bottom.php

index.php. Листинг 3.2

```
<?php
 require_once(“templates/top.php”);
// основной контент web-страницы находится тут.
require_once(“templates/bottom.php”);
?>
```

В top.php, помимо html-шаблона, должно быть подключение файла config.php:

top.php. Листинг 3.3

```
<?php
 require_once(“config/config.php”);
?>
<html>
 … html-шаблон.
```

В bottom.php находятся только закрывающиеся тэги шаблона.

После того, как шаблон готов и база настроена, приступим к выводу значений из таблицы базы данных.

Выборка данных из таблицы catalog базы данных, файл index.php. Листинг 3.4

```
<?php
 require_once(“templates/top.php”);
$query = "SELECT * FROM catalog
              WHERE hide = 'show' AND id_parent = 49
              ORDER BY pos";
    $cat = mysql_query($query);
    if (!$cat) exit("Ошибка при обращении к каталогу");
    if(mysql_num_rows($cat))
    {
      $catalog = mysql_fetch_array($cat);
```

```
        echo "<a style=\"border-left:1px solid #999999\"
href=index.php?id_catalog=$catalog[id_catalog]
                    class=\"menu_lnk\">$catalog[name]</a>";
    }
require_once("templates/bottom.php");
?>
```

Данный листинг требует пояснения: Сперва мы формируем переменную $query. Затем функция mysql_query выполняет запрос, содержащийся в переменной $query. Если запрос выполняется успешно, то переменная $cat будет содержать значение true. Если запрос, по каким-либо причинам, не может выполниться, то значение переменной $cat будет false, т.е. этой переменной не будет. Далее мы проверяем if(!$cat) – если переменной $cat не существует (в PHP ! – это символ отрицания), с помощью функции exit() останавливаем сценарий и выводим ошибку на экран. Важно понимать, что переменная $cat не будет существовать только в том случае, если был не правильно составлен запрос к базе. Если же запрос возвращает нулевой результат, то это не ошибка. Просто не будет выполняться условие mysql_num_rows($cat). Предположим, что результат не нулевой, тогда сценарий с помощью функции mysql_fetch_array($cat) создаст ассациативный массив $catalog, где ассациациями будут выступать имена строк таблицы catalog.

Пример выборки множества значений

Выборка данных с использованием цикла while. Листинг 3.5

```
 $query = "SELECT * FROM catalog
              WHERE hide = 'show'  ORDER BY pos";
    $cat = mysql_query($query);
    if (!$cat) exit("Ошибка при обращении к каталогу");
    if(mysql_num_rows($cat))
    {
      while($catalog = mysql_fetch_array($cat))
      {
        echo "<a style=\"border-left:1px solid #999999\"
href=index.php?id_catalog=$catalog[id_catalog]
                    class=\"menu_lnk\">$catalog[name]</a>";
      }
    }
```

Данный листинг точно такой же, как и предыдущий, за исключением того, что массив $catalog будет создаваться циклом while до тех пор, пока не пройдется по всем записям из запроса.

POST и GET. Передача данных

Существует два основных вида HTTP-запросов: POST и GET. Задача данных запросов – передача значений сценарию (от одной web-странички – к дргой, либо от пользователя - серверу).

Разница между вариантами GET и POST состоит только в том, как информация передается из формы в обрабатывающий сценарий. Метод GET посылает

всю собранную информацию как часть адреса URL. Метод POST передает ин формацию так, что пользователь этого не видит. Например, при использова нии метода GET, после передачи информации URL примет следующий вид:

http://www.colony.by/blogtheme.php?id=60&idbloger=130.

При использовании метода POST конечный пользователь увидит только та-кую запись:

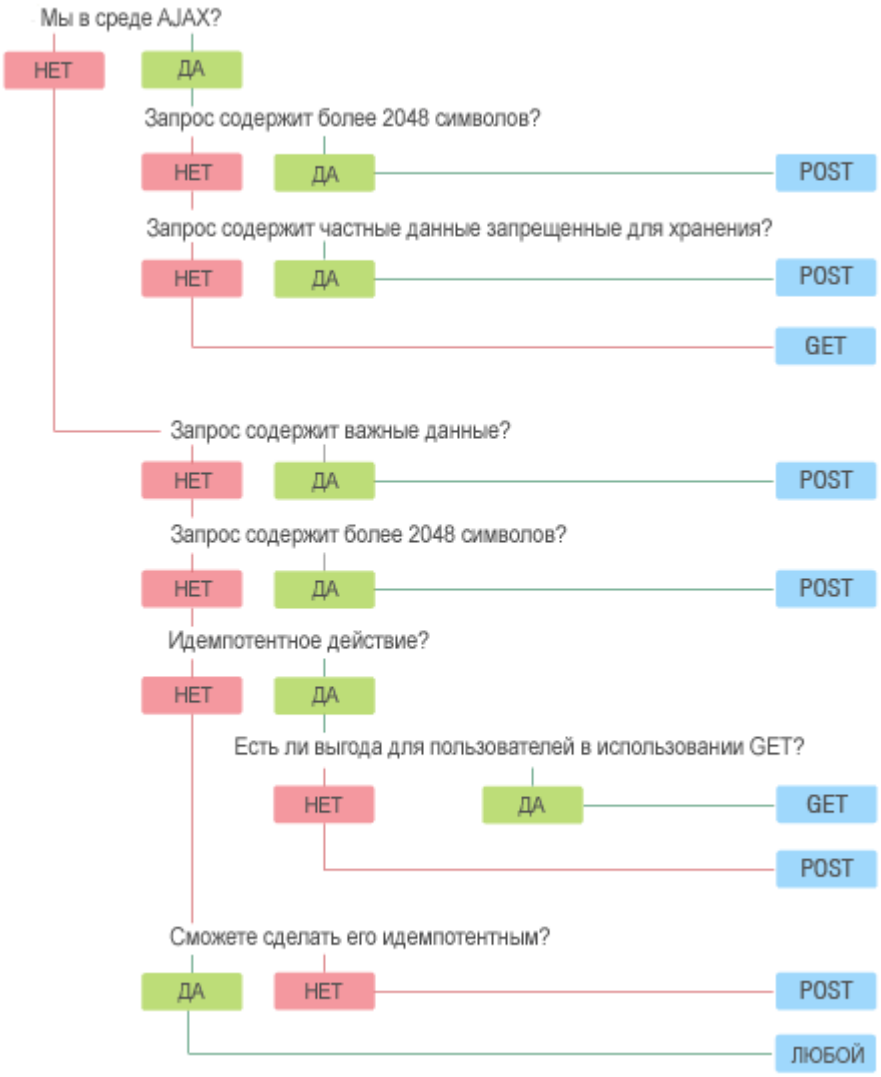
http://www.colony.by/news.php.

При выборе метода следует учитывать три фактора:

1. метод GET ограничивает объем передаваемой информации;
2. метод GET открыто пересылает введенную информацию в обрабаты-вающий сценарий, что может неблагоприятно сказаться на безопасно-сти. Например, каждый человек, которому виден монитор вашего ком-пьютера, может заметить введенный в форму пароль;
3. страницу, сгенерированную формой с помощью метода GET, разреша-ется помечать закладкой, а сгенерированную методом POST нет.

Руководство по выбору GET или POST

Давайте сперва введем новое слово в свой словарный запас: термин — *идем-потентный* (Не стоит лезть в википедию за его трактовкой, идемпотентность это свойство объекта проявляющееся в том, что повторное действие над этим объектом не изменяет его) и рассмотрим следующую схему:



Создание сайта визитки

В шапке сайта (top.php) пропишем следующие ссылки:

Создание ссылок . Листинг 3.6

```
…
<a href=”index.php?url=index”>Главная</a>
<a href=”index.php?url=about”>О компании</a>
<a href=”index.php?url=vacan”>Вакансии</a> и т.д.
…
```

Важно, чтобы у каждой ссылки был уникальный $_GET – параметр. В данном случае мы создали три $_GET переменных: $_GET[‘url’] = index, $_GET[‘url’] = about и $_GET[‘url’] = vacan

Соответственно, по клику на каждую ссылку файл index.php получает различные значения параметра $_GET[‘url’].

В самом файле index.php нам осталось только обработать передаваемые значения.

Index.php Выборка одной строки данных. Листинг 3.7

```
require_once(templates/top.php);
if($_GET[‘url’]) {
$file = $_GET[‘url’];
}
else {
$file = ‘index’;
}
$query = "SELECT * FROM ttext
              WHERE url = '”.$file.”'";
    $cat = mysql_query($query);
    if (!$cat) exit("Ошибка при обращении к каталогу");
    if(mysql_num_rows($cat))
    {
       $catalog = mysql_fetch_array($cat);

        echo  “<h1>”.$catalog[name].”</h1>”;
        echo  $catalog[body];
    }
require_once(templates/bottom.php);
```

В начале листинга идет проверка на существование $_GET[‘url’]. Далее, либо значение этой переменной, либо значение по умолчанию =’index’ передается в sql-запрос.

Сайт-визитка, по сверстанному ранее шаблону – готов.

4. Вспомогательный набор классов FrameWork. Работа с изображениями. Регистрация пользователей.

Любой FrameWork можно представить виде набора классов. Соответсвенно, чтобы пользоваться FrameWork-ом нужно иметь представление о том, как создавать объекты с использованием готовых классов.

Зачем же нужен FrameWork? Помимо решения главной задачи: добиться максимально-повторного использования кода, FrameWork помогает решить следующие задачи:

- Сокращение времени на разработку
- Уменьшает вероятность ошибки
- Удобство пользования
- Проверка пустых полей
- Проверка на соответствие заданном формату.
- Расширение базовой функциональности.

Пример формы

Для создания данной формы воспользуемся FrameWork-ом студии SoftTime.

Сперва создадим файл class.config.php, разместим его в папке config.

В этом файле у нас будет подключение всех классов, используемых в FrameWork-е.

Config.php. Подключение фрэймворка. Листинг 4.1

```
require_once("class/class.field.php");
 require_once("class/class.field.text.php");
 require_once("class/class.field.text.english.php");
 require_once("class/class.field.text.int.php");
 require_once("class/class.field.text.email.php");
 require_once("class/class.field.password.php");
 require_once("class/class.field.textarea.php");
 require_once("class/class.field.hidden.php");
 require_once("class/class.field.hidden.int.php");
 require_once("class/class.field.radio.php");
 require_once("class/class.field.select.php");
```

```
  require_once("class/class.field.checkbox.php");
  require_once("class/class.field.file.php");
  require_once("class/class.field.date.php");
  require_once("class/class.field.datetime.php");
```

После чего, для работы с данным FrameWork, достаточно подключить сам файл class.config.php. Подключим его к top.php.

Подключение class.config.php к top.php. Листинг 4.2

```
<?php
require_once(“config/config.php”);
require_once(“config/class.config.php”);
?>
<html>
 … html-шаблон.
```

Для создания форм с помощью данного FrameWork необходимо выполнить следующие шаги:

1. Создание объекта для каждого элемента формы.
2. Передача созданных объектов массивом в класс form
3. Вывод формы на экран с помощью метода print_form().

Пример создания элемента формы <input type=’text’ name=’name’>

Пример создания двух элементов формы с передаваемыми параметрами в класс field_text. Листинг 4.3

```
$name         = new field_text("name",
                                "Название",
                                true,
                                $_POST['name']);
$email         = new field_text("email",
                                "E-mail",
                                true,
                                $_POST['email']);
```

Ключевое слово new указывает на то, что мы создаем новый объект класса field_text. В скобках, через запятую в класс передается 4 параметра:

1. Значение атрибута name
2. Текстовое содержимое или название элемента формы, которое будет отображатья слева от элемента
3. True либо false. Обязательный либо необязательный элемент для заполнения.
4. Пользовательское значение

При необходимости можно прописать класс для CSS. Вот таким образом:

$name->css_class="название класса".

Передача созданных объектов массивом в класс form(). Листинг 4.4

```
    $form = new form(array("name" => $name,
                           "email" => $email),
                     "Добавить",
                     "field");
```

Наконец, можно приступить к выводу формы на экран.

Вывод созданной формы на экран. Листинг 4.5

```
$form -> print_form();
```

FrameWork может работать с 14-ью типами элементов форм. Рассмотрим их:

1) текстовое поле

```
$name       = new field_text("name",
                             "Название",
                             true,
                             $_POST['name']);
```

2) текстовое поле для английского алфавита

```
$nameenglish       = new field_text_english("nameinglish",
                             "Название",
                             true,
                             $_POST['nameinglish']);
```

3) текстовое поле для ввода цифр

```
$nameint       = new field_int("nameint",
                             "Название",
                             true,
                             $_POST['nameint']);
```

4) текстовое поле для e-maila

```
$nameemail       = new field_email ("nameemail",
```

```
                    "Название",
                    true,
                    $_POST['nameemail']);
```

5) текстовое поле для ввода пороля

```
$pass = new field_password("pass",
                    "Пароль",
                    true,
                    $_REQUEST['pass']);
```

6) текстовое поле для textarea

```
$podrobno = new field_textarea("podrobno",
                    "Подробнее о себе",
                    true,
                    $_REQUEST['podrobno']);
```

7) текстовое поле для ввода цифр

```
$hidden       = new field_hidden("hidden",
                    "Скрытое текстовое поле",
                    true,
                    $_POST[' hidden']);
```

8) текстовое поле для ввода цифр

```
$hiddenint       = new field_hidden("hiddenint",
                    "Скрытое текстовое поле",
                    true,
                    $_POST[' hiddenint']);
```

9) Переключатели

```
$radiobot    = new field_radio(“radiobot”,
                                “Горизонтальный вариант”,
                                Array (“yes”,  “no” ),
                                1,
                                “horizontal” );
$radiobot    = new field_radio(“radiobot”,
                                “Вертикальный вариант”,
                                Array (“yes”,
                                        “no” ),
                                1);
```

10) Выпадающий список

```
$yslovia = new field_select("yslovia",
                            "Выбери раздел статья или новость",
                             array("statia" => "Статья",
                                "news" => "Новости"),
                             $_REQUEST['yslovia']);
```

11) checkbox

```
$hide        = new field_checkbox("hide",
                                  "Отображать",
                                  $_REQUEST['hide']);
```

12) Загрузка файлов на сервер, где последний параметр – папка для хранения изображений

```
$urlpict   = new field_file("urlpict",
                            "Фото",
                             false,
```

```
                    $_FILES,
                    "../../article_photo/");
```

13) Вывод даты

```
$dates      = new field_date ("dates",
                    "Дата новости",
                    $_REQUEST['dates']);
```

14) Вывод даты и времени

```
$datetime      = new field_datetime("datetime",
                    "Дата новости",
                    $_REQUEST['datetime']);
```

После вывода формы на экран, необходимо настроить обработчик пользовательских значений. Сперва нам нужно убедиться, что пользователь нажал кнопку ‘Отправить’. Для этого нам нужно условие if(!empty($_POST)) {}. Внутри этого условия нам нужно проверить корректность заполнения HTML-формы. Для этого воспользуемся методом check() класса form(). Обработчик html-формы помещает все сообщения об ошибках в массив $error, и если имеется хотя бы одно сообщение – выводит сообщения на экран и повторно форму. Под ошибками понимается: пользователь не заполняет обязательное для заполнения поле; пользователь в объект fild_text_email пытается вставить что-то, что не является e-mail-ом; пользователь в объект field_text_int пытается передать не числовое значение и т.д. Для того, чтобы добавить ошибку, нужно создать новый элемент массива $error[], который будет содержать текст выводимой ошибки. Если ошибки есть, то необходимо вывести их на экран с помощью цикла foreach().

Если ошибок нет, то выполнится условие if(!empty($error)) {}, в котором и будет находится основной обработчик вводимых пользователем значений.

Итого у нас получилось 4 действия:

1. Проверяем, нажал ли пользователь кнопку ‘Отправить’
2. Создаем массив – обработчик ошибок.
3. Если ошибки есть выводим их на экран
4. Если ошибок нет, производим обработку пользовательских значений.

Итак, соберем все вместе в следующий листинг:

Обработка пользовательских значений. Листинг 4.6

```
    if(!empty($_POST))
    {
      // Проверяем корректность заполнения HTML-формы
      // и обрабатываем текстовые поля
      $error = $form->check();
    if(empty($error))
    {
    // обработка пользовательских значений, например, вставка в
таблицу базы данных.
    }
    if(!empty($error))
    {
      echo "<br>";
      foreach($error as $err)
      {
   echo "<span style=\"color:red\" class=main_txt>$err</span><br>";
      }
    }
   }
```

Работа с изображениями

Для загрузки изображений воспользуемся функцией resizeimg(), которая находится в файле utils.resizeimg.php

Подключим функцию: require_once ("utils/utils.resizeimg.php");

В функции resizeimg($big, $small, $width, $height) 4 входящих параметра.

$big – это путь к оригинальному изображению.

$small – это имя и путь к уменьшенной копии изображения.

$width и $height – соответственно, ширина и высота уменьшенного изображения.

Функция уменьшает изображение до тех пор, пока один из параметров ширины или высота не станет равным значению $width либо $height.

Итак, создадим объект для загрузки файлов.

Создание элемента формы <input type=file> для загрузки файлов. Листинг 4.7

```
$urlpict   = new field_file("urlpict",
                            "Фото",
                            false,
                            $_FILES,
                            "files/users/");
```

Обработка объекта $urlpict. Листинг 4.8

```
// Изображение
$var = $form->fields['urlpict']->get_filename();
            if(!empty($var))
            {
$picture = "files/users/".date("y_m_d_h_i_").$var;
$picturesmall = "files/users/s_".date("y_m_d_h_i_").$var;
            }
```

Мы создали две переменные $picture и $picturesmall, которые содержат соответственно путь к оригинальному изображению и путь к уменьшенной копии изображения. Чтобы не было одинаковых имен изображений, мы к каждому имени файла добавляем текущие год, месяц, день, час и минуту.

После того, как мы создали переменные, их можно передать в функцию resizeimg().

Использование функции resizeimg. Листинг 4.9

```
      resizeimg($picture, $picturesmall, 250, 200);
```

Регистрация пользователей

Собирая все вместе, создадим форму для регистрации пользователей.

Регистрация пользователей. Листинг 4.10

```
require_once ("templates/top.php");
require_once ("utils/utils.resizeimg.php");

$pol = new field_select("pol",
                        "Выбрать пол",
                        array("М" => "Муж.",    "Ж" => "Жен."),
                  $_REQUEST['pol']);
$familia = new field_text("familia",
                             "Фамилия",
                             true,
                             $_REQUEST['familia']);
$imya = new field_text("imya",
                             "Имя",
                             true,
                             $_REQUEST['imya']);
$otchestvo = new field_text("otchestvo",
                             "Отчество",
                             true,
                             $_REQUEST['otchestvo']);
$familia2 = new field_text("familia2",
                             "Фамилия при рождении",
                             true,
                             $_REQUEST['familia2']);
$mesto = new field_text("mesto",
                             "Место жительства (город)",
                             true,
                             $_REQUEST['mesto']);
$date        = new field_date("date",
```

```
                                  "Дата рождения",
                                  $_REQUEST['date']);
$zanyatie = new field_textarea("zanyatie",
                           "Основное занятие",
                           true,
                           $_REQUEST['zanyatie']);
$email = new field_text("email",
                           "E-mail",
                           true,
                           $_REQUEST['email']);
$name = new field_text("name",
                           "Ник",
                           true,
                           $_REQUEST['name']);
$pass = new field_password("pass",
                               "Пароль",
                               true,
                               $_REQUEST['pass']);
$passagain = new field_password("passagain",
                               "Повтор",
                               true,
                               $_REQUEST['passagain']);
@mkdir("files/users/$_REQUEST[name]/", 0777); //данная строчка нуж-
на для того, чтобы в каталоге files/users/ создать папку с именем
пользователя, и назначить ей уровень доступа 0777. С таким уровнем
доступа пользователи смогут загрузить изображение в созданную пап-
ку.
$urlpict   = new field_file("urlpict",
                            "Фото",
                            false,
                            $_FILES,
                            "files/users/$_REQUEST[name]/");

    $form = new form(array(
                "pol"            => $pol,
                "email"        => $email,
                "name"         => $name,
                "pass"           => $pass,
                "passagain"   => $passagain,
                "familia"       => $familia,
                "imya"          => $imya,
                "otchestvo"   => $otchestvo,
                "familia2"     => $familia2,
                "mesto"         => $mesto,
                "zanyatie"     => $zanyatie,
                "date"            => $date,
                "urlpict"        => $urlpict),
                     "Создать",
                     "main_txt",
                     "",
                     "in_input");
    // Обработчик HTML-формы
    if(!empty($_POST))
    {
      // Проверяем корректность заполнения HTML-формы
      // и обрабатываем текстовые поля
      $error = $form->check();
      // Проверяем равны ли пароли
```

```
      if($form->fields['pass']->value != $form-
>fields['passagain']->value)
      {
        $error[] = "Пароли не равны";
      }
      // Проверяем не зарегистрирован ли пользователь
      // с аналогичным именем ранее
      $query = "SELECT COUNT(*) FROM $tbl_users
                WHERE name = '{$form->fields[name]->value}'";
      $usr = mysql_query($query);
      if(!$usr)
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка добавления нового пользовате-
ля");
      }
      if(mysql_result($usr, 0))
      {
        $error[] = "Пользователь с таким именем уже существует";
      }
        // Изображение
        $var = $form->fields['urlpict']->get_filename();
            if(!empty($var))
            {
$picture = $_REQUEST[name]."/".date("y_m_d_h_i_").$var;
$picturesmall = $_REQUEST[name]."/s_".date("y_m_d_h_i_").$var;
resizeimg($picture, $picturesmall, 250, 200);
            }
      if(empty($error))
      {
        // Формируем SQL-запрос на добавление позиции
        $query = "INSERT INTO $tbl_users
                  VALUES (
                   NULL,
                        '{$form->fields[name]->value}',
                        '{$form->fields[pass]->value}',
                        '{$form->fields[email]->value}',
                        '{$form->fields[pol]->value}',
                        '{$form->fields[familia]->value}',
                        '{$form->fields[imya]->value}',
                        '{$form->fields[otchestvo]->value}',
                        '{$form->fields[familia2]->value}',
                        '{$form->fields[mesto]->value}',
                        '{$form->fields[date]->get_mysql_format()}',
                        '{$form->fields[zanyatie]->value}',
                        '$picture',
                        '$picturesmall',
                        'unblock',
                        NOW(),
                        NOW())";
        if(!mysql_query($query))
        {
          throw new ExceptionMySQL(mysql_error(),
                                   $query,
                                   "Ошибка добавления
                                   нового пользователя");
        }
```

```
        // С помощью javasctipt, осуществляем редирект на страницу,
сообщающую
        // об успешной регистрации
        ?>
        <script>
         document.location.href="index.php";
        </script>
       <?php
        }
    }

    // Выводим сообщения об ошибках если они имеются
    if(!empty($error))
    {
      echo "<br>";
      foreach($error as $err)
      {
   echo "<span style=\"color:red\" class=main_txt>$err</span><br>";
      }
    }
    // Выводим HTML-форму
    $form->print_form();

    //Подключаем нижний шаблон
    require_once ("templates/bottom.php");
```

5. Хранение данных на стороне пользователя.

Сессии (сеансы) в PHP

Сессии и cookies предназначены для хранения сведений о пользователях при переходах между несколькими страницами. При использовании сессий данные сохраняются во временных файлах на сервере. Файлы с cookies хранятся на компьютере пользователя, и по запросу отсылаются броузером серверу.

Использование сессий и cookies очень удобно и оправдано в таких приложениях как Интернет-магазины, форумы, доски объявлений, когда, во-первых, необходимо сохранять информацию о пользователях на протяжении нескольких станиц, а, во-вторых, своевременно предоставлять пользователю новую информацию.

Протокол HTTP является протоколом "без сохранения состояния". Это означает, что данный протокол не имеет встроенного способа сохранения состояния между двумя транзакциями. Т. е., когда пользователь открывает сначала одну страницу сайта, а затем переходит на другую страницу этого же сайта, то основываясь только на средствах, предоставляемых протоколом HTTP невозможно установить, что оба запроса относятся к одному пользователю. Т. о. необходим метод, при помощи которого было бы отслеживать информацию о пользователе в течение одного сеанса связи с Web-сайтов. Одним из таких методов является управление сеансами при помощи предназначенных для этого функций. Для нас важно то, что сеанс по сути, представляет собой

группу переменных, которые, в отличие от обычных переменных, сохраняются и после завершения выполнения PHP-сценария.

При работе с сессиями различают следующие этапы:

- открытие сессии
- регистрация переменных сессии и их использование
- закрытие сессии

Самый простой способ открытия сессии заключается в использовании функции **session_start()**, которая вызывается в начале PHP-сценария:

Синтаксис session_start(). Листинг 5.1

```
// Инициируем сессию
session_start();
// Помещаем значение в сессию
$_SESSION['name'] = "value";
```

Эта функция проверяет, существует ли идентификатор сессии, и, если нет, то создает его. Если идентификатор текущей сессии уже существует, то загружаются зарегистрированные переменные сессии.

Аутентификация пользователей

Функции связанные с пользователями помещены в отдельный файл utils.users.php. В данном случае нас интересует функция enter().

Функция enter(). Листинг 5.2

```
function enter($name, $password)
  {
    // Объявляем название таблицы глобальным
    global $tbl_users;
    // Проверяем, соответствует ли логин паролю
    // и если соответствует - осуществляем авторизацию
    $query = "SELECT * FROM $tbl_users
              WHERE name = '$name' AND
                    pass = '$password' AND
                    block = 'unblock'
              LIMIT 1";
    $usr = mysql_query($query);
    if(!$usr)
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                               "Ошибка аутентификации");
    }
    if(mysql_num_rows($usr))
    {
      $user = mysql_fetch_array($usr);
      // Вход на сайт
     $_SESSION['id_user_position'] = $user['id_position'];
      // Обновляем дату последнего посещения пользователя
```

```
      $query = "UPDATE $tbl_users
                SET lastvisit = NOW()
                WHERE id_position = $user[id_position]";
      if(!mysql_query($query))
      {
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                 "Ошибка аутентификации");
      }
      // Возвращаем признак успешной аутентификации
      return true;
    }
    // Возвращаем признак неудачной аутентификации
    else return false;
  }
```

Функция enter() принимает два входящих параметра: имя и пароль. После чего в SELECT-запросе идет их проверка, и если такая пара логин-пароль есть в таблице $tbl_users, мы инициализируем переменную сессии $_SESSION[‘id_user_position’], и обновляем в таблице время последнего посещения.

Вход на сайт. Листинг 5.3

```
    require_once ("templates/top.php");
    require_once ("utils/utils.users.php");
    // Если пользователь уже авторизован - инициируем
    // элементы HTML-формы
    // Формируем HTML-форму
    $name = new field_text("name",
                           "Ник",
                           true,
                           $_REQUEST['name']);
    $pass = new field_password("pass",
                               "Пароль",
                               true,
                               $_REQUEST['pass']);
    $form = new form(array("name"         => $name,
                                             "pass"         =>
$pass),
                     "Войти",
                     "main_txt",
                     "",
                     "in_input");
    // Обработчик HTML-формы
    if(!empty($_POST))
    {
      // Проверяем корректность заполнения HTML-формы
      // и обрабатываем текстовые поля
      $error = $form->check();
      // Проверяем имеется ли в базе данных пользователь
      // с указанным именем
      $query = "SELECT COUNT(*) FROM $tbl_users
                WHERE name = '{$form->fields[name]->value}'";
      $usr = mysql_query($query);
      if(!$usr)
      {
```

```
        throw new ExceptionMySQL(mysql_error(),
                                 $query,
                                "Ошибка извлечения параметров
                                 пользователя");
      }
      if(!mysql_result($usr, 0))
      {
        $error[] = "Пользователь с таким именем не существует";
      }
      if(empty($error))
      {
        // Проверяем соответствует ли логин паролю
         enter($form->fields['name']->value,
                  $form->fields['pass']->value);
        // Перегружаем страницу
<script>
  document.location.href="index.php";
</script>
      }
    }
    // Выводим сообщения об ошибках если они имеются
    if(!empty($error))
    {
      echo "<br>";
      foreach($error as $err)
      {
        echo "<span style=\"color:red\"
class=main_txt>$err</span><br>";
      }
    }
    // Выводим HTML-форму
    $form->print_form();
    //Подключаем нижний шаблон
    require_once ("templates/bottom.php");
```

Данный листинг создает два элемента формы для ввода логина и пароля и кнопку «Войти». Затем вводимые пользователем значения попадают в функцию enter(). После того, как все проверки пройдены и запросы выполнены, редиректимся на индексную страницу.

Далее, мы можем использовать разделение уровня доступа:

Разделение уровня доступа. Листинг 5.4

```
If ($_SESSION['id_user_position'])
{
//для авторизированных пользователей
}
else
{
// для неавторизированных пользователей
}
```

Реализация кнопки выход

Для реализации выхода пользователей необходимо убить сессию.

Создаем файл, например, с названием register_out.php, где пишем следующее:

Страница выхода. Листинг 5.5

```
<?php
session_start();
session_destroy();
?>
<script>
  document.location.href="index.php";
</script>
```

Это означает, что сеанс сессий на данном сайте у пользователя завершен.

Вспомнить пароль

Для разработки модуля «вспомнить пароль» можно воспользоваться готовой функцией remember из файла utils.users.php, находящегося в корневой директории.

Вспомнить пароль. Листинг 5.6

```
  function remember($name)
  {
    // Объявляем имя таблицы $tbl_users глобальным
    global $tbl_users;
    // Формируем SQL-запрос на извлечение пользовательских данных
    $query = "SELECT * FROM $tbl_users
              WHERE name = '$name'";
    $usr = mysql_query($query);
    if(!$usr)
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                               "Ошибка при восстановлении пароля");
    }
    // Извлекаем e-mail пользователя
    $user = mysql_fetch_array($usr);

    $thm =  convert_cyr_string("Восстановление пароля",'w','k');
    $msg = "Восстановление пароля на сайте\r\n".
           "Логин - $user[name] \r\n".
           "Пароль - $user[pass] \r\n";
    $msg =  convert_cyr_string(stripslashes($msg),'w','k');
    $header = "Content-Type: text/plain; charset=KOI8-R\r\n\r\n";
    // Если на странице администрирования указан
    // адрес отсылки сообщения - отправляем письмо
    @mail($user['email'], $thm, $msg, $header);
  }
```

Единственным входящим данным в функцию является логин. По логину мы можем определить e-mail пользователя и пароль, и отправить запрос на этот e-mail.

6. Постраничная навигация.

Для реализации постраничной навигации воспользуемся классом pager_mysql (из уже знакомого нам фрэймворка)

Постраничная навигация. Листинг 6.1

```
   // Количество ссылок в постраничной навигации
    $page_link = 3;
    // Количество позиций на странице
    $pnumber = 10;
    // Объявляем объект постраничной навигации
    $obj = new pager_mysql($tbl_news,
                           "",
                           "ORDER BY id_news DESC",
                           $pnumber,
                           $page_link);
    // Получаем содержимое текущей страницы
    $news = $obj->get_page();
    // Если имеется хотя бы одна запись - выводим
    if(!empty($news))
    {
      for($i = 0; $i < count($news); $i++)
      {
echo $news[$i]['name']
      }
    }

    // Выводим ссылки на другие страницы
    echo $obj;
```

В классе pager_mysql 5 входящих параметров:

1. Имя таблицы из базы
2. Условие выборки, если нужно выбрать все записи, то значение оставляем пустым.
3. Сортировка и ограничение записей.
4. Количество записей в постраничной навигации на одной странице.
5. Количество ссылок слева и справа от текущей нажатой.

Необычность данного класса постраничной навигации заключается в том, что нам нужно разбивать SELECT запрос на три части (первые три параметра класса).

Далее по листингу: если есть хотя бы одна запись, включаем цикл for. Количество оборотов цикла будет соответствовать количеству строк в таблице базы данных: $i < count($news), где $news – массив, содержащий все выбранные записи из базы, а функция count подсчитывает общее количество элементов массива.

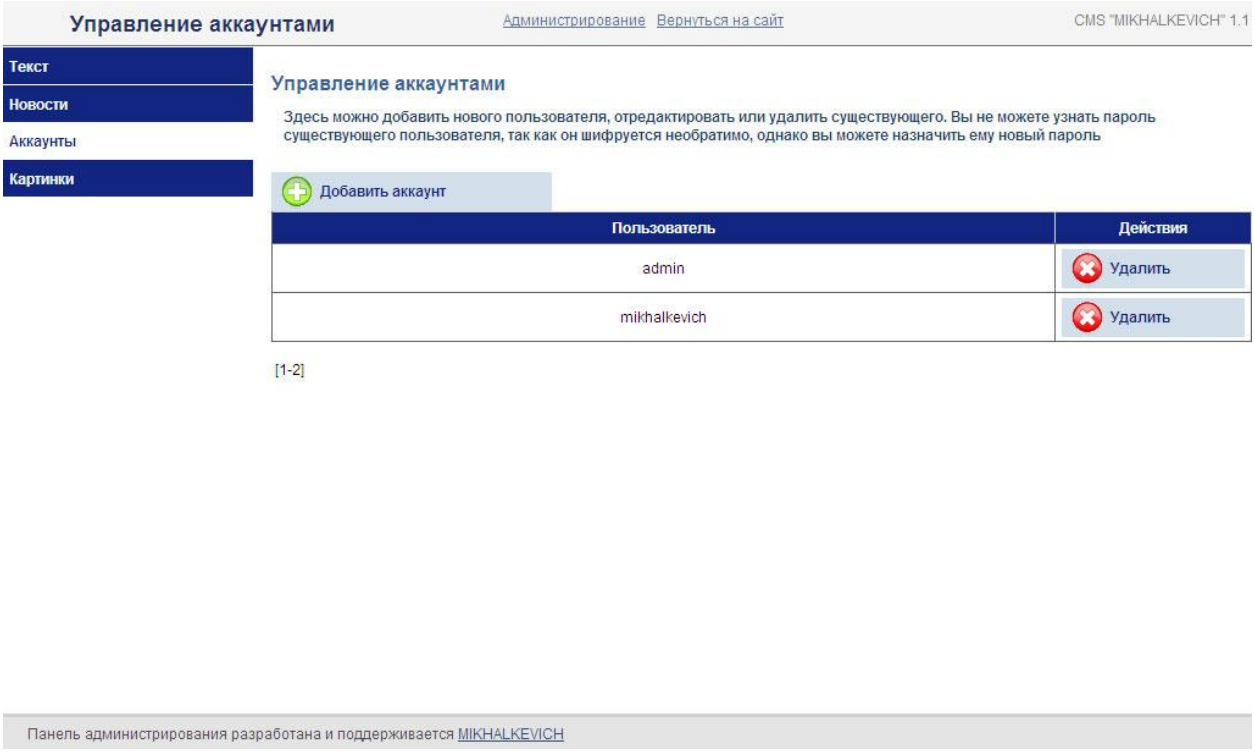
Внутри цикла мы получим сложный ассоциативный числовой массив, где ассоциацией является имя ячейки в таблице базы данных.

$news[$i][‘name’] – где $i – строка таблицы, name – имя ячейки.

7. Система администрирования содержимого сайта, CMS. Редактор кода.

Система администрирования

Так выглядит один из блоков системы администрирования.



Визуально, страницу системы администрирования можно разбить на следующие компоненты: шапка, подложка, блок меню и основная часть. Шапка и подложка подключается, как обычные шаблоны.

На данном этапе нас интересует подключаемый блок меню

Подключаем меню:

Файл menu.php. Листинг 7.1

```
// Открываем каталог /adminka
 $dir = opendir("..");
 // В цикле проходимся по всем файлам и
 // подкаталогам
 while (($file = readdir($dir)) !== false)
 {
   // Обрабатываем только подкаталоги,
```

```
    // игнорируя файлы
    if(is_dir("../$file"))
    {
      // Исключаем текущий ".", родительский ".."
      // каталоги, а также каталог utils
      if($file != "." && $file != ".." && $file != "utils")
      {
        // Ищем в каталоге файл с описанием
        // блока .htdir
        if(file_exists("../$file/.htdir"))
        {
          // Файл .htdir существует -
          // читаем название блока и его
          // описание
          list($block_name,
               $block_description) = file("../$file/.htdir");
        }
        else
        {
          // Файл .htdir не существует -
          // в качестве его названия
          // выступает имя подкаталога
          $block_name        = "$file";
          $block_description = "";
        }

        // Отмечаем текущий пункт другим стилем
        if(strpos($_SERVER['PHP_SELF'], $file) !== false)
        {
          $style = 'class=active';
        }
        else $style = '';

        // Формируем пункт меню
        echo "<div $style>
                <a class=menu
                   href='../$file'
                   title='$block_description'>
                   $block_name
                </a>
              </div>";
      }
    }
  }
  // Закрываем каталог
  closedir($dir);
```

Пояснения: открываем текущий каталок (т.е. папку adminka), проходимся по всем вложенным папкам, за исключением папки utils. Ищем файл с описанием раздела управления .htdir и выводим на экран первую строчку, содержащуюся в данном файле, и делаем ее ссылкой на индексный файл раздела управления. Вторая строчка вставляется в атрибут title. Если файл .htdir отсутствует, то выводим имя каталога. Таким образом, мы получим столько ссылок в блоке меню, сколько у нас есть каталогов. Каждый отдельный каталог – это один блок системы администрирования.

Далее, нам необходимо защитить от посторонних глаз блок администрирования. Для этого к тем файлам, которые необходимо запаролить, подключаем модуль авторизации:

require_once("../utils/security_mod.php");

файл security_mod.php . Листинг 7.2

```
// Если пользователь не авторизовался - авторизуемся
  if(!isset($_SERVER['PHP_AUTH_USER']) || (!empty($_GET['logout'])
&& $_SERVER['PHP_AUTH_USER'] == $_GET['logout']))
  {
    Header("WWW-Authenticate: Basic realm=\"Control Page\"");
    Header("HTTP/1.0 401 Unauthorized");
    exit();
  }
  else
  {
    // Утюжим переменные $_SERVER['PHP_AUTH_USER'] и
$_SERVER['PHP_AUTH_PW'],
    // чтобы мышь не проскочила
    if (!get_magic_quotes_gpc())
    {
      $_SERVER['PHP_AUTH_USER'] =
mysql_escape_string($_SERVER['PHP_AUTH_USER']);
      $_SERVER['PHP_AUTH_PW']   =
mysql_escape_string($_SERVER['PHP_AUTH_PW']);
    }

    $query = "SELECT * FROM $tbl_accounts
              WHERE name = '".$_SERVER['PHP_AUTH_USER']."'";
    $lst = @mysql_query($query);
    // Если ошибка в SQL-запросе - выдаём окно
    if(!$lst)
    {
      Header("WWW-Authenticate: Basic realm=\"Control Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
    // Если такого пользователя нет - выдаём окно
    if(mysql_num_rows($lst) == 0)
    {
      Header("WWW-Authenticate: Basic realm=\"Control Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
    // Если все проверки пройдены, сравниваем хэши паролей
    $account = @mysql_fetch_array($lst);
    if(md5($_SERVER['PHP_AUTH_PW']) != $account['pass'])
    {
      Header("WWW-Authenticate: Basic realm=\"Control Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
  }
```

Сперва мы проверяем на существование переменные $_SERVER['PHP_AUTH_USER'] и $_SERVER['PHP_AUTH_PW']. Если они существуют, значит, пользователь уже аутентифицировался с правами администратора. Если же таких переменных не существует, выводим панель для ввода логина и пароля. Далее по скрипту, мы делаеа запрос в базу данных, в таблицу $tbl_accaunt. Сверяем имя, вводимое пользователем с существующим в таблице. Если такого пользователя нет – выводим окно. Если такое имя есть, сравниваем хэши паролей. Вводимый пользователем пароль необходимо пропустить через функцию md5(), т.к. в базе мы храним пароли в зашифрованном виде, т.е. храним не сам пароль, а хэш пароля.

Далее рассмотрим индексный файл новостного блока системы администрирования:

Индексный файл админки. Листинг 7.3

```
<?php
 error_reporting(E_ALL & ~E_NOTICE);
 // Устанавливаем соединение с базой данных
 require_once("../../config/config.php");
 // Подключаем блок авторизации
 require_once("../utils/security_mod.php");
 // Подключаем SoftTime FrameWork
 require_once("../../config/class.config.dmn.php");
 // Подключаем блок отображения текста в окне браузера
 require_once("../utils/top.php");
   // Количество ссылок в постраничной навигации
   $page_link = 3;
   // Количество позиций на странице
   $pnumber = 10;
   // Объявляем объект постраничной навигации
   $obj = new pager_mysql($tbl_news,
                          "",
                          "ORDER BY id_news DESC",
                          $pnumber,
                          $page_link);
   // Получаем содержимое текущей страницы
   $news = $obj->get_page();
   // Если имеется хотя бы одна запись - выводим
   if(!empty($news))
   {
     ?>
     <table width="100%"
            class="table"
            border="0"
            cellpadding="0"
            cellspacing="0">
       <tr class="header" align="center">
         <td width=200>Дата</td>
         <td width=60%>Новость</td>
         <td width=40>Избр-е</td>
         <td>Действия</td>
       </tr>
     <?php
     for($i = 0; $i < count($news); $i++)
```

```
      {
        // Если новость отмечена как невидимая (hide='hide'), выво-
дим
        // ссылку "отобразить", если как видимая (hide='show') -
"скрыть"
        $colorrow = "";
        $url = "?id_news={$news[$i][id_news]}&page=$page";
        if($news[$i]['hide'] == 'show')
        {
          $showhide = "<a href=newshide.php$url
                           title='Скрыть новость в блоке новостей'>
                       <img src='../utils/img/folder_locked.gif'
border=0 align='absmiddle' />Скрыть</a>";
        }
        else
        {
          $showhide = "<a href=newsshow.php$url
                           title='Отобразить новость в блоке ново-
стей'>
        <img src='../utils/img/show.gif' border=0 align='absmiddle'
/>Отобразить</a>";
          $colorrow = "class='hiddenrow'";
        }
        // Проверяем наличие изображения
        if($news[$i]['urlpict'] != '' &&
           $news[$i]['urlpict'] != '-' &&
           is_file("../../".$news[$i]['urlpict']))
        {
        $url_pict = "<b><a
href=../../{$news[$i][urlpict]}>есть</a></b>";
        }
        else $url_pict = "нет";
        // Выводим новость
        echo "<tr $colorrow >
                <td><p align=center>{$news[$i][putdate]}</td>
                <td>
                  <a title='Редактировать текст новости'

href=newsedit.php$url>{$news[$i][name]}</a><br>
                  ".nl2br(print_page($news[$i]['body']))." $news_url
</td>
                <td align=center>$url_pict</td>
                <td class='menu_right' valign='top'
align=left>$showhide
                     <a href=# on-
Click=\"delete_position('newsdel.php$url',".
                                          "'Вы действительно хотите
удалить".
                                          " новостное сообщение?');\"
                       title='Удалить новость'>
                    <img border=0 src='../utils/img/editdelete.gif'
align='absmiddle' />Удалить</a>
                     <a href=newsedit.php$url
                title='Редактировать текст новости'>
               <img src='../utils/img/kedit.gif' border=0
align='absmiddle' />Редактировать текст</a>
              </td>
             </tr>";
```

```
        }
        echo "</table><br>";
      }

      // Выводим ссылки на другие страницы
      echo $obj;
    // Включаем завершение страницы
    require_once("../utils/bottom.php");
?>
```

Как видно из листинга, напротив каждой записи мы вывели ссылки на редактирование, удаление и скрытие либо отображение новости.

Создание переменной для передачи get-параметров. Листинг 7.4

```
$url = "?id_news={$news[$i][id_news]}&page=$_GET['page']";
```

Для того, чтобы передать id_news на другие страницы (для редактирования, скрытия либо удаления записи), нам достаточно будет к имени файла добавить переменную $url. Где $_GET[‘page’] – это текущая страница в постраничной навигации.

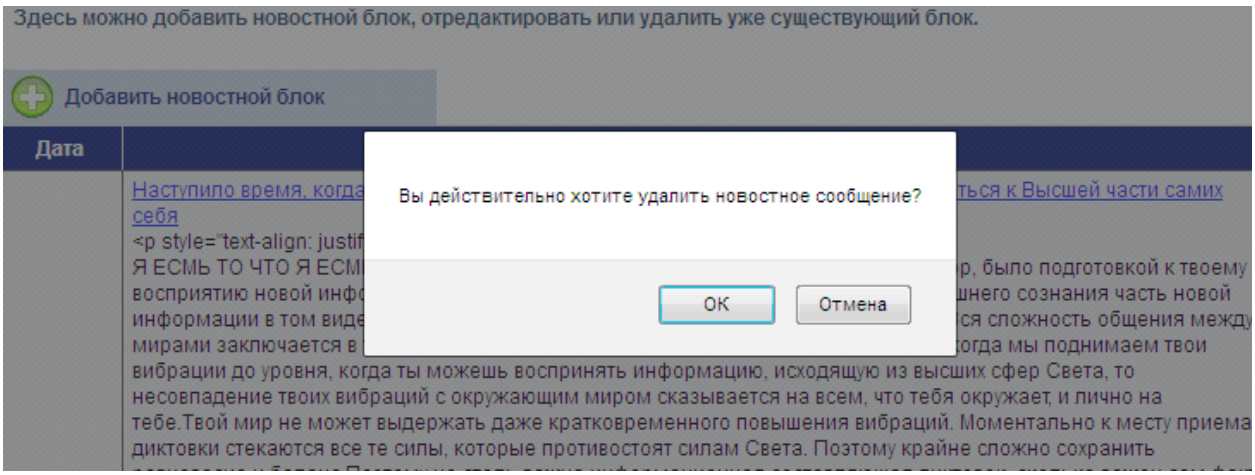
Удаление

Формирование ссылки на удаление. Листинг 7.5

```
Echo "<a href=# onClick=\"delete_position('newsdel.php$url',". "'Вы
действительно хотите удалить новостное сообщение?');\"
 title='Удалить новость'>Удалить</a>";
```

Из листинга видно, что мы на параметр href повесили заглушку. При нажатии на ссылку будет срабатывать функция javasctipt onClick, будет подгружаться функция delete_position() с двумя входящими параметрами: 1) адрес страницы удаления и 2) текстовое сообщение.

Перед удалением записи нам необходимо спросить пользователя, действительно ли он хочет удалить запись и предупредить его, что он совершает необратимое действие.



Задача функции delete_position – вывести данное окно подтверждения.

Функция delete_position. Листинг 7.6

```
<script type='text/javascript'>
<!--
  function delete_position(url, ask)
  {
    if(confirm(ask))
    {
      location.href=url;
    }
    return false;
  }
</sctipt>
```

Пояснения: метод confirm() выводит окно подтверждения с вопросом ask. Если пользователь нажимает OK, осуществляется переход на страницу url.

А вот и сам файл newsdel.php

Удаление записи. Листинг 7.7

```
  // Устанавливаем соединение с базой данных
  require_once("../../config/config.php");
  // Проверяем параметр id_news, предотвращая SQL-инъекцию
  $_GET['id_news'] = intval($_GET['id_news']);
    // Если новостное сообщение содержит
    // изображение - удаляем его
    $query = "SELECT * FROM $tbl_news
              WHERE id_news=$_GET[id_news]";
    $new = mysql_query($query);
    if(!$new)
    {
      throw new ExceptionMySQL(mysql_error(),
                               $query,
                              "Ошибка удаления
                               новостного блока");
    }
    if(mysql_num_rows($new) > 0)
    {
      $news = mysql_fetch_array($new);
      if(file_exists("../../".$news['urlpict']))
      {
        @unlink("../../".$news['urlpict']);
      }
    }
    // Формируем и выполянем SQL-запрос
    // на удаление новостного блока из базы данных
    $query = "DELETE FROM $tbl_news
              WHERE id_news=$_GET[id_news]
              LIMIT 1";
    if(mysql_query($query))
    {
      header("Location: index.php?page=$_GET[page]");
    }
    else
    {
```

```
     Exit("Ошибка удаления");
    }
```

Сперва, мы делаем SELECT-запрос. Затем функцией file_exists() проверяем, существует ли изображение, если есть, удаляем функцией @unlink. И только затем удаляем саму запись. Если в таблице полей для вставки изображений нет, то SELECT-запрос не нужен. Функцию intval() мы используем для того, чтобы убедиться в том что $_GET['id_news'] число.

После того, как DELETE-запрос выполнится, включается функция header() и параметр location перенаправляет нас на страницу index.php с тем $_GET['page'], который был переменной $url на индексной странице.

Редактирование

Под редактированием записей можно понимать тот же процесс добавления, только со вставленными в форму значениями из таблицы базы данных. Структурно, редактирование можно разбить на четыре этапа: 1) SELECT-запрос, 2) создание формы 3) вставка табличных значений в форму 4) добавление значений из формы в базу.

SELECT-запрос для редактирвания. Листинг 7.8

```
$query = "SELECT * FROM $tbl_news
          WHERE id_news=$_GET[id_news]";
$new = mysql_query($query);
if(!$new)
{
  throw new ExceptionMySQL(mysql_error(),
                           $query,
                          "Ошибка при обращении
                           к таблице новостей");
}
$news = mysql_fetch_array($new);
```

Таким образом, мы получим ассоциативный массив $news, где ассоциацией будет выступать имя столбца таблицы.

А теперь вспомним о суперглобальном массиве $_REQUEST, в который нам нужно передать информацию из базы данных.

Передача информации в суперглобальный массив $_REQUEST. Листинг 7.9

```
    $_REQUEST = $news;
```

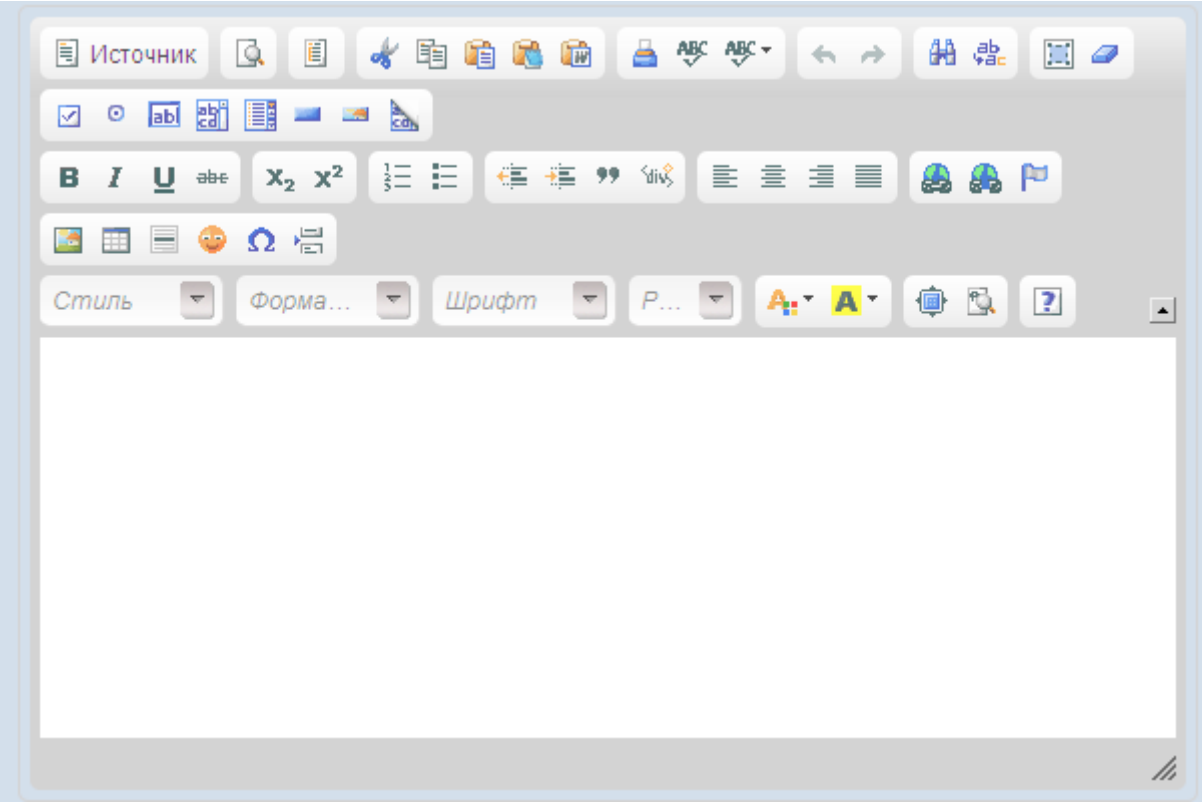
Дальнейший код аналогичен коду добавления записей (Занятие 3), за исключением того, что мы используем не INSERT-запрос, а UPDATE

UPDATE-запрос. Листинг 7.10

```
                $query = "UPDATE $tbl_news
                                  SET name = '{$form->fields['name']-
>value}',
                                  body = '{$form->fields['editor1']-
>value}',
                                  putdate = '{$form->fields['date']-
>get_mysql_format()}',
                                 $url_pict
                                 hide = '{$showhide}'
                                 WHERE id_news=".$_GET['id_news'];
          if(!mysql_query($query))
          {
            throw new ExceptionMySQL(mysql_error(),
                                      $query,
                                     "Ошибка при редактировании
                                      новостного сообщения");
          }
```

Редактор кода

Можно усовершенствовать объекты класса fild_textarea, добавив в него редактор кода.



1) В корень сайта добавим папки ckeditor и ckfinder.
2) На странице с редактором кода мы подключим внешние файлы скрипта.

Подключение скрипта редактора кода. Листинг 7.11

```
    <script type="text/javascript"
src="../../ckeditor/ckeditor.js"></script>
    <script type="text/javascript"
src="../../ckfinder/ckfinder.js"></script>
```

После вывода формы на экран, необходимо подключить на странице следующий код:

Подключение скрипта редактора кода. Листинг 7.12

```
<script type="text/javascript">
// This is a check for the CKEditor class. If not defined, the
paths must be checked.
if ( typeof CKEDITOR == 'undefined' )
{
    document.write(
        '<strong><span style="color: #ff0000">Error</span>: CKEdi-
tor not found</strong>.' +
        'This sample assumes that CKEditor (not included with
CKFinder) is installed in' +
        'the "/ckeditor/" path. If you have it installed in a dif-
ferent place, just edit' +
        'this file, changing the wrong paths in the &lt;head&gt;
(line 5) and the "BasePath"' +
        'value (line 32).' ) ;
}
else
{
    var editor = CKEDITOR.replace( 'editor1' );
    //editor.setData( '<p>Just click the <b>Image</b> or
<b>Link</b> button, and then <b>&quot;Browse Server&quot;</b>.</p>'
);

    // Just call CKFinder.setupCKEditor and pass the CKEditor in-
stance as the first argument.
    // The second parameter (optional), is the path for the CKFind-
er installation (default = "/ckfinder/").
    CKFinder.setupCKEditor( editor, '../../ckfinder/' ) ;

    // It is also possible to pass an object with selected CKFinder
properties as a second argument.
    // CKFinder.setupCKEditor( editor, { basePath : '../', skin :
'v1' } ) ;
}
        </script>
```

Т.о. на данной странице обычный textarea превратится в редактор кода.

8. Библиотека jQuery.

Существует три способа подключения библиотеки jQuery:

Первый способ: подключение локального файла библиотеки. Листинг 8.1

```
<script type=”text/javascript” src=”js/jquery-1.4.2.min.js”>
</script>
```

Библиотека подключается, как любой другой js-файл

Второй способ: подключение библиотеки, хранящейся на удаленном хсте google. Листинг 8.2

```
<script type=”text/javascript”
src=”http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js”>
</script>
```

Еще один способ – подключение библиотеки, хранящейся на сайте GOOGLE Code. Это делается в надежде на то, что к моменту посещения вашего сайта пользователи уже будут иметь копию библиотеки, кэшированную с другого сайта, который в свое время загрузил эту библиотеку с GOOGLE Code, что обеспечит сокращение времени загрузки страниц для пользователей нашего сайта.

Третий способ: использование Google Libraries API. Листинг 8.3

```
<script type=”text/javascript” src=”http://www.google.com/jsapi” >
</script>
<script type=”text/javascript” >
      google.load(“jquery”, “1.4.2”);
</script>
```

На сайте GOOGLE Code предлагается еще один вариант загрузки – с помощью Google Libraries API. Данный способ необходимо использовать, если мы планируем подключать помимо jQuery другие библиотеки.

Вся работа с jQuery ведётся с помощью функции $. Если на сайте применяются другие javascript библиотеки, где $ может использоваться для своих нужд, то можно использовать её синоним — jQuery. Второй способ считается более правильным, а чтобы код не получался слишком громоздким пишут его следующим образом:

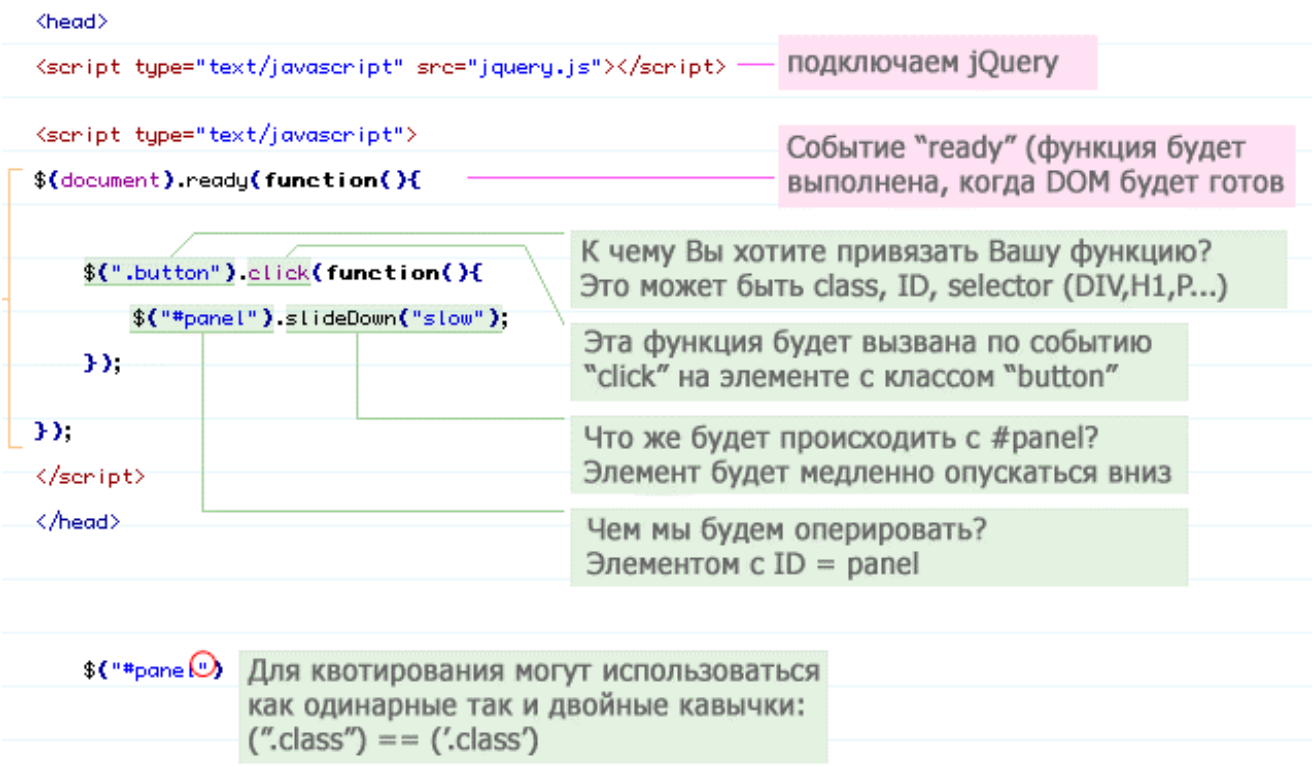
Весь jQuery код внутри данной функции. Листинг 8.4

```
jQuery(function($) {
  // Тут код скрипта, где в $ будет jQuery
})
```

Три этапа работы библиотеки jQuery: 1) выбор селектора 2) подключение обработчика событий 3) вызов функции.

Пример работы jQuery. Листинг 8.5

```
  $(".button").click(function(){
    $("#panel").slideDown("slow");
  });
```



Основа работы jQuery – это умение обращаться к любому тегу на странице, т.е. знание селекторов. Селекторы jQuery особенно просто освоить тем, кто уже знаком с CSS, поскольку им придется иметь дело с тем же синтаксисом.

Селекторы

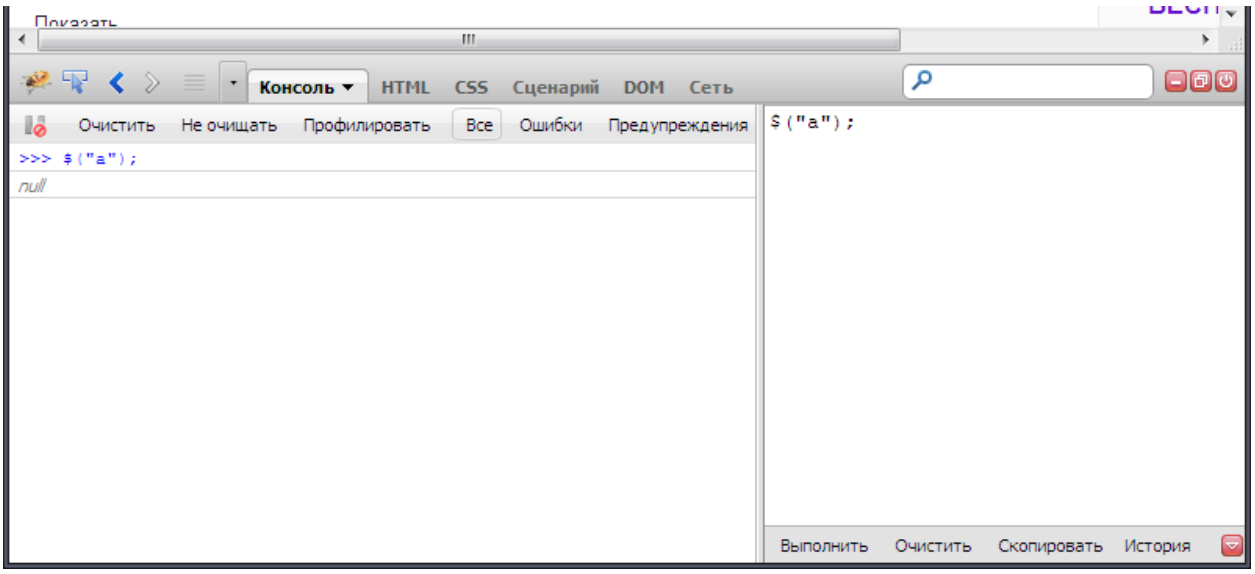
- *$("p"), $("a"), $("div")* - выбор элементов по типу тега.
- *$(".foo")* – выбор элементов по имени класса.
- *$("#bar")* – выбор элементов по идентификатору.
- *$("p.foo")* – комбинированные селекторы.
- *$("p.foo, #bar")* – групповые селекторы. Используются, когда необходимо получить доступ сразу к нескольким элементам. Команда возвратит элементы, соответствующие хотя бы одному из указанных селекторов.
- *$("#bar span")* – выбор элементов потомков. Вложенность может быть любой.
- *$("#bar>span")* – выбор дочерних элементов. Данный селектор отбирает лишь те элементы, которые являются непосредственными потомками элементов-родителей.
- *$(".foo+p")* – выбор следующего элемента.
- *$(".foo~p")* - выбор сестринского элемента. Отбор сестринских элементов осуществляется точно так же, как и отбор следующих элементов, за исключением того, что данный селектор будет отыскивать все

сестринские элементы, располагающиеся после начального, а не только следующий.

Фильтры

- *$("p:first"), $("p:last")* – выбор first – первого, last – последнего элемента
- *$("p:not(.foo)")* – выбор элементов, не соответствующих селектору.
- *$("p:odd"), $("p:even")* – выбор элементов по признаку четности. Odd – выбирает нечетные элементы. Even - четные.
- *$("p:eq(3)")* – выбор элементов по индексу. В качестве параметра передается индекс требуемого значения. Индекс первого элемента – 0.
- *$("p:contains(какой-то текст)")* – выбор элементов, содержащих определенный текст. Фильтр чувствителен к регистру.
- *$("p:has(span)")* – выбор элементов, содержащих указанный элемент.
- *$(":empty")* –выбор пустых элементов
- *$("p:parent")* – выбор родительских элементов.
- *$(":hidden"), $(":visible")* – выбор соответственно скрытых и видимых элементов.
- *$("[src=img/pic1.jpg]")* – выбор элементов по значению атрибута.
- *$("[src!=img/pic1.jpg]")* – выбор элементов, не имеющих заданного атрибута, или имеющих другое значение.
- *$(":button"), $(":checkbox"), $(":input"), $(":radio")* и т.д. – выбор соответствий по типу форм.
- *$(":disabled"), $(":enabled")* – выбор включенных и отключенных элементов форм.
- *$(":selected"), $(":checked")* – выбор выделенных или отмеченных элементов формы.

Познакомившись с селекторами и фильтрами jQuery, и подключив саму библиотеку, можно протестировать работу библиотеки. Для этого воспользуемся плагином firebug для firefox. Включим консоль (F12), и впишем какой-нибудь существующий на странице тег.



Если jQuery не подключился – мы увидим на экране левого окна null.

Если же с библиотекой все в порядке, мы увидим список всех выбранных элементов.

Методы live() и die()

Задача метода live() – связать обработчик событий с функцией jQuery.

Метод Live(). Листинг 8.6

```
$(".selector").live("click", function() {
console.log("Событие – щелчок по ссылке");
});
```

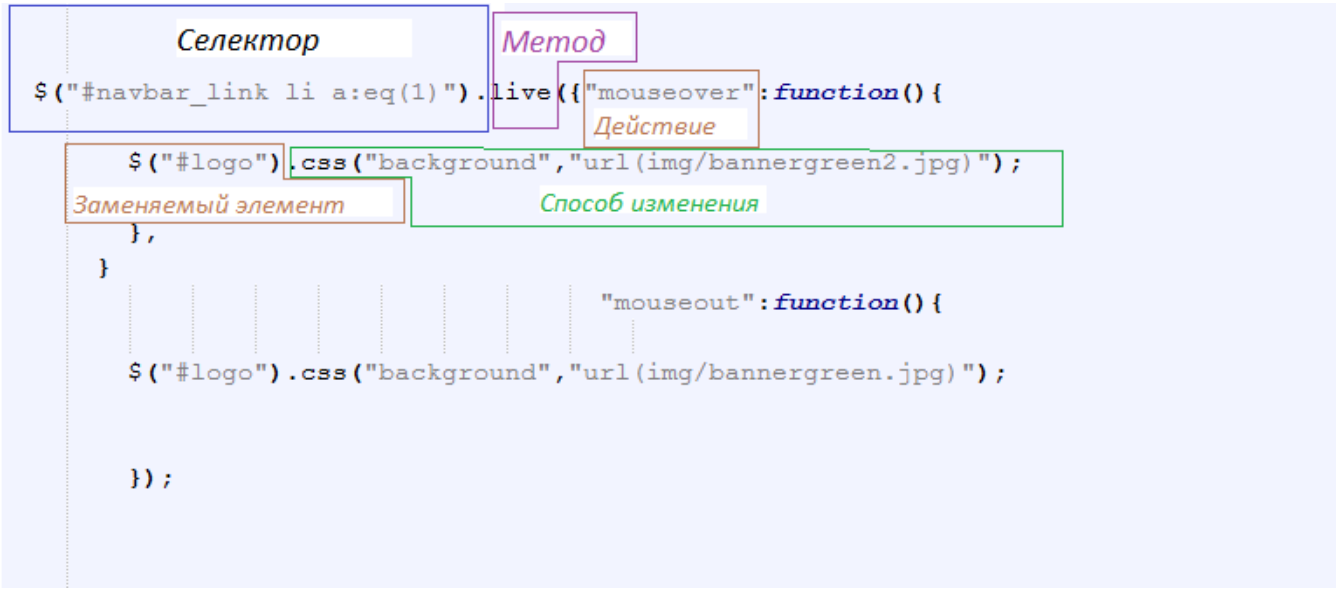
Таким образом, мы связали селектор a с функцией через событие click

Чтобы привязать к одному селектору разные обработчики (например, click и mouseover), можно использовать следующий код

Связка нескольких обработчиков с одним селектором. Листинг 8.7

```
$("p").live({
      "click": function(){функция},
      "mouseover":function(){функция}
});
```

Для наглядности работу метода live рассмотрим на изображении:



Для того чтобы удалить все события от абзацев, используем следующий код:

Отключение всех событий. Листинг 8.8

```
$("p").die();
```

Отключение конкретного события. Листинг 8.9

```
$("p").die("click");
```

Метод one()

Назначение и способ использования метода one() и live() совпадают. За исключением того, что метод one() отключает обработчик событий после того, как событие успеет произойти один раз.

События

blur( )
Вызывает событие blur для каждого элемента набора.
blur(функция)
Назначает функцию к событию blur для каждого элемента набора.
change( )
Вызывает событие change для каждого элемента набора.
change(функция)
Назначает функцию к событию change для каждого элемента набора.
click( )
Вызывает событие click для каждого элемента набора.
click(функция)
Назначает функцию к событию click для каждого элемента набора.
dblclick( )
Вызывает событие dblclick для каждого элемента набора.
dblclick(функция)

Назначает функцию к событию dblclick для каждого элемента набора.

error( )

Вызывает событие error для каждого элемента набора.

error(функция)

Назначает функцию к событию error для каждого элемента набора.

focus( )

Вызывает событие focus для каждого элемента набора.

focus(функция)

Назначает функцию к событию focus для каждого элемента набора.

keydown( )

Вызывает событие keydown для каждого элемента набора.

keydown(функция)

Назначает функцию к событию keydown для каждого элемента набора.

keypress( )

Вызывает событие keypress для каждого элемента набора.

keypress(функция)

Назначает функцию к событию keypress для каждого элемента набора.

keyup( )

Вызывает событие keyup для каждого элемента набора.

keyup(функция)

Назначает функцию к событию keyup для каждого элемента набора.

load(функция)

Назначает функцию к событию load для каждого элемента набора.

mousedown(функция)

Назначает функцию к событию mousedown для каждого элемента набора.

mouseenter(функция)

Назначает функцию к событию mouseenter для каждого элемента набора.

mouseleave(функция)

Назначает функцию к событию mouseleave для каждого элемента набора.

mousemove(функция)

Назначает функцию к событию mousemove для каждого элемента набора.

mouseout(функция)

Назначает функцию к событию mouseout для каждого элемента набора.

mouseover(функция)

Назначает функцию к событию mouseover для каждого элемента набора.

mouseup(функция)

Назначает функцию к событию mouseup для каждого элемента набора.

resize(функция)

Назначает функцию к событию resize для каждого элемента набора.

scroll(функция)

Назначает функцию к событию scroll для каждого элемента набора.

select( )

Вызывает событие select для каждого элемента набора.

select(функция)

Назначает функцию к событию select для каждого элемента набора.

submit( )

Вызывает событие submit для каждого элемента набора.

submit(функция)

Назначает функцию к событию submit для каждого элемента набора.

unload(функция)

Назначает функцию к событию unload для каждого элемента набора.

Методы
Attr

attr(имя)

Получение доступа к свойству первого совпавшего элемента.
Используя этот метод можно легко получить значение свойства первого совпавшего элемента. Если элемент не имеет указанного атрибута, то возвращается undefined (не определено). Атрибутами могут быть: title, alt, src, href, width, style и т.д.

attr(свойства)

Устанавливает атрибуты всех элементов набора, используя при этом объект, который содержит пары ключ/значение.

attr(ключ, значение)

Изменяет значение единственного свойства для каждого совпавшего элемента.

attr(ключ, функция)

Изменяет значение единственного свойства для каждого совпавшего элемента.
Но вместо указания непосредственно значения указывается функция, которая возвращает значение как свой результат.

removeAttr(имя)

Удаляет указанный атрибут из каждого совпавшего элемента.

Класс

addClass(класс)

Добавляет указанный(е) класс(ы) к каждому совпавшему элементу.

hasClass(класс)

Возвращает true, если хотя бы один из набора совпавших элементов обладает

указанным классом.

removeClass(класс)

Удаляет все или указанный(е) класс(ы) из набора совпавших элементов.

toggleClass(класс)

Добавляет указанный класс к элементу, если его нет, и удаляет указанный класс, если элемент уже обладает таковым.

toggleClass(класс, переключатель)

Добавляет указанный класс, если переключатель установлен в true, и удаляет указанный класс, если переключатель установлен в false.

HTML

html( )

Получает содержимое HTML (innerHTML) каждого совпавшего элемента. С документами XML использовать данный метод нельзя, но можно использовать с документами XHTML.

html(val)

Добавляет код HTML для каждого совпавшего элемента. С документами XML использовать данный метод нельзя, но можно использовать с документами XHTML.

Текст:

text( )

Получает содержимое всех совпавших элементов.

text(val)

Вставка текста во все совпавшие элементы.

Значение:

val( )

Получает содержимое атрибута value для первого элемента ввода в наборе.

val(val)

Устанавливает значение атрибута value для каждого совпавшего элемента в наборе.

val(val)

Может отмечать значение или делать выбор в таких элементах, как radio, checkbox и select.

CSS

css(имя) Возвращает свойство стиля для первого совпавшего элемента.

css(свойства) Устанавливает свойства стиля css всех элементов набора,

используя при этом объект, который содержит пары ключ/значение.
css(имя, значение) Устанавливает значение одного свойства стиля css для всех элементов набора.

Позиционирование

offset( )
Возвращает текущие значения отступов относительно документа для первого элемента в наборе.
position( )
Возвращает значение позиции элемента сверху и слева относительно отступов его “родителя”.
scrollTop( )
Возвращает значение отступа прокрутки сверху для первого элемента в наборе.
scrollTop(значение)
Устанавливает значение отступа прокрутки сверху для всех элементов набора.
scrollLeft( )
Возвращает значение отступа прокрутки слева для первого элемента в наборе.
scrollLeft(значение)

Устанавливает значение отступа прокрутки слева для всех элементов набора.

Базовые эффекты

show( )
Отображает каждый из совпавших элементов набора, если они были скрыты (не имеет значения как: либо через функцию hide(), либо посредством display:none в таблице стилей).

show(скорость, вызов)
Отображает все совпавшие элементы набора с использованием анимационных эффектов, после чего опционально запускает указанную в аргументе вызов функцию.

hide( )
Скрывает каждый из совпавших элементов набора, если они были видимыми.

hide(скорость, вызов)
Скрывает все совпавшие элементы набора с использованием анимационных эффектов, после чего опционально запускает указанную в аргументе вызов функцию.

toggle( )
Переключает режим отображения каждого из элементов набора.

toggle(переключатель)
Переключает режим отображения каждого из элементов набора в зависимости от значение переключателя (true — отображает элементы, false — скрывает элементы).

toggle(скорость, вызов)
Переключает режим отображения всех совпавших элементов набора с использованием анимационных эффектов, после чего опционально запускает указанную в аргументе вызов функцию.

Эффект «скольжения»:

slideDown(скорость, вызов)
Раскрывает все элементы набора, используя эффект изменения высоты элементов. Также, по завершению операции возможен запуск функции, переданной в аргумент “вызов”.

slideUp(скорость, вызов)
Скрывает все элементы набора, используя эффект изменения высоты элементов. Также, по завершению операции возможен запуск функции, переданной в аргумент “вызов”.

slideToggle(скорость, вызов)
Переключает видимость всех совпавших элементов набора, используя эф-

фект изменения высоты элементов. Также, по завершению операции возможен запуск функции, переданной в аргумент “вызов”.

Эффекты появления/исчезания:

fadeIn(скорость, вызов)

Делает видимыми все элементы набора, используя изменение прозрачности элементов. Также, по завершению операции возможен запуск функции, переданной в аргумент “вызов”.

fadeOut(скорость, вызов)

Делает невидимыми все элементы набора, затем устанавливает CSS свойство display в “none”. Также, по завершению операции возможен запуск функции, переданной в аргумент “вызов”.

fadeTo(скорость, видимость, вызов)

Делает менее видимыми все элементы набора, изменяя прозрачность элементов до величины, указанной в аргументе “видимость”.

Пользовательская анимация:

animate(параметры, duration, easing, вызов)

Данная функция предназначена для создания пользовательской анимации.

animate(параметры, опции)

Данная функция предназначена для создания пользовательской анимации.

stop(clearQueue, gotoEnd)

Останавливает все запущенные анимационные эффекты для всех указанных элементов.

Пример использования jQuery.

Задача: по клику на кнопку скрыть блок текста, по клику на другую кнопку – отобразить текст.

Скрытие и отображение блока текста по клику на кнопку. Листинг 8.10

```
<script type="text/javascript">
$(function(){
  $("button:eq(0)").click(function(){
    $("#test").fadeOut(2500,function(){
$("p:first").text("fadeOut"); });
  });
  $("button:eq(1)").click(function(){
    $("#test").fadeIn(2500,function(){ $("p:first").text("fadeIn");
});
  });
});
</script>
</head>
<body>
<p class="result"> </p>
```

```
<button>Hide</button><button>Show</button>
<div id="test">
<p>Я шел, спускаясь в темные коридоры, и потом опять поднимаясь
наверх. Я был один. Я кричал, мне не отвечали - я был один в этом
обширном, запутанном, как лабиринт, доме.</p>
<p><em>Ги де Мопассан</em></p>
</div>
</body>
```

9. Ajax.

AJAX, или, более длинно, Asynchronous Javascript And Xml - технология для взаимодействия с сервером без перезагрузки страниц. За счет этого уменьшается время отклика и веб-приложение по интерактивности больше напоминает десктоп.

Начнем с самого простого – загрузка HTML кода в необходимый нам DOM элемент на странице. Для этой цели нам подойдет метод load.

jQuery(..).load

Данный метод может принимать следующие параметры:

1. url запрашиваемой страницы
2. передаваемые данные (необязательный параметр)
3. функция, которой будет скормлен результат (необязательный параметр)

Загрузка страницы по клику. Листинг 9.1

```
<script type="text/javascript">
$(function () {
  $("button:first").click(function(){
    $("#target").load("testLoad.htm");
  });
  $("button:last").click(function(){
    $("#target").empty();
  });
});
</script>
…
<div id="target"></div>
<button>Загрузить</button>
<button>Очистить</button>
```

Как видно из листинга, по клику на первую кнопку – произойдет загрузка всей страницы testLoad.htm в контейнер с идентификатором target. По клику на вторую кнопку - очисттится содержимое контейнера.

Если в выбранный элемент необходимо с помощью AJAX-запроса загрузить только некторые элементы указанной web-страницы, то помимо url-страницы

можно через пробел указать селекторы:

Загрузка выбранных селекторов по клику. Листинг 9.1

```
<script type="text/javascript">
$(function () {
  $("button:first").click(function(){
    $("#target").load("testLoad.htm h2,p:last", function(){
alert("Готово!"); });
  });
  $("button:last").click(function(){
    $("#target").empty();
  });
});
</script>
…
<div id="target"></div>
<button>Загрузить</button>
<button>Очистить</button>
```

Если же в выбранный элемент нужно загрузить ответ сервера сформированный на основании отправленных ему данных, то можно воспользоваться следующим листингом:

Загрузка выбранных селекторов по клику. Листинг 9.1

```
<script type="text/javascript">
$(function () {
  $("button:first").click(function(){
    $("#target").load("testLoad.php", { name: "John", age: "35" });
  });
  $("button:last").click(function(){
    $("#target").empty();
  });
});
</script>
…
<div id="target"></div>
<button>Загрузить</button>
<button>Очистить</button>
```

AJAX-запрос будет отправлен к файлу testLoad.php, который получит данные ввиде глобального массива $_POST[‘name’] и $_POST[‘age’] с соответствующими значениями.

$.ajax(options)

Основная функция для работы с AJAX-запросами – это функция ajax(), которая может представить гораздо больше возможностей управления AJAX-запросами, чем рассмотренная ранее функция load().

Использование вспомогательной функции ajax. Листинг 9.1

```
<script type="text/javascript">
$(function () {
  $("button:first").click(function(){
    $.ajax({
      url: "testAjax.php",
      type: "POST",
      data: "name=Jonh&age=35",
      timeout: 3000,
      beforeSend: function(){
        $("div").text("Загрузка...");
      },
      success: function(data){
        $("div").html(data);
      },
      error: function(xhr, status){
        $("div").html("<span>" + status + "</span>");
      }
    });
  });
  $("button:last").click(function(){
    $("div").empty();
  });
});
</script>
…
<div id="target"></div>
<button>Загрузить</button>
<button>Очистить</button>
```

Итак, рассмотрим основные опции функции ajax.

- url – файл, к которому будет отправлен запрос.
- type – способ передачи данных: $_POST либо $_GET
- timeout – время выполнения запроса, число в милисекундах. Работа ajax останавливается, если за данное время запрос не успел обработаться. Параметр необязательный.
- beforesend – функция срабатывающая, во время выполнения запроса. Т.к. по умолчанию, ajax-запросы передаются в асинхронном режиме, то в процессе выполнения запроса, мы можем выполнять любые другие функции.
- success - функция срабатывающая после успешного выполнения запроса.
- error – функция, выводящая ошибки запроса, если таковые имеются.
- data – строка с передаваемыми данными. Объект должен представлять собой пары ключ/значение.

Часто возникает следующая ситуация: все AJAX-запросы должны отправляться одному и тому же файлу, с использованием одних и тех же опций.

Различаться будут только отправляемые на сервер данные. В таком случае рациональнее воспользваться методом ajaxSetup()

Совместное использование ajaxSetap() и ajax(). Листинг 9.1

```
<script type="text/javascript">
$(function () {
  $.ajaxSetup({
    url: "testAjaxSetup.php",
    type: "POST",
    timeout: 3000,
    beforeSend: function(){
      $("div:last").empty();
      $("#result").text("Загрузка...");
    },
    success: function(data){
      $("div:last").html(data);
      $("#result").text("Готово!");
    },
    error: function(xhr, status){
      $("#result").html("<span>" + status + "</span>");
    }
  });
  $("button:eq(0)").click(function(){
    $.ajax({ data: "q=1&er=none" });
  });
  $("button:eq(1)").click(function(){
    $.ajax({ data: "q=2&er=none" });
  });
  $("button:eq(2)").click(function(){
    $.ajax({ data: "q=3&er=yes" });
  });
});
</script>
</head>
<body>
<div id="result"></div>
<div></div>
<button>Запрос №1</button>
<button>Запрос №2</button>
<button>Запрос №3 (с ошибкой)</button>
```

Как видно из листинга, все передаваемые параметры, за исключением параметра data, находятся в функции ajaxSetup. Задача функции ajax сводится к тому, чтобы связать конкретный селектор с данными, которые необходимо передать в файл-обработчик.

По щелчку на любую из кнопок листинга в testAjaxSetup.php будут поступать переменные $_POST[‘q’] и $_POST[‘er’].

Недостаток технологии Ajax:

Поисковая оптимизация. Если вы решили организовать с помощью AJAX'a навигацию сайта или просто несколько страниц, не забывайте, что поисковые роботы исполнять JavaScript пока не научились, поэтому индексация будет

затруднена. В качестве решения можно попробовать сделать доступным контент другим способом, например, сделать «Карту сайта» с полным списком страниц.

Кнопка «Назад». По статистике, кнопка браузера «Назад» является вторым по популярности средством навигации после перехода по ссылке. Страницы, созданные с использованием AJAX (читай JavaScript), такую возможность не поддерживают, потому что их содержание генерируется, а адрес остается один и тот же. Можно попробовать это исправить, сделав ссылку «Назад» средствами JavaScript, но это часто не оправдано сложно.

Избранное Как уже упоминалось выше, у нескольких страниц AJAX может быть один адрес, поэтому пользователь может добавить в избранное не то, что ожидает. Решение заключается в том, чтобы снабдить каждую страницу своим адресом.

Неопределенное время ответа. Время ответа сервера на запрос варьируется в зависимости от занятости последнего, то есть - не определено. А во врему загрузки данных с помощью AJAX браузер никак не отображает, что что-то происходит. Чтобы не оставлять пользователя в неведении, можно выводить надпись «Идет загрузка» или анимированное изображение.

JavaScript код, приходящий в ответе, не выполняется. Хотя это, в общем-то, логично, если вспомнить, что объект XMLHttpRequest - "душа" аякса - задумывался для работы с xml. Решение пока только одно - метод eval.

Не все гладко с IE. IE некорректно парсит js-код в HTML-куске, полученном через AJAX, если есть классические скрывающие комментарии <!-- //-->. Если передаваемый сервером заголовок с кодировкой не понятен IE, то возникает системная ошибка **-1072896658**

10. Поиск.

Этапы создания поиска по содержимому сайта:

1. Создание формы поиска. Это может быть обычная форма-html, либо объект класса field_text (рассмотренного ранее FrameWork-а). При необходимости можно снабдить поисковое поле фильтрами. **Поиск** – это выбор соответствий из базы по пользовательским запросам. При **фильтре** мы сами предлагаем пользователю возможные варианты выбора либо диапазона значений. На рисунке представлен вид поисковой

формы, в которой поиск смешан с фильтром.

2. Обработка формы поиска осуществляется в условии if(isset($_POST)){};
3. Проверка вводимых пользователем данных. Для этого сперва нужно проверить включена ли на сервере функция get_magic_quotes_gpc(). Если не включена, то необходимо вводимые пользователем данные экронировать для MySQL, т.е. пропустить через функцию mysql_escape_string.
4. Формирование переменных уточняющих пользовательский запрос. Каждый элемент формы – это уточняющая часть запроса. Например:

Формирование переменных, уточняющих пользовательский запрос. Листинг 10.1

```
      if(!empty($_POST['district']) && $_POST['district'] !=
'none')
            {
            $tmp1 = "& firma LIKE '%".trim($_POST[district])."%'";
// trim - избавление от пробелов в начале и в конце строки.
            }
```

5. Формирование SELECT-запроса.
6. Вывод полученного результата на экран.

А вот, непосредственно, и сам обработчик:

Обработка поисковой формы. Листинг 10.2

```
<?php
if(isset($_POST['search']))
{
```

```
          echo "Результаты поиска";

          // Защищаем данные от SQL-инъекции
          if (!get_magic_quotes_gpc())
          {
            $_POST['district'] =
mysql_escape_string($_POST['district']);
          }

          if(!empty($_POST['district']) && $_POST['district'] !=
'none')
            {
            $tmp1 = "& firma LIKE '%".trim($_POST[district])."%'";
            }
              if(!empty($_POST['name']) && $_POST['name'] !=
'none')
            {
            $tmp1 = "& firma LIKE '%".trim($_POST[name])."%'";
            }
          $query = "SELECT * FROM $catalog_firm WHERE id > 0
                    ".$tmp1." ORDER BY firma DESC LIMIT 31";

          // Выполняем SQL-запрос
          $cat = mysql_query($query);
          if(!$cat)
          {
           throw new ExceptionMySQL(mysql_error(),
            $query,
         "Ошибка при обращении  к таблице риэлторских услуг");
          }
          // количество рядов в наборе должно быть больше нуля
          if (mysql_num_rows($cat) > 0)
          {
    if (mysql_num_rows($cat) > 30)
    {
    ?>
    <div class="sobitie"><div class="tekst_tabs">
    <?php
    echo "<P><B>Внимание! По вашему запросу найдено слишком много
товаров. На экран выведено последние 30.</B></P>";
    ?>
    </div></div>
    <?php
    }
    ?>
    <div class="sobitie"><div class="tekst_tabs">

    <?php
    while($position = mysql_fetch_array($cat))
          {
              echo $position["name"];
          }

    ?>
    </div></div>
    <?php
          }
          else echo " Поиск не дал результатов.
```

```
                    Попробуйте изменить критерии поиска. ";
} ?>
```

11. Протокол HTTP. $_SESSION, $_COOKES. Сокеты. Библиотека cURL.

В ответ на запрос серверу клиент получает HTTP-документ, который состоит из HTTP-заголовков и тела документа, содержащего, как правило, HTML-страниц или изображение.

HTTP-заголовки, как правило, формируются сервером автоматически и в большинстве случаев Web-разработчику нет надобности отправлять их вручную. Впрочем, последнее справедливо только для PHP, тогда как, например, при создании CGI-программы при помощи Perl или C разработчик вынужден самостоятельно реализовывать эту часть HTTP-протокола.

Для управления HTTP-заголовками в PHP предназначены функции, представленные ниже.

Функция	Описание
header()	Отправляет HTTP-заголовок
headers_list()	Возвращает список отправленных или готовых к отправке HTTP-заголовков
headers_sent()	Проверяет, отправлены ли HTTP-заголовки

Простейшей процедурой, которую можно осуществить при помощи функции header(), является переадресация, осуществляемая при помощи HTTP-заголовка Location.

Переадресация при помощи HTTP-заголовка Location. Листинг 11.1

```
<?php
header("Location:http://colony.by");
?>
```

Вообще говоря, механизм HTTP-заголовков дублирован в языке разметки HTTP и добиться схожего поведения можно при помощи передачи HTTP-заголовка через META-тег

Переадресация средствами HTML. Листинг 11.2

```
<HTML>
<HEAD>
<MTA HTTP-EQUIV= 'Refresh' CONTENT='O ; URL=http://colony.by'>
</HEAD>
</HTML>
```

На каждый запрос клиента сервер может возвращать HTTP-код состояния, отражающий вид переадресации при окончательном йли временном перемещении документа. Если документ найден и успешно отправлен клиенту, в HTTP-заголовки помещается код состояния 200; если документ не найден - 404;

Коды состояния разделяются на классы, каждый из которых начинается с новой сотни.

Информационные ответы

Ответы в диапазоне 100-199 - информационные; они показывают, что запрос клиента принят и обрабатывается.

100 Continue

Начальная часть запроса принята, и клиент может продолжать передачу запроса.

101 Switching Protocols

Сервер выполняет требование клиента и переключает протоколы в соответствии с указанием, данным в поле заголовка Upgrade.

Успешные запросы клиента

Ответы в диапазоне 200-299 означают, что запрос клиента обработан успешно.

200 OK

Запрос клиента обработан успешно, и ответ сервера содержит затребованные данные.

201 Created

Этот код состояния используется в случае создания нового URI. Вместе с этим кодом результата сервер выдает заголовок Location (см. главу 19), который содержит информацию о том, куда были помещены новые данные.

202 Accepted

Запрос принят, но обрабатывается не сразу. В теле содержимого ответа сервера может быть дана дополнительная информация о данной транзакции. Гарантии того, что сервер в конечном итоге удовлетворит допустимым.

203 Non-Authoritative Information

Информация в заголовке содержимого взята из локальной копии или у третьей стороны, а не с исходного сервера.

204 No Content

Ответ содержит код состояния и заголовок, но тело содержимого отсутствует. При получении этого ответа броузер не должен обновлять свой документ. Обработчик чувствительных областей изображений может возвращать этот код, когда пользователь щелкает на бесполезных или пустых участках изображения.

205 Reset Content

Броузер должен очистить форму, используемую в данной транзакции, для дополнительных входных данных. Полезен для CGI-приложений, требующих ввода данных.

206 Partial Content

Сервер возвращает лишь часть данных затребованного объема. Используется в ответе на запрос с указанием заголовка Range. Сервер должен указать диапазон, включенный в ответ, в заголовке Content-Range.

Переадресация

Код ответа в диапазоне 300-399 означает, что запрос не выполнен и клиенту нужно предпринять некоторые действия для удовлетворения запроса.

300 Multiple Choices

Затребованный URI обозначает более одного ресурса. Например, URI может обозначать документ, переведенный на несколько языков. В теле содержимого, возвращенном сервером, может находиться перечень более конкретных данных о том, как выбрать ресурс правильно.

301 Moved Permanently

Затребованный URI уже не используется сервером, и указанная в запросе операция не выполнена. Новое местонахождение затребованного документа

указывается в заголовке Location. Во всех последующих запросах данного документа следует указывать новый URI.

302 Moved Temporarily

Затребованный URI перемешен, но лишь временно. Заголовок Location указывает на новое местонахождение. Сразу же после получения этого кода состояния клиент должен разрешить запрос при помощи нового URI, но во всех последующих запросах необходимо пользоваться старым URI.

303 See Other

Затребованный URI можно найти по другому URI (указанному в заголовке Location). Его следует выбрать методом GET по данному ресурсу.

304 Not Modified

Это код ответа на заголовок lf-Modified-Since, если URI не изменялся с указанной даты. Тело содержимого не посылается, и клиент должен использовать свою локальную копию.

305 Use Proxy

Доступ к затребованному URI должен осуществляться через proxy-сервер, указанный в заголовке Location.

Неполные запросы клиента

Коды ответов в диапазоне 400-499 означают, что запрос клиента неполный. Эти коды могут также означать, что от клиента требуется дополнительная информация.

400 Bad Request

Означает, что сервер обнаружил в запросе клиента синтаксическую ошибку.

401 Unauthorized

Этот код результата, передаваемый с заголовком WWW-Authenticate, показывает, что пославший запрос пользователь не имеет необходимых полномочий и что при повторении запроса с указанием данного URI пользователь должен такие полномочия предоставить.

402 Payment Required

Этот код в HTTP еще не реализован.

403 Forbidden

Запрос отклонен по той причине, что сервер не хочет (или не имеет возможности) ответить клиенту.

404 Not Found

Документ по указанному URI не существует.

405 Method Not Allowed

Этот код выдается с заголовком Allow и показывает, что метод, используемый клиентом, для данного URI не поддерживается.

406 Not Acceptable

Ресурс, указанный клиентом по данному URI, существует, но не в том формате, который нужен клиенту. Вместе с этим кодом сервер выдает заголовки Content-Language, Content-Encoding и Content-Type.

407 Proxy Authentication Required

Proxy-сервер должен санкционировать запрос перед тем, как пересылать его. Используется с заголовком Proxy-Authenticate.

408 Request Time-out

Этот код ответа означает, что клиент не передал полный запрос в течение некоторого установленного промежутка времени (который обычно задается в конфигурации сервера) и сервер разрывает сетевое соединение.

409 Conflict

Данный запрос конфликтует с другим запросом или с конфигурацией сервера. Информацию о конфликте следует возвратить в информационной части ответа.

410 Gone

Данный код показывает, что затребованный URI больше не существует и навсегда удален с сервера.

411 Length Required

Сервер не примет запрос без указанного в нем заголовка Content-Length.

412 Precondition Failed

Результат вычисления условия, заданного в запросе одним или несколькими заголовками if. . ., представляет собой "ложь".

413 Request Entity Too Large

Сервер не будет обрабатывать запрос, потому что его тело слишком велико.

414 Request-URI Too Long

Сервер не будет обрабатывать запрос, потому что его URI слишком длинный.

415 Unsupported Media Type

Сервер не будет обрабатывать запрос, потому что его тело имеет неподдерживаемый формат.

Ошибки сервера

Коды ответов в диапазоне 500-599 показывают, что сервер столкнулся с ошибкой и, вероятно, не сможет выполнить запрос клиента.

500 Internal Server Error

При обработке запроса на сервере один из его компонентов (например, CGI-программа) выдал аварийный отказ или столкнулся с ошибкой конфигурации.

501 Not Implemented

Клиент запросил выполнение действия, которое сервер выполнить не может.

502 Bad Gateway

Сервер (или proxy-сервер) получил недопустимые ответы другого сервера (или proxy-сервера).

503 Service Unavailable

Данный код означает, что данная служба временно недоступна, но в будущем доступ к ней будет восстановлен. Если сервер знает, когда это произойдет, может быть также выдан заголовок Retry-After.

504 Gateway Time-out

Этот ответ похож на 408 (Request Time-out) , за исключением того, что шлюз или уполномоченный сервер превысил лимит времени.

505 HTTP Version not supported

Сервер не поддерживает версию протокола HTTP, использованную в запросе.

***

При работе с HTTP-заголовками следует помнить, что они всегда предваряют содержимое страницы. Вывод любой информации (при помощи конструкции echo, функции print () или непосредственно вне тегов <?php и ?» В окно браузера при водит к тому, что начинается отправка HTTP-документа.

Даже переход на новую строку блокирует работу отправки заголовка.

Иногда бывает полезно проконтролировать, какие HTTP-заголовки были отправлены клиенту. Дя этого удобно воспользоваться функцией headers_list(), которая имеет следующий синтаксис:

array headers_list ()

Использование функции headers_list(). Листинг 11.3

```
<?php
  header("X-my-header: Hello world!");
  $arr = headers_list();
  echo "<pre>";
  print_r($arr);
  echo "</pre>";
?>
```

Изменение типа файла

Изменение типа файла. Листинг 11.4

```
<?php
  // Весь вывод направляем в буфер
  ob_start();
  // Выводим содержимое страницы
  for($i = 0; $i < 30000; $i++) echo $i."<br>";

  // Задаем имя, которое будет предложено
  // клиенту для сохранения файла
  header("Content-Disposition: attachment; filename=text.txt");
  // В качестве типа файла задаем бинарный поток
  header("Content-type: application/octet-stream");
  // Отправляем клиенту размер страницы
  // в HTTP-заголовке Content-length
  header("Content-length: ".ob_get_length());

  // Отправляем содержимое буфера вывода клиенту
  ob_end_flush();
?>
```

PHP представляет альтернативный способ управления буферизацией вывода.

Функции управления выводом

Функция	Описание
ob_clean()	Очищает буфер вывода
ob_end_clean()	Очищает буфер вывода и отключает буферизацию вывода
ob_end_flush()	Отправляет буфер вывода клиенту и отключает буферизацию вывода
ob_flush()	Отправляет буфер вывода клиенту
ob_get_clean()	Возвращает текущее содержание буфера и удаляет текущий буфер
ob_get_contents()	Возвращает содержимое буфера вывода
ob_get_flush()	Очищает буфер, возвращая его содержимое в виде строки, и отключает буферизацию
ob_get_length()	Возвращает размер буфера вывода
ob_get_level()	Возвращает уровень вложения буфера
ob_get_status([$full_status])	Возвращает статус буфера вывода в виде массива; если необязательный параметр $full_status принимает значение true, возвращается более подробная информация.
ob_gzhandler($buffer, $mode)	Функция обратного вызова, которая используется функцией ob_start() для задания обработчика сжатия данных
ob_implicit_flush([$flag])	Включает (если значение параметра $flag устанавливается в 1) или выключает (если значение $flag устанавливается в 0) неявную очистку буфера
ob_list_handlers()	Возвращает список всех используемых обработчиков буферизации
ob_start([$output_callback [, $chunk_size [, $erase]]])	Включает буферизацию вывода; в качестве параметра $output_callback может принимать название функции обратного вызова, которую задает обработчик. Необязательный параметр $chunk_size задает количество байт буфера, через которые будет осуществляться повторный вызов функции $output_callback. Если необязательный параметр $erase принимает значение false, то буфер не будет удаляться после завершения работы скрипта
output_add_rewrite_var ($name, $value)	Добавляет к URL, обнаруженным в буфере вывода, дополнительный GET-параметр с именем $name и значением $value
output_reset_rewrite_vars()	Уничтожает дополнительные GET-параметры, добавленные

	функцией output_add_rewrite_var()

Подавление кэширования

Механизм кэширования применяется с целью оптимизации пересылки данных между клиентом и сервером. Запрошенный пользователем по HTTP-протоколу документ может быть сохранен в кэше промежуточного сервера или браузера, и при повторном запросе будет выдаваться без обращения к источнику.

Кэши принято подразделять на два вида: локальные и глобальные. Локальный кэш создается браузером клиента, тогда как глобальный располагается на прокси-сервере провайдера (или организации, в которой имеется свой внутренний прокси-сервер).

Примечание. Прокси-сервером, в отличие от обычного сервера, предоставляющего доступ какому-либо ресурсу, называют сервер-посредник, расположенный между клиентом и обычным сервером. В отличие от шлюз-сервера, осуществляющего обычное транслирование запросов клиента и ответов сервера, в задачи пркси-сервера входит предоставление дополнительных услуг, таких как преобразование медиатипа, анонимная фильтрация, сжатие протокола, кэширование и др.

В большинстве случаев кэширование позволяет ускорить работу с Интернетом и значительно снизить трафик, но иногда кэширование может мешать работе веб-приложений. Если посетители часто загружают страницы веб-сайта с редко изменяющейся информацией, такой как рассписание пригородного транспорта, кэширование полезно, поскольку экономит трафик и сокращает время выполнения запроса. Если же посетитель загружает динамические страницы, например, активного форума, кэширование, без сомнения, вредно, поскольку содержимое таких страниц меняется слишком часто, и посетитель вынужден будет постоянно в ручную обновлять содержимое страницы.

Для подавления кэширования можно использовать HTTP-заголовки.

Примечание. К сожалению, в последнее время кэширующие прокси-серверы настраиваются крайне агрессивно. Дело в том, что динамические страницы, созданные например, при помощи PHP, вообще не должны подвергаться кэшировнию. Однако сплошь ирядом наблюдается обратное. Более того, ряд кэширующих серверов игнорируют HTTP-заголовки, подавляющие кэширование. В этом случае разработчику ничего не остается делать, как добавлять к URL GET-параметр со случайно сгенерируемым значением.

Подавление кэширования. Листинг 11.5

```
<?php
//любая дата в прошлом
header ("Expires: Mon, 23 May 2008 02:00:00 GMT");
header ("Last-Modified: " .gmdate(D, d M Y H:i:s"). "GMT";
header ("Cache-Control: no-cache, must-revalidate");
header ("Pragma: no-cache");
?>
```

HTTP-заголовок Expires задает дату, по достижении которой документ считается устаревшим, поэтому задание для этого загаловка уже прошедшей даты предотвращает кэширование данной страницы.

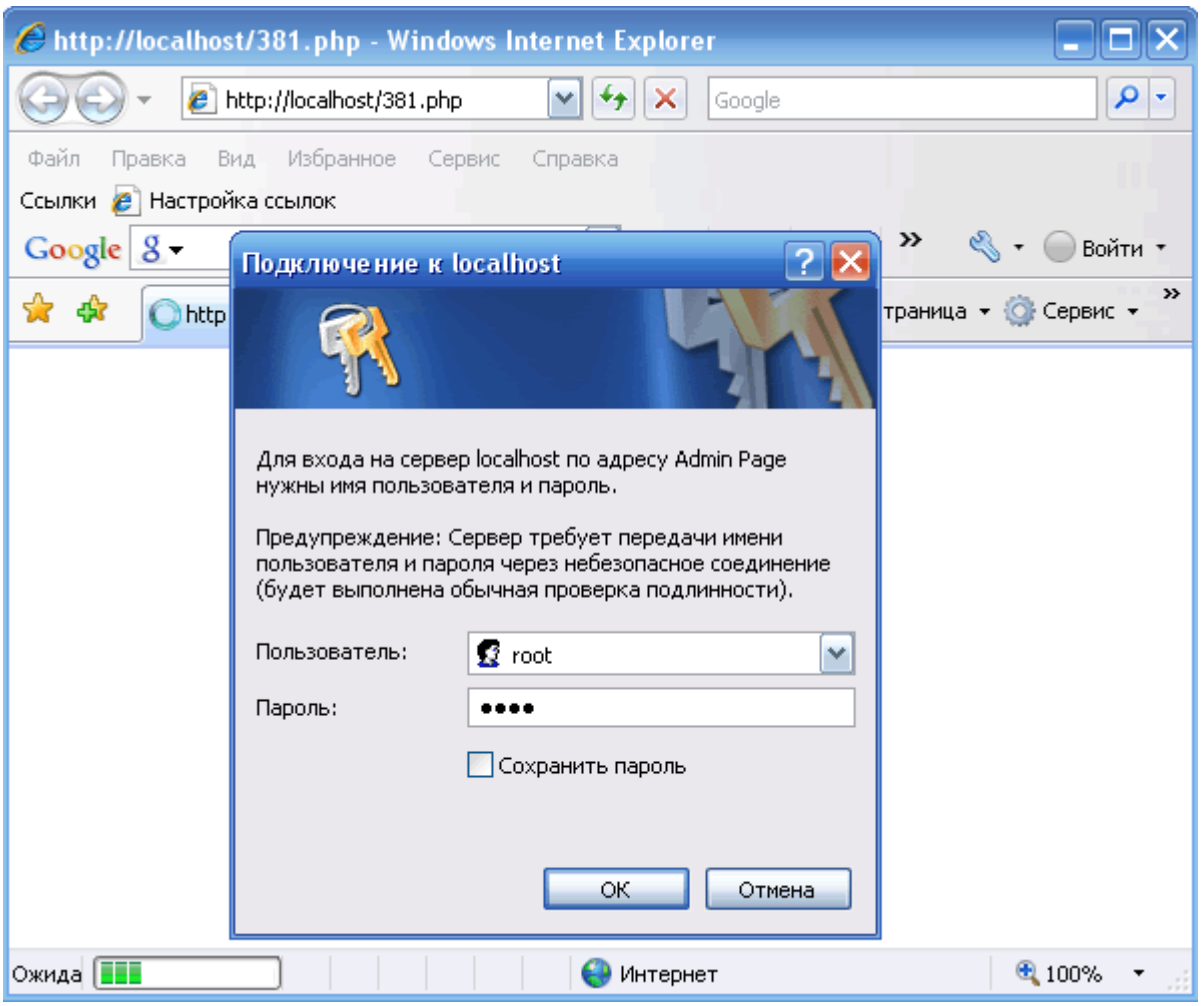
HTTP-заголовок Last-Modified определяет дату последнего изменения веб-страницы. Если с момента последнего обращения к ней прокси-сервера значение параметра этого заголовка изменилось, происходит повторная загрузка страницы, поэтому присвоение этой директиве текущего времени при каждом обращении предотвращает кэширование.

HTTP-заголовок Cache-Control предназначен для управления кэшированием, и указание его значения равным no-cache также приводит к запрету кэширования. В устаревшем стандарте HTTP 1.0 для запрета кэширования нужно присвоить значение no-cache HTTP-заголовку Pragma.

Для того, чтобы сообщить промежуточному прокси-серверу о том, что данный документ можно кэшировать, HTTP-заголовку Cache-Control следует передать значение public. Если информация не предназначена для публичных кэш-серверов и может быть сохранена только в локальном кэше браузера, HTTP-заголовку Cache-Control следует передать значение private.

Базовая аутентификация.

Веб-сервер позволяет защитить страницу при помощи базовой аутентификации — пока пользователь не введет правильные логин и пароль, он не будет допущен к страницам сайта.



Имя пользователя будет помещено сервером в элемент суперглобального массива $_SERVER['PHP_AUTH_USER'], а пароль в $_SERVER['PHP_AUTH_PW'].

Примечание. Элементы суперглобального массива $_SERVER['PHP_AUTH_USER'] и $_SERVER['PHP_AUTH_PW'] доступны только в том случае, если PHP установлен в качестве модуля, а не CGI-приложения.

Для удобства код аутентификации удобно выделить в отдельный файл security_mod.php, включение которого при помощи директивы require_once() будет приводить к защите страницы паролем.

Файл security_mod. Листинг 11.6

```
<?php
  // Если пользователь не авторизовался - авторизуемся
  if(!isset($_SERVER['PHP_AUTH_USER']) || (!emp-
ty($_GET['logout']) && $_SERVER['PHP_AUTH_USER'] ==
$_GET['logout']))
  {
    Header("WWW-Authenticate: Basic realm=\"Control Page\"");
    Header("HTTP/1.0 401 Unauthorized");
    exit();
```

```
  }
  else
  {
    // Утюжим переменные $_SERVER['PHP_AUTH_USER'] и
$_SERVER['PHP_AUTH_PW'],
    // чтобы мышь не проскочила
    if (!get_magic_quotes_gpc())
    {
      $_SERVER['PHP_AUTH_USER'] =
mysql_escape_string($_SERVER['PHP_AUTH_USER']);
      $_SERVER['PHP_AUTH_PW']   =
mysql_escape_string($_SERVER['PHP_AUTH_PW']);
    }

    $query = "SELECT * FROM $tbl_accounts
              WHERE name = '".$_SERVER['PHP_AUTH_USER']."'";
    $lst = @mysql_query($query);
    // Если ошибка в SQL-запросе - выдаём окно
    if(!$lst)
    {
      Header("WWW-Authenticate: Basic realm=\"Control Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
    // Если такого пользователя нет - выдаём окно
    if(mysql_num_rows($lst) == 0)
    {
      Header("WWW-Authenticate: Basic realm=\"Control Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
    // Если все проверки пройдены, сравниваем хэши паролей
    $account = @mysql_fetch_array($lst);
    if(md5($_SERVER['PHP_AUTH_PW']) != $account['pass'])
    {
      Header("WWW-Authenticate: Basic realm=\"Control Page\"");
      Header("HTTP/1.0 401 Unauthorized");
      exit();
    }
  }
?>
```

Как видно из листинга, при неудачной авторизации клиенту отправляется повторное приглашение для ввода пароля:

WWW-Authenticate: Basic realm = "Admin Page"

HTTP/1.0 401 Unauthorized

Далее работа скрипта останавливается при помощи функции exit(). Дополнительно можно реализовать ограничение количества попыток ввода пароля. Для этого достаточно вместо приведенных выше HTTP-заголовков послать заголовок "Страница не найдена":

HTTP/1.0 404 Not Found

После того, как модуль защиты security_mod.php создан, его можно включить перед загрузкой защищаемых страниц при помощи директивы require_once().

Работа с cookie.

Cookies — это небольшие файлы, сохраняемые просматриваемыми серверами на машине посетителя и содержащие текстовую информацию о настройках пользователя, доступную для считывания создавшему их серверу.

Для создания cookie предназначена функция setcookie(), которая имеет следующий синтаксис:

bool setcookie ($name [, $value [, $expire [, $path [, $domain [, $secure]]]]])

Функция принимает следующие аргументы:

* $name — имя cookie;

* $value — значение, хранящееся в cookie с именем name;

* $expire — время в секундах с 1 января 1970 года. По истечении этого времени cookie удаляется с машины клиента;

* $path — путь, по которому доступен cookie;

* $domain — домен, из которого дотупен cookie;

* $secure — директива, определяющая, доступен ли файл cookie по защищенному протоколу HTPPS. По умолчанию эта директива имеет значение 0, что означает возможность доступа к cookie по стандартному незащищенному протоколу HTTP.

Данная функция возвращает true при успешной установке cookie на машине клиента и false —в противном случае. После того как cookie установлен, его значение можно получить на всех страницах веб-приложения, обращаясь к суперглобальному массиву $_COOKIE и используя в качестве ключа имя cookie.

Так как cookie передается в заголовок HTTP-запроса, то вызов функции setcookie() необходимо размещать до начала вывода информации в окно браузера функциями echo(), print() и т.д., а также до включения в файл HTML-тегов.

Примечание. Файл для cookie создается только в том случае, если выставляется время жизни cookie; в противном случае cookie действует только до конца сеанса, т.е.до того момента, пока пользователь не закроет окно браузера.

Подсчет количества обращений с странице. Листинг 11.7

```
<?php
//увеличиваем значение cookie
$_COOKIE['counter']++;
//устанавливаем cookie
setcookie(counter, $_COOKIE['counter']);
//выводим значение cookie
echo "Вы посетили эту страницу $_COOKIE[counter] раз";
?>
```

Установка cookies с определенным временем жизни. Листинг 11.8

```
<?php
  // cookie действительна в течение 10 мин после создания
  setcookie("name", "value", time() + 600);
  // действие этой cookie прекращается в полночь 25 января 2011
года
  setcookie("name", "value", mktime(0,0,0,1,25,2011));
  // действие этой cookie прекращается в 18.00 25 января 2011
года
  setcookie("name", "value", mktime(18,0,0,1,25,2011));
?>
```

Нередко посетители отключают cookies в настройках своих браузеров. Для корректрой работы в веб-приложение, использующее cookie, необходимо помещать код, проверяющий, включены ли cookies у посетителя. Такая проверка позволит использовать другой механизм аутентификации, например сессии, или просто позволит вывести сообщение о необходимости включить cookies.

Проверка, включены ли cookies. Листинг 11.9

```
<?php
//выставляем уровень обработки ошибок
error_reporting(E_ALL & ~E_NOTICE);
if(!isset($_GET['probe']))
{
//устанавливаем cookie с именем "test"
if(setcookie("test", "set"))
//отправляем заголовок переадресации на страницу,
//с которой будет предпринята попытка установить cookie
header ("Location: $_SERVER[PHP_SELF] ?probe=set");
}
else
{
if(!isset($_COOKIE['test']))
{
echo "Для корректной работы приложения необходимо включить
cookies";
}
else
{
echo "Cookies включены";
}
}
?>
```

Работа с сессиями

Сессия во многом походит на cookie и представляет собой текстовой файл хранящий пары ключ/значение, но уже не на машине клиента, а на сервере. Во многих случаях сессии являются более предпочтительным вариантом, чем cookies.

Так как на сервере скапливается большое количество файлов, принадлежащих сессиям разных клиентов, то для их идентификации каждому новому клиенту назначается уникальный номер — идентификатор сессии (SID), который передается либо через строку запроса, либо через cookie, если они доступны. К недостаткам сессий относится невозможность контроля времени их жизни из PHP-скриптов, так как этот параметр задается в конфигурационном файле php.ini директивой session. cookie_lifetime.

Оба механизма, сессии и cookie, взаимно дополняют друг друга. Cookies хранится на машине посетителя, и продолжительность их жизни определяет разработчик. Обычно они применяются для долгосрочных задач (от нескольких часов) и хранения информации, которая относится исключительно к конкретному посетителю (личные настройки, логины, пароли и т.д.). В свою очередь, сессии хранятся на сервере, и продолжительность из существования (обычно не большую) определяет администратор сервера. Они предназначены для краткосрочных задач (до нескольких часов) и хранения и обработки информации обо всех посетителях в целом (количество посетителей on-line и т.д.). Поэтому использовать тот или иной механизм следует в зависимости от преследуемых целей.

Узнаем текущий идентификатор сессии. Листинг 11.10

```
<?php
  // Инициируем сессию
  session_start();
  // Узнаем текущий идентификатор сессии
  echo session_id();
?>
```

Сокеты

Подключение к удаленному серверу можно осуществить с помощью функции fsockopen(), которая имеет следующий синтаксис:

resouce fsockopen ($target, $port [, $errno [, $errstr [, $timeout]]])

Функция принимает пять параметров, первый из которых содержит адрес соединения, а второй номер порта. В случае удачи, функция возвращает де-

скриптор соединения $sock, в противном случае — значение false. Если при этом функции переданы два необязательных параметра $errno и errstr, в них размещается код и текстовое описание ошибки соответственно. Пятый, также необязательный параметр — значение тайм-аута, определяющего максимальное время ожидания ответа сервера в секундах. При успешной установке соединения функция возвращает дескриптор соединения, при неудаче — false.

Рассмотрим пример работы функции fsockopen().

Создадим функцию с двумя входящими параметрами: хостом и подключаемой страницей.

Функция get_content для извлечения внешнего контента. Листинг 11.11

```
<?php
function get_content($hostname, $path)
{
$line="";
//устанавливаем соединение, имя которого
//передано в параметре $hostname
$fd=fsockopen($hostname, 80, $errno, $errstr, 30);
//проверяем успешность установки соединения
if(!fd) echo "$errstr ($errno)<br>/>\n";
else
{
//формируем HTTP-запрос для передачи его серверу
$headers="GET $path HTTP/1.1\r\n";
$headers.="Host: $hostname\r\n";
$headers.="Connection: Close\r\n\r\n";
//отправляем HTTP-запрос серверу
fwrite ($fd, $headers);
//получаем ответ
while (!feof($fd))
{
$line=fgets($fd, 1024);
}
fclose($fd);
}
return $line;
}
?>
```

С помощью созданной функции загрузим главную страницу портала http://www.php.net/

Пример использования функции fsockopen. Листинг 11.12

```
<?php
require_once(“get_content.php”);
$hostname="www.php.net";
$path="/";
//устанавливаем большее время работы
//скрипта- пока вся страница не загружена,
//она не будет отображаться
```

```
set_time_limit(180);
//вызываем функцию
echo get_content($hostname, $path);
?>
```

Еще один пример использования данной функции: выведем на экран первые десять запросов “разработка сайтов под ключ” взятые из google.

Парсим google. Листинг 11.13

```
require_once(“get_content.php”);
$hostname="www.google.by";
$path="/search?q=разработка+сайтов+под+ключ&ie=utf-8&oe=utf-
8&aq=t&rls=org.mozilla:ru:official&client=firefox";
//устанавливаем большее время работы
//скрипта- пока вся страница не загружена,
//она не будет отображаться
set_time_limit(180);
//вызываем функцию
$content=get_content($hostname, $path);
 $content=strstr($content,'<');
   //$www = preg_match_all("<cite>mikhalkevich.colony.by<cite>",
$content, $cont);
 preg_match_all("/<cite>.*<\/cite>/isU", $content, $cont,
PREG_PATTERN_ORDER);
   echo '<pre>';
   print_r ($cont);
```

При работе с сокетами мы вынуждены брать на себя всю черновую работу по отправке серверу HTTP-запросов, получению о обработке ответов на них. В приведенном примере обращение к fsockopen() оформлено в виде функции get_content(), которая получает два параметра: $hostname — имя сервера, с которым устанавливается соединение, и $path — путь к странице относительно имени сервера.

А вот еще один пример использования данной функции:

Библиотека CURL

Помимо сокетов, обеспечивающих низкоуровневое обращение к серверу, PHP располагает специальным расширением CURL (Client URL Library).

Примечание. В случае расширения CURL не требуется удаление HTTP-заголовков, возвращаемых сервером, так как библиотека их удаляет по умолчанию. Однако CURL можно настроить на выдачу HTTP-заголовков, передаваемых сервером, если установит при помощи функции curl_setopt() ненулевое значение параметра CURLOPT_HEADER.

Внимание! Перед запуском скрипта, проверьте подключена ли библиотека CURL в конфигурационном файле php. ini. Для подключения библиотеки

раскоментируйте строку extension=php_curl.dll, иначе скрипт работать не будет.

Загрузка страницы с использованием расширения CURL. Листинг 11.14

```
<?php
//задаем адрес удаленного сервера
$curl=curl_init('http://www.php.net');
//устанавливаем параметры соединения
curl_setopt($curl, CURLLOPT_RETURNTRANSFER, 1);
//получаем содержимое страницы
$content=curl_exec($curl);
//закрываем CURL-соединение
curl_close($curl);
echo $content;
?>
```

Получение HTTP-заголовков с сервера

Загружаем только заголовки HTTP-ответа. Листинг 11.15

```
<?php
//функция получения HTTP-заголовков
function get_content($hostname, $path)
{
$line="";
//устанавливаем соединение, имя которого
//передано в параметре $hostname
$fd=fsockopen($hostname, 80, $errno, $errstr, 30);
//проверяем успешность установки соединения
if(!fd) echo "$errstr ($errno)<br/>\n";
else
{
//формируем HTTP-запрос для передачи его серверу
$headers="GET $path HTTP/1.1\r\n";
$headers.="Host: $hostname\r\n";
$headers.="Connection: Close\r\n\r\n";
//отправляем HTTP-запрос серверу
fwrite ($fd, $headers);
$end=$false;
//получаем ответ
while (!$end)
{
$line=fgets($fd, 1024);
if(trim($line)=="") $end=true;
else $out[]=$line;
}
fclose($fd);
}
return $out;
}
$hostname="www.php.net";
$path="/";
//устанавливаем большее время работы
//скрипта- пока вся страница не загружена,
//она не будет отображаться
set_time_limit(180);
//вызываем функцию
```

```
$out=get_content($hostname, $path);
//выводим содержимое массива
echo "<pre>";
print_r($out);
echo "</pre>";
?>
```

Результат работы функции выглядит следующим образом:

Array (

[0] => HTTP/1.1 200 OK

[1] => Date: Thu, 09 Apr 2009 10:37:36 GMT

[2] => Server: Apache/1.3.41 (Unix) PHP/5.2.9RC3-dev

[3] => X-Powered-By: PHP/5.2.9RC3-dev

[4] => Last-Modified: Thu, 09 Apr 2009 09:40:13 GMT

[5] => Content-language: en

[6] => Set-Cookie: COUNTRY=RUS%2C85.141.158.205;

expires=Thu,16-Apr-2009 10:37:36 GMT; path=/;

domain=.php.net

[7] => X-PHP-QUESTION:

I wonder if anyone will ever notice this

[8] => Connection: close

[9] => Transfer-Encoding: chunked

[10] => Content-Type: text/html;charset=utf-8)

Первая строка является стандартным ответом сервера о том, что запрос успешно обработан (код ответа 200). Если запрашиваемый ресурс не существует, то будет возвращен код ответа 404 (HTTP/1.1 404 Not Found).

Второй заголовок сообщает время формирования документа на сервере, необходимое для кэширования, которое рассматривается ниже.

Третий заголовок сообщает тип и версию веб-сервера. Как можно видеть портал http://www.php.net работает на сервере под управлением Unix, исполь-

зуя в качестве веб-серера Apache/1.3.41, а также PHP версии PHP/5.2.9RC3-dev.

Четвертый заголовок сообщает, что страница сгенерирована при помощи PHP версии 5.2.9RC3-dev.

Пятый заголовок сообщает о дате последней модификации страницы, впрчем, при динамическом формировании содержимого данного сайта он не актуален.

Шестой заголовок сообщает, что браузеру передаются данные на английском языке.

Седьмой заголовок устанавливает cookie с именем COUNTRY и значением RUS%2C85.141.158.205, содержащим страну пользователя и его IP-адрес сроком на неделю для домена .php.net. Данная информация необходима для раздела downloads сайта http://www.php.net, в котором посетителю предлагается ближайший к нему сервер.

Восьмой заголовок передает клиенту просьбу закрыть соединение после получения ответа.

Девятый заголовок Transfer-Encoding (имеющий значение chunked) указывает получателю, что ответ разбит на фрагменты.

Десятый заголовок Content-Type указывает тип загружаемого документа (text/html) и его кодировку (charset=utf-8).

В рассмотренном примере, для получения заголовков использовался запрос GET, однако для этой цели в протоколе HTTP предусмотрен специальный метод HEAD. Таким образом мы можем переписать скрипт более простым способом.

Примечание. Метод HEAD может быть запрещен на HTTP-сервере, поэтому иногда целесообразнее пользоваться методом GET, отсекая тело документа.

Использование метода HEAD. Листинг 11.16

```
<?php
//функция получения HTTP-заголовков
function get_content($hostname, $path)
{
$line="";
//устанавливаем соединение, имя которого
//передано в параметре $hostname
$fd=fsockopen($hostname, 80, $errno, $errstr, 30);
//проверяем успешность установки соединения
if(!fd) echo "$errstr ($errno)<br/>\n";
else
{
```

```
//формируем HTTP-запрос для передачи его серверу
$headers="HEAD $path HTTP/1.1\r\n";
$headers.="Host: $hostname\r\n";
$headers.="Connection: Close\r\n\r\n";
//отправляем HTTP-запрос серверу
fwrite ($fd, $headers);//получаем ответ
while (!feof($fd))
{
$out[]=fgets($fd, 1024);
}
fclose($fd);
}
return $out;
}
$hostname="www.php.net";
$path="/";
//устанавливаем большее время работы
//скрипта- пока вся страница не загружена,
//она не будет отображаться
set_time_limit(180);
//вызываем функцию
$out=get_content($hostname, $path);
//выводим содержимое массива
echo "<pre>";
print_r($out);
echo "</pre>";
?>
```

Еще более компактно рассматриваемую задачу можно решить с использованием расширения CURL, установив при помощи функции curl_setopt() параметры CURLOPT_HEADER и CURLOPT_NOBODY, первый из которых требует включения в результат HTTP-заголовков, а второй — игнорирования тела HTTP-документа.

Использование CURL. Листинг 11.17

```
<?php
function get_content($hostname)
{
//задаем адрес удаленного сервера
$curl=curl_init($hostname);
//вернуть результат в виде строки
curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
//включить в результат HTTP-заголовки
curl_setopt($curl, CURLOPT_HEADER, 1);
//исключить тело HTTP-документа
curl_setopt($curl, CURLOPT_NOBODY, 1);
//получаем HTTP-заголовки
$content=curl_exec($curl);
//закрываем CURL-соединение
curl_close($curl);
//преобразуем строку $ content в массив
return explode("\r\n", $content);
}
$hostname="http://www.php.net";
$out=get_content($hostname);
//выводим содержимое массива
```

```
echo "<pre>";
print_r($out);
echo "</pre>";
?>
```

Работа с доменами и IP-адресами

Все хосты в сети Интернет имеют уникальные IP-адреса, однако для удобства вместо них используются доменные имена, которые при помощи DNS-серверов привязываются к IP-адресам. В протоколе HTTP1.1 появилась возможность привязать к одному IP-адресу несколько доменных имен.

Функции для работы с IP-адресами и доменными именами:

Функция	Описание
checkdnsrr($hostname [, $type])	Находит на DNS-сервере записи ресурсов вида $type для хоста $hostname
gethostbyname($hostname)	Возвращает для доменного имени $hostname соответствующий ему IP-адрес
gethostbynamel($hostname)	Возвращает для доменного имени $hostname соответствующие ему IP-адреса
gethostbyaddr($ip_address)	Для IP-адреса $ip_address возвращает соответствующее ему доменное имя
ip2long($ip_address)	Преобразует IP-адрес $ip_address в целое число
long2ip($proper_address)	Преобразует целое число в IP-адрес $proper_address

Определение IP адреса по домену. Листинг 11.18

```
<?php
$hostname="http://sevidi.narod.ru";
$ip_address=gethostbyname($hostname);
echo "IP-адрес $hostname: $ip_address";
?>
```

Определение сетевого адреса (домена) по IP адресу. Листинг 11.19

```
<?php
$ip_address="127.0.0.1";
$hostname=gethostbyaddr($ip_address);
echo "Имя хоста с IP-адресом $ip_address: $hostname";
?>
```

Сервис Whois

Сервис Whois позволяет определить, на кого зарегистрирован тот или иной IP-адрес. Для этого достаточно послать IP-адрес по порту 43 соответствующего Whois-сервера.

Создание Whois-сервиса. Листинг 11.20

```
<?php
<center>
 <form method=post>
 <input type=text name=ip size=35>
 <input type=submit value='Введите IP-адрес'>
 </form>
</center>
<?php
if(!empty($_POST['ip'])) echo
whois("whois.arin.net",$_POST['ip']);

function whois($url,$ip)
{
  // Соединение с сокетом TCP, ожидающим на сервере
  // "whois.arin.net" по порту 43.
  // В результате возвращается дескриптор соединения $sock.
  $sock = fsockopen($url, 43, $errno, $errstr);
  if (!$sock) exit("$errno($errstr)");
  else
  {
    echo $url."<br>";
    // Записываем строку из переменной $_POST["ip"]
    // в дескриптор сокета.
    fputs ($sock, $ip."\r\n");
    // Осуществляем чтение из дескриптора сокета.
    $text = "";
    while (!feof($sock))
    {
      $text .= fgets ($sock, 128)."<br>";
    }
    // закрываем соединение
    fclose ($sock);

    // Ищем реферальный сервер
    $pattern = "|ReferralServer: whois://([^\n<:]+)|i";
    preg_match($pattern, $text, $out);
    if(!empty($out[1])) return whois($out[1], $ip);
    else return $text;
  }
} ?>
```

12. Объектно-ориентированное программирование.

Независимо от языка программирования объектно-ориентированный подход имеет ряд общих принципов, а именно:

- возможность создавать **абстрактные типы данных**, позволяющая наряду с предопределенными типами данных (такими как integer, bool, double, string) вводить свои собственные типы данных (классы) и

объявлять "переменные" таких типов данных (объекты) . Создавая свои собственные типы данных, программист оперирует не машинными терминами (переменная, функция), а объектами реального мира, поднимаясь тем самым на новый абстрактный уровень. Яблоки и людей нельзя умножать друг на друга, однако низкоуровневый код запросто позволит совершить такую логическую ошибку, тогда как при использовании абстрактных типов данных такая операция становится невозможной;

- **инкапсуляции**, допускающая взаимодействие пользователя с абстрактными типами данных только через их интерфейс и скрывающая внутреннюю реализацию объекта, не допуская влияния на его внутреннее состояние. Память человека ограничена и не может содержать все детали огромного проекта, тогда как использование инкапсуляции позволяет разработать объект и использовать его, не заботясь о внутренней реализации, прибегая только к небольшому числу интерфейсных методов;
- **наследование**, позволяющее развить существующий абстрактный тип данных - класс, создав на его основе новый касс. При этом новый класс автоматически получает возможности уже существующего абстрактного типа данных. Зачастую абстрактные типы данных слишком сложны, поэтому прибегают к их последовательной разработке, выстраивая иерархию классов от общего к частному;
- **полиморфизм**, допускающий построение целых цепочек и разветвленных деревьев наследующих друг другу абстрактных типов данных (классов). При этом весь набор классов будет иметь ряд методов с одинаковыми названиями: любой из классов данного дерева гарантированно обладает методом с таким именем. Этот принцип помогает автоматически обрабатывать массивы данных разного типа.

Полиморфизм

При использовании разветвленных наследственных иерархий все объекты производных классов автоматически снабжаются методами базового класса. Производные классы могут использовать методы базового класса без изменений, адаптировать их или заменять своей собственной реализацией. Как бы ни были реализованы эти методы, можно достоверно утверждать, что все объекты наследственной иерархии классов будут обладать некоторым количеством методов с одними и теми же названиями. Это явление называется полиморфизмом.

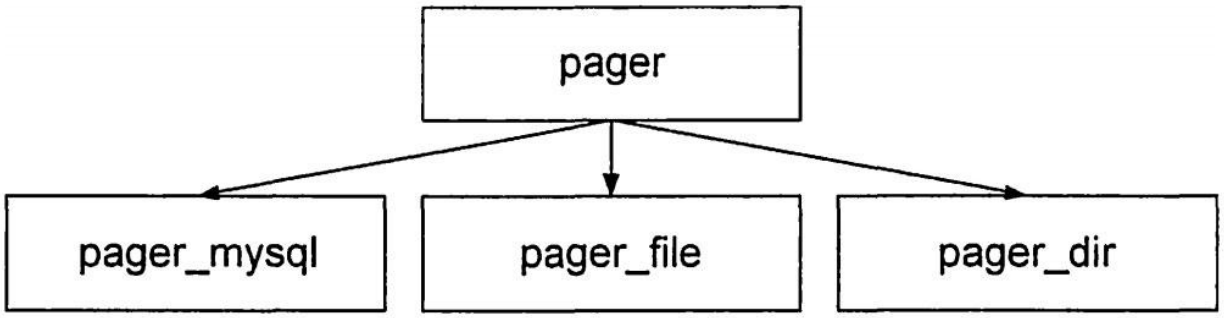
В качестве примера рассмотрим постраничную навигацию.

В Web-приложении источником списка элементов, к которому следует применить постраничную навигацию, могут выступать база данных, файл со строками, директория с изображениями. Каждый раз создавать отдельную функцию для реализации постраничной навигации не всегда удобно, так как придется дублировать значительную часть кода в нескольких функциях.

В данном случае удобнее реализовать постраничную навигацию в методе базового класса pager, а методы, работающие с конкретными источниками, переопределить в производных классах:

- ✓ pager_mysql -постраничная навигация дЛЯ СУБД MySQL;
- ✓ pager_file - постраничная навигация для текстового файла;
- ✓ pager_dir -постраничная навигация для файлов в директории.

Иерархия классов постраничной навигации представлена на рис. 2.1

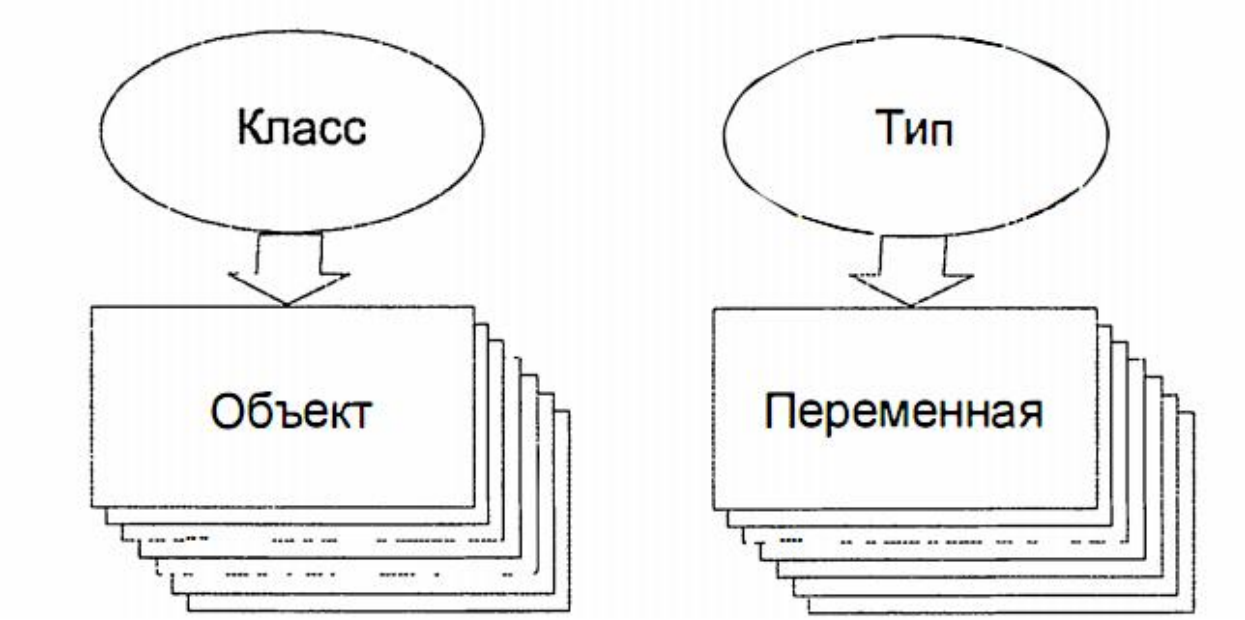


Класс pager() "не знает", каким образом производные классы будут узнавать общее количество элементов в списке, сколько элементов будет отображаться на одной странице, сколько ссылок находится слева и справа от текущей страницы. Поэтому помимо метода _tostring () класс будет содержать четыре защищенных (protected) метода, которые возвращают:

- o get _ total () -общее количество элементов в списке;
- o get_pnuer () -количество элементов на странице;
- o get_page_link () - количество элементов слева и справа от текущей страницы;

- ✓ O get parameters () - строку, которую необходимо передать по ссылкам на другую страницу (например, при постраничном выводе результатов поиска по ссылкам придется передавать результаты поиска) .

Эти методы не имеют реаизации в классе pager (пустые) и перегружаются производными классами, которые формируют параметры в зависимости от источника.



Переменные объявляются при помощи типа, объекты при помощи класса.

Создание класса

Касс объявляется при помощи ключевого слова class, после которого следует уникальное имя класса и тело класса в фигурных скобках. В теле класса объявляются переменные и функции класса, которые соответственно называются методами и членами. В листинге 1.1 при водится общий синтаксис объявления класса.

Создание класса. Листинг 12.1

```
<?php
class имя_класса
{
 //Члены и методы класса
}
?>
```

Важной особенностью PHP является то, что PHP-скрипты могут вкючаться в документ при помощи тегов <?php и ?>. Один документ может содержать множество включений этих тегов, одн. ако касс должен объявляться в одном неразрывном блоке <?php и ?>. Попытка разорвать объявление класса приводит к генерации интерпретатором ошибки разбора Parse error: parse erгог, unexpected ';', expecting T_FUNCTION.

Так как прерывать объявление класса недопустимо, его не удастся механически разбить и при помощи инструкций include (), include once ( ), require(), require_once(). Допускается, однако, использование этих конструкций внутри методов.

Создание объекта

Объект объявляется при помощи ключевого слова new, за которым следует имя класса. В листинге 1.2 создается пустой класс emp И объявляется объект $obj данного класса.

Создание класса emp и объявление объекта $obj данного класса. Листинг 12.2

```
<?php
class emp {}
 $obj = new emp;
?>
```

Объект – это обычная переменная.

Объект существует до конца времени выполнения скрипта или пока не будет уничтожен явно при помощи конструкции unset ()

Спецификаторы доступа

Открытые члены класса объявляются спецификатором доступа public и доступны как методам класса, так и внешнему по отношению к классу коду. Закрытые методы и члены класса объявляются при помощи спецификатора private и доступны только в рамках класса; обратиться к ним извне невозможно.

Создание открытых и закрытых членов класса. Класс employee. Листинг 12.3

```
<?php
class  employee
{
public  $surname ;
public  $name ;
public  $patronymic ;
private  $age ;
}
?>
```

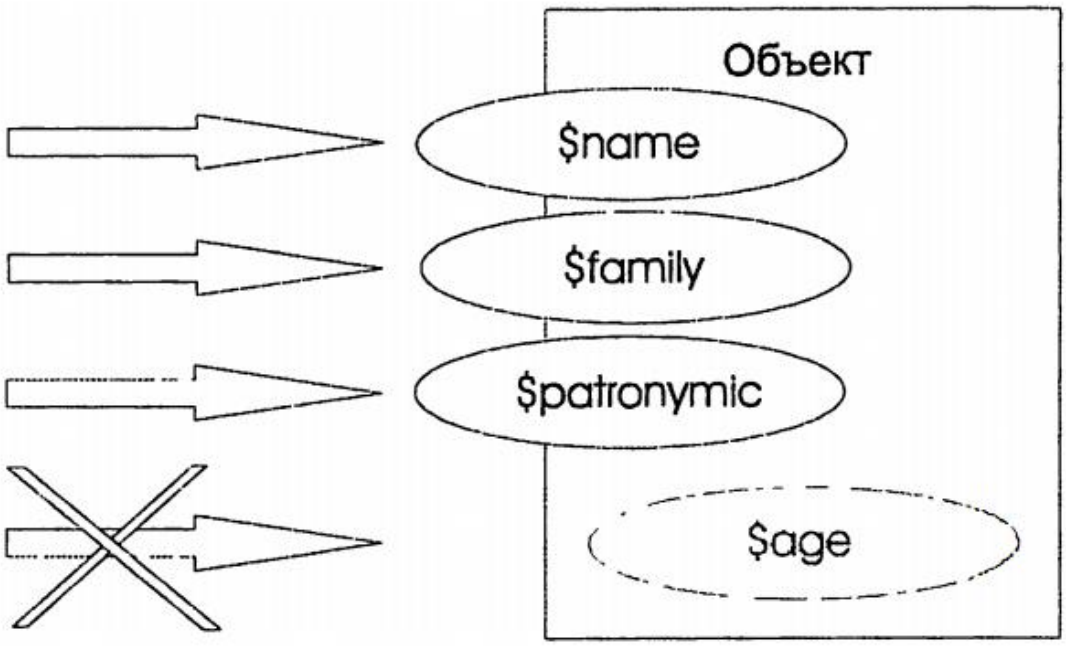
Рассмотрим пример. Пусть класс employee располагается в файле class.employee.php. В листинге 1.3 объявляется объект $emp класса employee, при этом члены класса получают необходимые значения и выводятся в окно браузера.

Обращение к членам класса employee. Листинг 12.4

```
<?php
//  Подключаем  объявление  класса
require_once ("class . employee .php" );
//  Объявляем  объект  касса  employee
$emp  =  new  employee () ;
//  Присваиваем  значения  членам  класса
$emp->surname     ="Борисов";
$emp->name        ="Игорь";
```

```
$emp->patronymic  ="Иванович" ;
//  $emp->age  =  23 ;  //  Ошбка
//  Вывод  члена  класса
echo  $emp->surname ."  ". $emp->name ."  ".$emp->patronymic .
"<br> "
?>
```

Как видно из листинга 1.4, при обращении к член касса используется оператор ->, после которого следует ввести имя этого члена. Заметьте, что при обращении к членам касса не добавляется символ $. Обращение к закрытому члену класса $age завершится ошибкой Fatal error: Cannot access private propert employee: :$age. На рис. 1.2 приводится схематическое изображение процесса доступа к членам объекта, снабженным разными спецификаторами доступа.



Методы класса

Популярность объектно-ориентированного подхода заключается в том, что классы представляют собой не просто контейнеры для хранения переменных (в качестве таких контейнеров в PHP могут выступать ассоциативные массивы), но также позволяют включать методы, обрабатывающие как открытые, так и закрытые члены касса.

Метод класса представляет собой обычную функцию PHP, которую предваряет один из спецификаторов доступа. В листинге 1.5 приводится пример класса class_mth, В состав которого входит метод show_message () . Задача метода сводится к выводу в окно браузера сообщения Hello world!.

Объявление метода класса. Листинг 12.5

```
<?php
  class cls_mth
  {
    public function show_message()
    {
      echo "Hello world!";
    }
  }

  $obj = new cls_mth();
  $obj->show_message();
?>
```

Для обращения к любому элементу класса внутри метода следует применять конструкцию $this-> . Член $this, который неявно присутствует в каждом кассе, является ссылкой на текущий объект касса.

Вернемся к классу employee, определенному в листинге 1 .3. Класс содержит закрытую переменную $age, определяющую возраст сотрудника. Создадим четыре метода:

- get_age() -метод, возвращающий значение закрытого члена $age;
- set_age() - метод, возвращающий значение закрытого члена $age и проверяющий его принадлежность к. числовому типу данных и промежутку от 18 до 65;
- get_info () -метод, возвращающий фамилию и имя сотрудника;
- get_full_info() - метод, возвращающий фамилию, имя и возраст сотрудника.

Использование открытых методов для доступа к закрытому члену касса. Листинг 12.6

```
<?php
  class employee
  {
    // Открытые члены
    public $surname;
    public $name;
    public $patronymic;

    // Открытые методы
    public function get_age()
    {
      return $this->age;
    }
    public function set_age($val)
    {
      $val = intval($val);
```

```
      if($val >= 18 && $val <= 65)
      {
         $this->age = $val;
         return true;
      }
      else return false;
    }
    public function get_info()
    {
      return $this->surname." ".$this->name." ".$this-
>patronymic;
    }
    public function get_full_info()
    {
      return "{$this->get_info()} ({$this->get_age()})";
    }

    // Закрытые члены
    private $age;
  }
?>
```

- Метод get_ age ( ) не принимает ни одного параметра -единственная его задача обратиться к закрытому члену $age и вернуть это значение вызвавшему его коду. На первый взгляд, такой подход может по казаться избыточным - вместо непосредственного обращения к члену $age вводится функция-посредник, на вызов которой затрачиваются лишние ресурсы. Однако лишние затраты на проектирование методов доступа к закрытым членам, а также вызов функций вместо прямого обращения с лихвой окупаются в дальнейшем. Например, для вычисления текущего возраста сотрудника на основании сведений из базы данных, актуальных на момент заполнения анкеты (вероятно, несколько лет назад), не придется менять весь код, использующий класс employee: достаточно изменить внутреннюю реализацию метода get_age ( ), после чего изменения автоматически отразятся на всей системе.
- Метод set_age () из листинга 1.6 принимает единственный параметр $val, который при помощи функции intval () приводится к числовому типу. После этого осуществляется проверка, принадлежит ли параметр $val интервалу от 18 до 65. Если параметр удовлетворяет этому условию, закрытый член класса $this ->age получает новое значение, а метод возвращает значение true, свидетельствующее об успешном выполнении операции. Если новое значение не удовлетворяет условию, установленному для члена $this->age, метод set_age

() возвращает значение false, говорящее о неудачном завершении операции.

- Следует обратить внимание на формирование строк в методах get_info () И get_full_info (). Строки в РН объединяются при помощи символа точки (.), при этом все типы, отличные от строкового, приводятся К нему автоматически. Однако это не единственный способ формирования строки: можно прибегнуть к так называемой интерполяции, при которой переменная вставляется в строку, заключенную в двойные кавычки. Например, переменная $name = "Hello", подставленная в строку "$name world ! ", при водит К формированию строки "Hello world !" .

Использование касса exployee. Листинг 12.7

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee;

  // Передаем значения членам класса
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";
  if(!$emp->set_age(23)) exit("Ошибка вычисления возраста");

  echo $emp->get_full_info(); // Борисов Игорь Иванович (23)
?>
```

Дамп объекта

При отладке объектно-ориентированных приложений зачастую требуется проанализировать текущее состояние объекта. Для этого предусмотрена функция print_r (). Для удобства восприятия вызов функции print_r () обрамляется HTML-тегами <pre> и </pre>, которые сохраняют структуру переносов и отступов при отображении результата в браузере.

Получение структуры объекта касса employee. Листинг 12.8

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";
  $emp->set_age(23);
```

```
  // Выводим структуру объекта
  echo "<pre>";
  print_r($emp);
  echo "</pre>";
?>
```

Результатом работы скрипта из листинга 1.8 являются следующие строки:

eployee Object

(

[surnae] => Борисов

[пае] => Игорь

[patronyc] => Иванович

[age : private] => 23

)

Таким образом, метод возвращает все члены объекта, в том числе и закрытые. Закрытые члены помечаются спецификатором private.

Специальные методы класса

Помимо методов, создаваемых разработчиком класса, объектно-ориентированная модель PHP предоставляет разработчику специальные методы.

Метод	Описание
__construct()	Конструктор класса; метод, который автоматически выполняется в момент создания объекта до вызова всех остальных методов класса.
__destruct()	Деструктор класса; метод, который автоматически выполняется в момент уничтожения объекта
__autoload()	Перегружаемая функция, не являющаяся методом класса. Позволяет автоматически загружать класс при попытке создания его объекта
__set()	Аксессор; метод, предназначенный для установки значению свойству
__get()	Аксессор; метод, предназначенный для чтения свойства
__isset()	Позволяет задать логику проверки существования свойства при помощи конструкции isset()
__unset()	Позволяет задать логику удаления свойства при помощи конструкции unset ()
__call()	Позволяет задать динамический метод
__tostring()	Позволяет интерполировать (подставлять) объект в строку
__set_state()	Позволяет осуществить экспорт объекта

__clone ()	Позволяет управлять клонированием объека
__sleep ()	Позволяет управлять поведением объекта при его сериализации при помощи функции serialize ()
__wakeup()	Позволяет управлять поведением объекта при его восстановлении из сериализованного состояния при помощи функции unserialize ()

Функции для работы с методами и классами

Функция	Описание
class_exists($class_name)	Возвращает true, если класс с именем $class_name объявлен, и false - в противном случае
get_class_methods($class_name)	Возвращает массив с именами методов класса $class_name
get_class_vars ($class_name)	Возвращает массив с именами и значения членов касса $class_name
get_class($obj)	Возвращает имя касса, которому принадлежит объект $obj
get_declared_classes()	Возвращает массив с именами объявленных классов
get_object_vars($obj)	Возвращает массив с именами и значениями членов объекта $obj
и др…	

Унифицированные методы: конструктор __construct() и деструктор _destruct()

Конструктор - это специальный метод класса, который автоматически выполняется *в момент создания объекта до вызова всех остальных методов класса*. Данный метод используется главным образом для инициализации объекта, обеспечивая согласованность его членов. Для объявления конструктора в классе необходимо создать метод с именем _construct().

Деструктор - это специальный метод класса, который автоматически выполняется в момент уничтожения объекта. Данный метод вызывается всегда самым последним и используется главным образом для корректного освобождения зарезервированных конструктором ресурсов. Для объявления деструкора в классе необходимо создать метод с именем __destruct().

Класс с использованием конструктора. Листинг 12.9

```
<?php
  class cls
  {
    public function __construct()
    {
      echo "Вызов конструктора<br>";
    }
    public function print_msg()
```

```
    {
      echo "Вызов метода<br>";
    }
    public function __destruct()
    {
      echo "Вызов деструктора<br>";
    }
  }
?>
```

Реализация класса с конструктором. Листинг 12.10

```
<?php
  require_once("class.point.php");

  $obj = new cls();
  $obj->print_msg();
  echo "Произвольный текст<br>";
?>
```

Обратите внимание на то, что создав объект, автоматически вызываем два унифицированных метода: конструктор и деструктор. Причем, конструктор создастся до вывода любого другого метода класса, а деструктор – последним.

Чтобы вызвать конструктор, объявленный в родительском классе, следует обратиться к методу **parent::__construct()**.

Реализация класса с конструктором. Листинг 12.11

```
<?php
class BaseClass {
     function __construct() {
         print "Конструктор класса BaseClass\n";
     }
}

class SubClass extends BaseClass {
     function __construct() {
         parent::__construct();
         print "Конструктор класса SubClass\n";
     }
}

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

Автозагрузка классов, функция __autoload()

Обычно класс оформляется в виде отдельного файла, который вставляется в нужном месте при помощи конструкции require_once (). Если используется большое количество классов, то в начале скрипта выстраивается целая вереница конструкций require_once (), что может быть не очень удобно, особенно если путь к кассам приходится часто изменять. Начиная с PHP 5,

разработчику предоставляется специальная функция _autoload (), которая позволяет задать путь директории с классами и автоматически подключать кассы при обращении к ним в теле программы. Данная функция принимает в качестве единственного параметра имя касса (листинг 2.1).

Создание класса. Листинг 12.12

```
<?php
//  Функция автозагрузок классов
function __autoload ($classnae)
{
 reqire_once ("class/class.$classnae.php" );
}
…
?>
```

Аксессоры. Методы __set() и __get()

Неписаным правилам объектно-ориентированного программирования является использование только закрытых членов, доступ к которым осуществляется через открытые методы касса. Это позволяет скрыть внутреннюю реализацию касса, ограничить диапазон значений, которые можно присваивать члену, и сделать член доступным только для чтения.

Неудобство заключается в том, что для каждого из членов приходится создавать отдельный метод для чтения и присваивания нового значения, имена которых зачастую не совпадают с именами членов.

Выходом из ситуации является использование свойств, обращение к которым выглядит точно так же, как к открытым членам класса. Для их реализации необходимо перегрузить специальные методы _get () И _set (), которые часто называют аксессоры. Метод _get ( ), предназначенный для чтения свойства, принимает единственный параметр, который служит ключом. Метод __set() позволяет присвоить свойству новое значение и принимает два параметра, первый из которых является ключом, а второй - значением свойства.

В примере из листинга 2.2 при помощи метода __set () объект присваиваются новые свойства, которые помещаются в массив $this ->arr, а перегруженный метод __get() позволяет извлечь их из массива.

Использование методов __set() и __get(). Листинг 12.13

```
<?php
  class cls
  {
    private $arr = array();

    private function __get($index)
    {
      return $this->arr[$index];
    }
```

```
    private function __set($index, $value)
    {
      $this->arr[$index] = $value;
    }
  }
?>
```

Обращение к несуществующему элементу $name. Листинг 12.14

```
<?php
  require_once("class.cls.php");

  $obj = new cls();
  $obj->name = "Hello world!<br>";
  echo $obj->name;
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Основная идея этих методов заключается в том, что мы внешним кодом можем реализовать переменные внутри класса. get() – получили внешнюю переменную, __set() – обработали.

Проверка существования членов класса. Метод __isset()

Убедиться в существовании члена из внешнего кода можно при помощи конструкции __isset(). Который принимает в качестве единственного параметра имя свойства и возвращает true, если свойство с таким именем существует, и false - в противном случае.

Использование метода __isset(). Листинг 12.15

```
<?php
…
private  function _isset ($index)
{
return  isset ($this->$ index) ;
}
…
?>
```

Уничтожение члена класса. Метод __unset()

Для уничтожения членов класса служит специальный метод _unset ().

Использование метода __unset(). Листинг 12.16

```
<?php
  class cls
  {
    private $arr = array();

    public function __get($index)
    {
```

```
      return $this->arr[$index];
    }

    public function __set($index, $value)
    {
      $this->arr[$index] = $value;
    }

    public function __isset($index)
    {
      return isset($this->arr[$index]);
    }

    public function __unset($index)
    {
      unset($this->arr[$index]);
    }
 }
?>
```

Использование конструкции unset(). Листинг 12.17

```
<?php
  require_once("class.cls.php");

  $obj = new cls();
  $obj->name = "Новое свойство класса";

  echo "<pre>";
  print_r($obj);
  echo "</pre>";

  unset($obj->name);

  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Интерполяция объекта. Метод __tostring()

Специаьный метод __tostring() позволяет интерполировать (подставлять) объект в строку. Для подстановки значений переменных необходимо строку в двойные кавычки.

Использование метода __tostring(). Листинг 12.18

```
<?php
  class employee
  {
    public function __construct($surname, $name, $patronymic,
$age = 18)
    {
      $this->surname = $surname;
      $this->name = $name;
      $this->patronymic = $patronymic;
      $this->age = $age;
    }
```

```
    public function __toString()
    {
      return "{$this->surname} {$this->name[0]}.{$this-
>patronymic[0]}";
    }

    public $surname;
    public $name;
    private $patronymic;
  }
?>
```

Интерполяция объекта. Листинг 12.19

```
<?php
  require_once("class.employee.php");

  $obj = new employee("Борисов", "Игорь", "Иванович");
  echo "Сотрудник $obj недавно принят на работу";
?>
```

Наследование и спецификаторы доступа

Одной из главных целей объектно-ориентированного подхода является повторное использование кода. Иногда сложно определить, требуется ли такой подход в решении поставленной задачи или нет. Однако если созданный класс используется лишь однажды в одном приложении, можно утверждать, что его создание - пустая трата времени. Объектно-ориентированное программирование значительно снижает читабельность и эффективность, поэтом окупает себя только при интенсивном использовании наследования.

Наследование позволяет создать новый класс на основе уже существующего, автоматически включив в новый касс все члены и методы старого. В рамках наследования "старый" класс называется *базовым*, а вновь создаваемый класс - *производным*. При объявлении производного класса необходимо указать имя базового касса с помощью ключевого слова extends .

Члены и методы, объявленные со спецификаторами доступа public и private, при наследовании ведут себя по отношению к производному классу точно так же, как и по отношению к внешней программе. Это означает, что из производного касса доступны все методы и члены, объявленные со спецификатором доступа public, и не доступны компоненты, объявленные как private.

Иногда удобно, чтобы член или метод базового класса, оставаясь закрытым для внешнего кода, был открыт для производного класса. В этом случае прибегают к специальному спецификатору доступа protected.

Запрет создания объектов базового класса. Листинг 12.20

```
<?php
  class base
  {
    private $var;
    protected function __construct($var)
    {
      $this->var = $var;
    }
  }
  class derived extends base
  {
    public function __construct($var)
    {
      parent::__construct($var);
    }
  }

  // $obj = new base(20); // Ошибка
  $obj = new derived(20);
  echo "<pre>";
  print_r($obj);
  echo "</pre>";
?>
```

Этим приемом часто пользуются, когда объект базового касса абстрактен и скрывает в себе функциональность, необходимого для производных классов, а сам по себе не имеет смысла.

Перегрузка методов

В производном классе можно создать метод с таким же названием, что и в базовом классе, который заменит метод базового класса при вызове. Такая процедура называется перегрузкой методов.

Перегрузка метода print_var(). Листинг 12.21

```
<?php
  class base
  {
    public function print_var()
    {
      echo "Вызов метода print_var() базового класса<br>";
    }
  }
  class derived extends base
  {
    public function print_var()
    {
      echo "Вызов метода print_var() производного класса<br>";
    }
  }

  $obj = new derived();
  $obj->print_var();
?>
```

Таким обраом, метод base : : print _ var () не вызывается при обращении к объекту производного касса derived.

Однако в рамках производного касса остается возможность вызвать метод базового класса, обратившись к нему при помощи префикса parent::. В листинге 2.11 представлена перегрузка метода print_ var( ), при которой метод производного класса сначала вызывает метод базового класса.

Обращение к методу базового класса. Листинг 12.22

```
<?php
  class base
  {
    public function print_var()
    {
      echo "Вызов метода print_var() базового класса<br>";
    }
  }
  class derived extends base
  {
    public function print_var()
    {
      parent::print_var();
      echo "Вызов метода print_var() производного класса<br>";
    }
  }

  $obj = new derived();
  $obj->print_var();
?>
```

Абстрактные классы

Объект класса pager не имеет смысла и не может быть использован по назначению - работают только его наследники. Для того чтобы предотвратить возможность создания объекта такого класса, PHP предоставляет специальный инструмент - класс можно объявить абстрактным. Для этого объявление касса предваряют ключевым словом abstract.

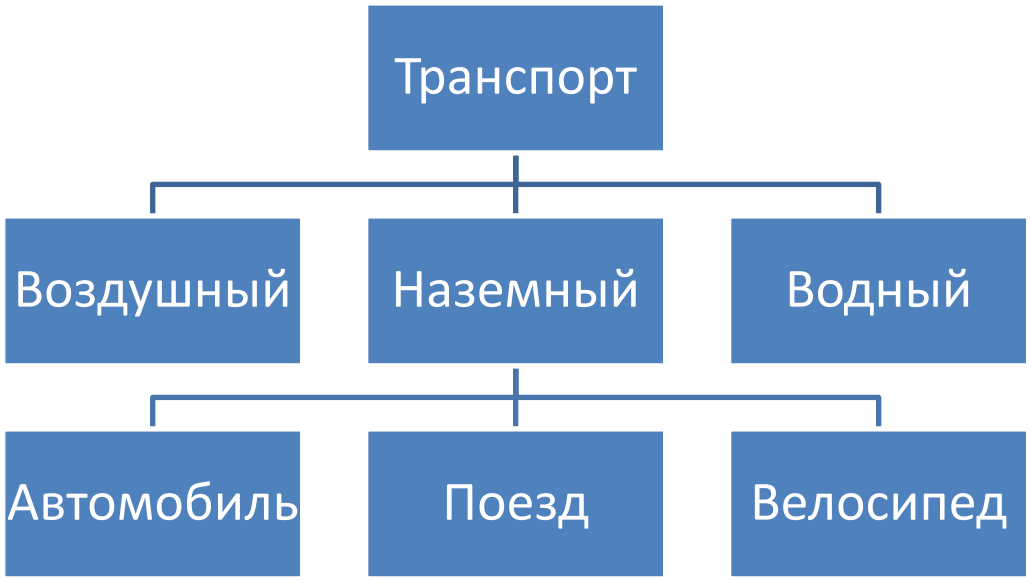
Объявление абстрактного класса. Листинг 12.23

```
<?php
  abstract class base
  {
    …
  }
?>
```

Теперь попытка создать экземпляр класса pager (листинг 11.23) будет заканчиваться сообщением об ошибке Fatal error: Cannot instantiate abstract class pager (Невозможно объявить объект абстрактного класса) .

Абстрактными следует объявлять классы, которые не существуют в реальности и объекты которых заведомо не потребуются.

Пример:



Где транспорт воздушный, наземный и водный являются абстрактными классами. А автомобиль, поезд и велосипед – конкретными.

Абстрактные методы

Напомним, что в базовом классе pager находятся только заглушки методов get_total(), get_pnumber(), get_page_link() и get_parameters() .

Заглушками в программировании называют методы, которые не выполняют никакой работы.

При помощи абстрактных методов исключается возможность неумышленного нарушения полиморфизма для производных классов.

Объявление абстрактного класса и методов. Листинг 12.24

```
<?php
  abstract class base
  {
abstract  fuction  get_total() ;
abstract  function  get-nuer() ;
abstract  function  get-age_link() ;
abstract  function  get_paraeters() ;
  }
?>
```

Наличие в классе абстрактного метода требует, чтобы класс также был объявлен абстрактным.

13. Обработка ошибок.

Синтаксис исключений.

Для реализации механизма исключений в PHP введены следующие ключевые слова: try (контролировать), throw (генерировать) и catch (обрабатывать).

В случае возникновения непредвиденных ситуаций внутри блока try можно выполнить так называемую генерацию (throwing) исключения. Ключевое слово throw запускает механизм обработки исключений. Оно представляет собой конструкцию языка (не функцию) с двумя входящими параметрами: сообщение ошибки и код. Оба параметра необязательны.

Наконец, за блоком try должен следовать как минимум один блок catch.

Создание контролируемого блока. Листинг 13.1

```
<?php
  try
  {
    //операторы
    ...
    //генерация исключений
    throw  Выражение_генерации_искчения;
    //операторы
    ...
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
  }
?>
```

С одним блоком try может быть связано несколько блоков catch. Использование нескольких блоков catch оправдано, если каждый из них ожидает перехвата отдельного типа исключения

Класс Exception является предопределенным и не требует объявления.

Передача сообщения и кода из точки генерации исключения. Листинг 13.2

```
<?php
  try
  {
    $code = rand(0,1);
    if(!$code)
    {
      // Генерация исключений
      throw new Exception("Первая точка входа", $code);
    }
    else
    {
      // Генерация исключений
      throw new Exception("Вторая точка входа", $code);
```

```
    }
  }
  catch(Exception $exp)
  {
    // Блок обработки исключительной ситуации
    echo "Исключение {$exp->getCode()} : {$exp-
getMessage()}<br>";
    echo "в файле {$exp->getFile()}<br>";
    echo "в строке {$exp->getLine()}<br>";
  }
?>
```

Интерфейс класса Exception

Если бы класс реализовывася внешним разработчиком, то он мог бы выглядеть так, как это представлено в листинге 4.7

Гипотетическая реализация касса Exception. Листинг 13.3

```
<?php
class Expection
{
//  Текстовое  сообщение,  описывающее
//  исключительную  ситуацию
protected  $message  =  'Unknown  exception ';
//  ЧИСЛОВОЙ  код,  назначенный  данному
//  типу  исключительных  ситуаций
protected  $code  =  0;
// и  файла , в  котором  произошла
//  исключительная  ситуация
protected  $file ;
//  Номер  строк, в которой произошла
/ /  исключительная ситуация
protected  $line;
//  Конструктор  класса,  инициализирующий
//  члены  $message  и  $code  (оба параметра
//не  являются  обязательными)
public  function __construct ($message = null, $code = 0);
//  Метод,  возвращающий  текстовое
//  сообщение  в  члене  $this->message
public  final  function  getMessage() ;
//  Метод,  возвращающий  числовой  код
//  $this->code ,  характеризующий
//  исключительную  ситуацию
public  final  function  getCode() ;
//  Метод,  возвращающий  имя  файла
//  $this->file,  в  котором  произошла
//  исключительная ситуация
public  final  function  getFile() ;
//  Метод,  возвращающий  номер  строк
//  $this->line , в  которой  произошла
//  исключительная  ситуация
public  final  function  getLine() ;
//  Стек  обработки  исключительной
//  ситуации  в  виде  массива
public  final  function  getTrace();
//  Стек обработки  исключительной
```

```
//  ситуации в виде  строки
null,  $code
public  final  function  getTraceAs String() ;
//  Пере  грузка  метода __toString() ,
//  возвращающего  строку при  использовании
//  объекта  в  строковом  контексте
public  function __toString() ;
}
?>
```

Как видно из листинга, большинство общедоступных методов являются финальными или защещенными.

Защищенные члены класса Exception:

Член	Описание
`$this->message`	Текстовое сообщение, описывающее исключительную ситуацию
`$this->code`	Числовой код, назначенный данному типу исключительных ситуаций
`$this->file`	Имя файла, в котором произошла исключительная ситуация
`$this->line`	Номер строки файла `$this->file`, в которой произошла исключительная ситуация

Эти методы нельзя переопределить. Но можно создать собственных подкласс Exceptions.

Определение пользовательского класса. Листинг 13.4

```
<?php
  class myException extends Exception {
  function __toString() {
    return "<table border=\"\"><tr><td><strong>Исключение " .
      $this->getCode()."</strong>: ".$this->getMessage()."<br
/>" .
      " в ".$this->getFile()., строка ".$this->getLine() .
      "</td></tr></table><br />";
   }
}

try {
  throw new myException("Произошла очень серьезная ошибка", 42);
}
catch (myException $m) {
   echo $m;
}
?>
```

В этом коде объявляется новый класс исключения с именем myException, расширяющий базовый класс Exception. Различие между этими классами состоит в переопределении метода __toString() для обеспечения более изящного вывода сообщения об исключительной ситуации.

В следующем примере мы рассмотрим способы создания различных исключений, связанных с различными категориями ошибок записи и вывода содержимого текстового файла на экран.

Исключения, связанные с файловым вводом-выводом. Листинг 13.5

```
<?php
  class myException extends Exception {
class fileOpenException extends Exception {
  function __toString() {
    return "Ошибка файла fileOpenException ".$this->getCode().":
" . $this->getMessage()."<br />в ".$this->getFile() .
           ", строка ".$this->getLine()."<br />";
   }
}

class fileWriteException extends Exception {
  function __toString() {
    return "Ошибка файла fileWriteException ".$this-
>getCode().":" .
           $this->getMessage()."<br />в ".$this->getFile() .
           ", строка ".$this->getLine()."<br />";
   }
}

class fileLockException extends Exception {
  function __toString() {
    return "Ошибка файлаfileLockException ".$this->getCode().":"
. $this->getMessage()."<br />в ".$this->getFile() .
           ", строка ".$this->getLine()."<br />";
   }
}?>
```

Эти подклассы не выполняют никаких действий, представляющих особый интерес, они лишь переопределяют способ вывода ошибок на экран.

Далее рассмотрим сценарий обработки заказов с добавленной обработкой исключительных ситуаций.

Использование собственных подклассов в обработке исключений. Листинг 13.6

```
<?php
  require_once("file_exceptions.php");

  // создание коротких имен переменных
  $tireqty = $_POST['tireqty'];
  $oilqty = $_POST['oilqty'];
  $sparkqty = $_POST['sparkqty'];
  $address = $_POST['address'];
```

```
  $DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
?>

<html>
<head>
  <title>Автозапчасти от Вована - Результаты заказа</title>
</head>
<body>
<h1>Автозапчасти от Вована</h1>
<h2>Результаты заказа</h2>

<?php

$date = date('H:i, jS F');
echo "<p>Заказ обработан в ".$date."</p>";

$totalqty = $tireqty + $oilqty + $sparkqty;
echo "Заказано товаров: ".$totalqty."<br />";

if($totalqty == 0) {
  echo "Вы ничего не заказали на предыдущей странице!<br />";
} else {
  if ($tireqty>0) {
    echo $tireqty." покрышек<br />";
  }
  if ($oilqty>0) {
    echo $oilqty." бутылок масла<br />";
  }
  if ( $sparkqty>0 ) {
    echo $sparkqty." свечей зажигания<br />";
  }
}

define('TIREPRICE', 100);
define('OILPRICE', 10);
define('SPARKPRICE', 4);
$totalamount = $tireqty * TIREPRICE
             + $oilqty * OILPRICE
             + $sparkqty * SPARKPRICE;
$totalamount = number_format($totalamount, 2, '.', ' ');

echo "<p>Итого по заказу: ".$totalamount."</p>";
echo "<p>Адрес доставки: ".$address."</p>";

$outputstring = $date . "\t" . $tireqty . " покрышек\t"
  . $oilqty . " бутылок масла\t" . $sparkqty . " свечей зажига-
ния\t\$"
  . $totalamount . "\t". $address . "\n";

// открываем файл для дозаписи
try {
  if (!($fp = @fopen("$DOCUMENT_ROOT/../orders/orders.txt",
'ab')))
    throw new fileOpenException();
  if (!flock($fp, LOCK_EX))
    throw new fileLockException();
  if (!fwrite($fp, $outputstring, strlen($outputstring)))
```

```
      throw new fileWriteException();

  flock($fp, LOCK_UN);
  fclose($fp);
  echo "<p>Заказ записан.</p>";
}
catch (fileOpenException $foe) {
  echo "<p><strong>Невозможно открыть файл заказов."
    . " Обратитесь к Web-мастеру.</strong></p>";
}
catch (Exception $e) {
  echo "<p><strong>В данный момент мы не можем обработать ваш
заказ."
    . " Попробуйте повторить его позже.</strong></p>";
}

?>

</body>
</html>
```

Уровни выдачи сообщений об ошибках

PHP позволяет устанавливать степень тщательности обработки ошибок. Можно задавать типы событий, генерирующие сообщения. По умолчанию PHP выводит сообщения обо всех ошибках, которые не являются уведомлениями.

Для установки уровня сообщений используется набор предопределенных констант, перечисленных в таблице.

Каждая константа представляет тип ошибок, генерирующих сообщение либо игнорируемых. Например, если указать уровень ошибок E_ERROR, будут выводиться сообщения только о неисправимых ошибках. Допускается объединение констант методами двоичной арифметики для получения различных уровней сообщений об ошибках.

Устанавливаемый по умолчанию уровень (все сообщения кроме уведомлений) указывается следующим образом:

E_ALL & ~E_NOTICE

Использование установленного уровня вывода уведомлений. Листинг 13.7

```
<?php
  error_reporting(E_ALL & ~E_NOTICE);

…php код…
?>
```

Это выражение включает две предопределенных константы, объединенных с помощью операторов побитовых арифметических действий. Амперсанд (&) задает битовую операцию AND, а тильда (~) — битовую операцию NOT. Выражение можно прочитать так: E ALL AND NOT E NOTICE.

Таблица. Константы, определяющие уровень сообщений об ошибках

Значение	Имя	Описание
1	E_ERROR	Сообщения о неисправимых ошибках времени выполнения
2	E_WARNING	Сообщения об исправимых ошибках времени выполнения
4	E_PARSE	Сообщения об ошибках синтактисического анализатора
8	E_NOTICE	Уведомления и предупреждения, что выполненные действия могут быть ошибочными
16	E_CODE_ERROR	Сообщения о сбоях запуска механизма PHP
32	E_CODE_WARNING	Сообщения об исправимых ошибках в процессе запуска механизма PHP
64	E_COMPILE_ERROR	Сообщения об ошибках компиляции
128	E_COMPILE_WARNING	Сообщения об исправимых ошибках компиляции
256	E_USER_ERROR	Сообщения о сгенерированных пользователем ошибках
512	E_USER_WARNING	Сообщения о сгенерированных пользователем предупреждениях
1024	E_USER_NOTICE	Сообщения о сгенерированных пользователем уведомлениях
2047	E_ALL	Сообщения обо всех ошибках и предупреждениях
2048	E_STRICT	Сообщения об использовании устаревших не рекомендованных функциях; не включено в E_ALL, однако исключительно полезно при просмотре кода

Константа E_ALL представляет собой комбинацию всех типов ошибок. Ее можно заменить, связав все остальные константы битовыми операциями ИЛИ (OR,|):

E_ERROR | E_WARNING | EJPARSE | E_NOTICE | E_CORE_ERROR | E_CORE_KARNING | E_COMPILE_ERROR | E_COMPILE_WARNING | E_OSER_ERROR | E_USER_WARNING | E_USER_NOTICE

Подобным же образом реализуется устанавливаемый по умолчанию уровень сообщений за исключением того, что отсутствует уровень уведомлений (константа E_NOTICE):

E_ERROR | E_WARNING | E_PARSE | E_CORE_ERROR | E_CORE_WARNING | E_COMPILE_ERROR | E_COMPILE_WARNING | E_USER_ERROR | E_OSER WARNING | E USER NOTICE

14. PHP PDO. Трехуровневая архитектура. Логика данных. Логика приложения.

Трехуровневая архитектура

Трехуровневая архитектура подразумевает разделение кода на три уровня:

- **Уровень представления**. Содержит код, отвечающий за интерфейс приложения (как правило, html выполняемые и подключаемые яваскрипты, флэш)
- **Уровень логики приложения**. Задача этого уровня получать запросы от уровня представления и возвращать обратно ответы, используя уровень данных. Почти каждое событие, происходящее на уровне представления, приводит к обращению к уровню логики приложения, за исключением событий, с которыми уровень представления может разобраться сам (например, провести проверку введенных пользователем данных). Если, к примеру, пользователь обращается к механизму поиска товара в каталоге, уровень представления обращается к уровню логики приложения и требует сообщить, какие товары соответствуют критериям поиска. Далее уровень логики приложения должен обращаться к уровню данных. Получив данные, уровень логики приложения передает их уровню представления.
- **Уровень данных**. Уровень базы данных отвечает за управление данными и передачу этих данных уровню логики приложения.

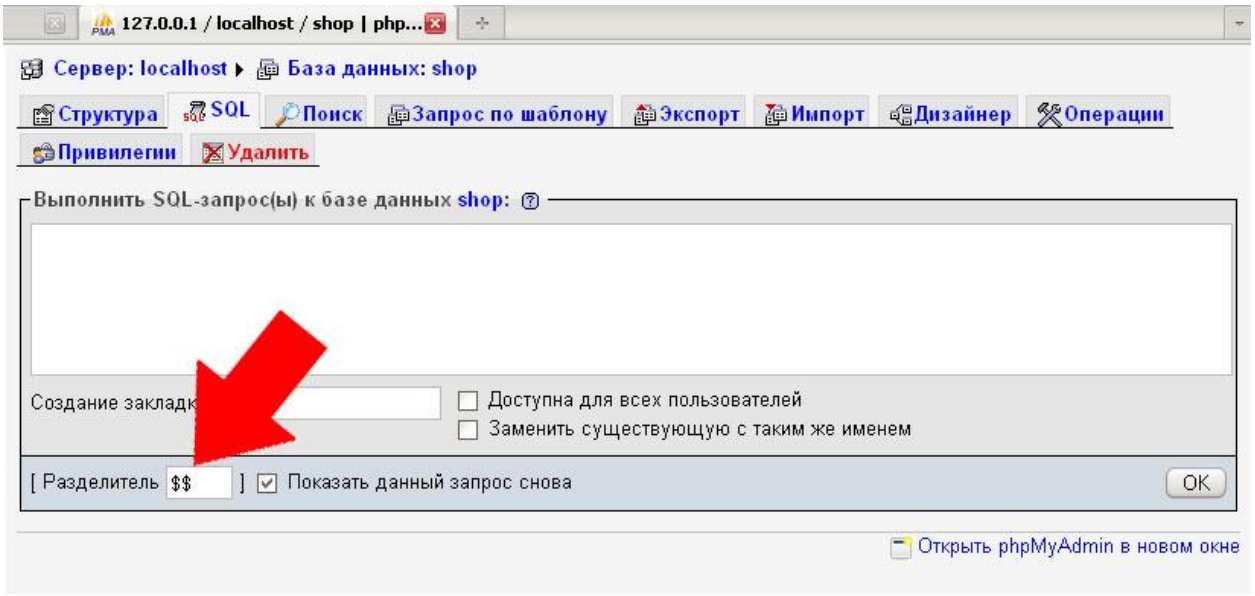
Сами по себе уровни являются сугубо логическими единицами. Код, относящийся к разным уровням, не обязательно должен физически размещаться в разных файлах. Но для удобства дальнейшего сопровождения проекта, необходимо к этому стремится.

Трехуровневая архитектура, которую мы сейчас рассматриваем, - это частный (хотя и наиболее популярный) вариант архитектуры приложения. На практике мы ранее сталкивались с двухуровневой архитектурой (“клиент/сервер”). Так же возможна более сложная архитектура, что позволяет сделать приложение еще более гибким и масштабируемым, однако тогда придется потратить гораздо больше времени на проектирование.

Логика данных. Процедуры MySQL

Процедуры – это набор операторов SQL на стороне MySQL-сервера, сохраняемые под определенным именем, по которому этот набор можно вызывать. Хранимые процедуры, аналогично функциям PHP могут принимать аргументы и возвращать значения.

В phpMyAdmin процедуры можно создавать в поле для вставки SQL-запросов. При этом желательно использовать уникальный разделитель:



Причем, при создании процедуры не обязательно использовать символы $$ (как на изображении), главное, чтобы огранитель отличался от использования по умолчанию точки с запятой, и чтобы такого символа не встречалось внутри самой процеудыр.

Итак, создадим процедуру с именем catalog_get_departments_list()

Создание хранимой процедуры. Листинг 14.1

```
CREATE PROCEDURE catalog_get_departments_list()
BEGIN
  SELECT department_id, name FROM department ORDER BY depart-
ment_id;
END$$
```

В первой строчке мы задаем имя процедуры. Тело процедуры помещаем между строками BEGIN и END$$. Все что делает данная процедура – это выполняет оператор SELECT и возвращает результаты выполнения.

Подготовительные запросы (prepared statement)

Подготовленные запросы имеются в MySQL начиная с 4.1 и нужны по трём причинам.

1. Скорость. Если вы выполняете однообразные запросы, то mysql парсеру каждый раз приходится выполнять распознавание - какой тип запроса, какие данные передаются и тому подобное. Если сделать прототип запроса, а в последствии передавать только данные, то ясное дело что это скажется на скорости.
2. Передача данных. Мало того что для подобных запросов передача бинарных данных не нуждается в конвертировании в строку, так и то что стандартное ограничение на один запрос имеет около 1 мб можно подвинуть благодаря тому что каждый кусок данных передаётся отдельным запросом, а не одним общим (INSERT скажем).
3. Безопасность. Благодаря отделению данных от запроса вы можете обезопасить себя от SQL injection.

Процедура с использованием подготовительного параметра (prepared statement). Листинг 14.2

```
<?php
CREATE PROCEDURE catalog_get_products_on_catalog(
  IN inShortProductDescriptionLength INT,
  IN inProductsPerPage INT, IN inStartItem INT)
BEGIN
  PREPARE statement FROM
    "SELECT   product_id, name,
              IF(LENGTH(description) <= ?,
                 description,
                 CONCAT(LEFT(description, ?),
                        '...')) AS description,
              price, discounted_price, thumbnail
     FROM     product
     WHERE    display = 1 OR display = 3
     ORDER BY display DESC
     LIMIT    ?, ?";

  SET @p1 = inShortProductDescriptionLength;
  SET @p2 = inShortProductDescriptionLength;
  SET @p3 = inStartItem;
  SET @p4 = inProductsPerPage;

  EXECUTE statement USING @p1, @p2, @p3, @p4;
END$$
?>
```

Эта процедура – прктический пример использования подготовительного оператора, представляющего собой шаблон запроса с параметрами, значения которых задаются непосредственно перед выполнением этого запроса.

С помощью оператора PREPARE мы создаем подготовительный оператор, извлекающий данные о нужных товарах. Переменные в этом операторе обозначаются знаком вопроса (?). Поскольк в нашем примере 5 параметров, мы передаем 5 параметров оператору EXECUTE.

Уровень логики приложения
PHP PDO

Классы PDO позволяют обращаться к различным источникам данных с помощью одного и того же API(интерфейс программирования приложений). Это значит, что используя PDO вам не придется переделывать код, отвечающий за доступ к данным, если вы решите вместо MySQL использовать другую СУБД.

Использование PDO мощный и современный способ взаимодействия с базами данных. В классе PDO содержатся методы для выполнения различных операций с базами данных. Мы можем воспользоваться этими методами для обработки данных, установки соединений с базами данных и решения других типичных задач.

Функциональность класса PDO позволяет установить соединение с базой данных и выполнять SQL-запросы. За открытие соединения отвечает конструктор класса PDO, который в качестве параметра получает строку соединения с сервером базы данных, и необязательный аргумент, указывающий, будет ли создаваемое соединение постоянным. В строке соединения содержится вся информация, необходимая для установки с сервером MySQL.

Создание объекта PDO. Листинг 14.3

```
$dbh = new PDO('mysql:dbname='.$dbname.';host='.$dbhost,
                $dbuser,
                $dbpass,
                array(PDO::ATTR_PERSISTENT => $persistent));
```

Для установки соединения конструктору необходимо передать следующие переменные:

- $dbuser – имя пользователя базы данных
- $dbpass – пароль
- $dbhost – имя хоста (localhost)
- $dbname – имя базы данных
- $persistent – (true, если мы хотим создать постоянное соединение, false – в противном случае).

Чтобы закрыть соединение с базой данных, нужно присвоить переменной $dbn значение null ($dbn = null)

Приведенный ниже пример демонстрирует, как можно создать, открыть и закрыть соединение с базой данный, а также перехватить любые генерируемые при этом исключения.

PDO и обработка исключений. Листинг 14.4

```
try {
// Открываем соединение
$dbh = new PDO('mysql:dbname='.$dbname.';host='.$dbhost,
                $dbuser,
                $dbpass,
                array(PDO::ATTR_PERSISTENT => $persistent));}
// Закрываем соединение
$dbn = null;
}
catch (PDOException $e)
{
 echo 'Облом с подключением: ' . $e->getMessage();
}
```

Открыв соединения, мы достигаем момента, к которому стремились с самого начала занятия: мы можем выполнять через это соединение операторы SQL.

Вот методы PDO, которые мы будем использовать для выполнения SQL-запросов.

- PDOStatement::execute() – используется для выполнения операторов INSERT, UPDATE и DELETE, не возвращающих данные.
- PDOStatment::fetch() – используется для получения из базы данных одной строки данных.
- PSOStatment::fetchAll() – используется для получения множества строк данных.
- PDOStatment::prepare() – подготовительный SQL-оператор. Это параметризованный SQL-запрос, значения параметров которого заменены или метками параметров(?) или именованными переменными (:имя переменной). Преимущество подготовительных операторов перед обычными SQL-запросами заключается в том, что перед выполнением запроса, подготовительный оператор проверяется, что предотвращает атаки инъекцией, и подготовительные операторы выполняются быстрее, поскольку сервер базы данных после первого их выполнения может заново использовать подготовленный план выполнения.

Разработка уровня логики приложения всегда начинается с создания конфигурационного файла, с необходимыми константами

Создание конфигурационного файла с необходимыми константами. Листинг 14.5

```
<?php
// Database connectivity setup
define('DB_PERSISTENCY', 'true');
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'root');
define('DB_PASSWORD', '');
define('DB_DATABASE', 'shop');
define('PDO_DSN', 'mysql:host=' . DB_SERVER . ';dbname=' .
DB_DATABASE);
define('SITE_ROOT', dirname(dirname(__FILE__)));
// Server HTTP port (can omit if the default 80 is used)
define('HTTP_SERVER_PORT', '80');
?>
```

Далее нам необходимо поместить в папку class уже созданный класс DatabaseHandler(). Данный класс будет содержать следующие методы:

- GetHandler() – отвечает за соединение с базой данных
- Execute() – выполняет хранимую процедуру (SQL-запросов) с операторами INSERT, UPDATE, DELETE
- GetAll() – используется для выполнения хранимых процедур, возвращающих множество строк данных.
- GetRow() – используется для выполнения хранимых процедур, возвращающих одну строку данных.
- GetOne() – используется для выполнения запросов, возвращающих единственное значение.

database_handler.php. Листинг 14.6

```
<?php
// Class providing generic data access functionality
class DatabaseHandler
{
  // Hold an instance of the PDO class
  private static $_mHandler;

  // Private constructor to prevent direct creation of object
  private function __construct()
  {
  }

  // Return an initialized database handler
  private static function GetHandler()
  {
    // Create a database connection only if one doesn't already ex-
ist
    if (!isset(self::$_mHandler))
    {
      // Execute code catching potential exceptions
      try
      {
```

```
        // Create a new PDO class instance
        self::$_mHandler =
          new PDO(PDO_DSN, DB_USERNAME, DB_PASSWORD,
                  array(PDO::ATTR_PERSISTENT => DB_PERSISTENCY));

        // Configure PDO to throw exceptions
        self::$_mHandler->setAttribute(PDO::ATTR_ERRMODE,
                                       PDO::ERRMODE_EXCEPTION);
      }
      catch (PDOException $e)
      {
        // Close the database handler and trigger an error
        self::Close();
        trigger_error($e->getMessage(), E_USER_ERROR);
      }
    }

    // Return the database handler
    return self::$_mHandler;
  }

  // Clear the PDO class instance
  public static function Close()
  {
    self::$_mHandler = null;
  }

  // Wrapper method for PDOStatement::execute()
  public static function Execute($sqlQuery, $params = null)
  {
    // Try to execute an SQL query or a stored procedure
    try
    {
      // Get the database handler
      $database_handler = self::GetHandler();

      // Prepare the query for execution
      $statement_handler = $database_handler->prepare($sqlQuery);

      // Execute query
      $statement_handler->execute($params);
    }
    // Trigger an error if an exception was thrown when executing
the SQL query
    catch(PDOException $e)
    {
      // Close the database handler and trigger an error
      self::Close();
      trigger_error($e->getMessage(), E_USER_ERROR);
    }
  }

  // Wrapper method for PDOStatement::fetchAll()
  public static function GetAll($sqlQuery, $params = null,
                                $fetchStyle = PDO::FETCH_ASSOC)
  {
    // Initialize the return value to null
    $result = null;
```

```
    // Try to execute an SQL query or a stored procedure
    try
    {
      // Get the database handler
      $database_handler = self::GetHandler();

      // Prepare the query for execution
      $statement_handler = $database_handler->prepare($sqlQuery);

      // Execute the query
      $statement_handler->execute($params);

      // Fetch result
      $result = $statement_handler->fetchAll($fetchStyle);
    }
    // Trigger an error if an exception was thrown when executing
the SQL query
    catch(PDOException $e)
    {
      // Close the database handler and trigger an error
      self::Close();
      trigger_error($e->getMessage(), E_USER_ERROR);
    }

    // Return the query results
    return $result;
  }

  // Wrapper method for PDOStatement::fetch()
  public static function GetRow($sqlQuery, $params = null,
                                $fetchStyle = PDO::FETCH_ASSOC)
  {
    // Initialize the return value to null
    $result = null;

    // Try to execute an SQL query or a stored procedure
    try
    {
      // Get the database handler
      $database_handler = self::GetHandler();

      // Prepare the query for execution
      $statement_handler = $database_handler->prepare($sqlQuery);

      // Execute the query
      $statement_handler->execute($params);

      // Fetch result
      $result = $statement_handler->fetch($fetchStyle);
    }
    // Trigger an error if an exception was thrown when executing
the SQL query
    catch(PDOException $e)
    {
      // Close the database handler and trigger an error
      self::Close();
      trigger_error($e->getMessage(), E_USER_ERROR);
```

```
    }

    // Return the query results
    return $result;
  }

  // Return the first column value from a row
  public static function GetOne($sqlQuery, $params = null)
  {
    // Initialize the return value to null
    $result = null;

    // Try to execute an SQL query or a stored procedure
    try
    {
      // Get the database handler
      $database_handler = self::GetHandler();

      // Prepare the query for execution
      $statement_handler = $database_handler->prepare($sqlQuery);

      // Execute the query
      $statement_handler->execute($params);

      // Fetch result
      $result = $statement_handler->fetch(PDO::FETCH_NUM);

      /* Save the first value of the result set (first column of
the first row)
         to $result */
      $result = $result[0];
    }
    // Trigger an error if an exception was thrown when executing
the SQL query
    catch(PDOException $e)
    {
      // Close the database handler and trigger an error
      self::Close();
      trigger_error($e->getMessage(), E_USER_ERROR);
    }

    // Return the query results
    return $result;
  }
}
?>
```

Теперь, для того, чтобы выполнить любую SQL-процедуру, необходимо

1. Подключить класс DatabaseHandler()
2. Вызывать хранимую процедуру
3. Передать имя вызываемой процедуры в необходимый метод (Execute(), getAll(), getRow() либо getOne())

Вызов хранимой процедуры. Листинг 14.7

```
class Catalog
{
  // Retrieves all departments
  public static function GetDepartments()
  {
    // После ключевого слова CALL пишем имя процедуры
    $sql = 'CALL catalog_get_departments_list()';

    // Execute the query and return the results
    return DatabaseHandler::GetAll($sql);
  }
}
```

Мы создали класс Catalog с static-методом GetDepartments(). Задача данного метода передать имя выполняемой процедуры в static-метод GetAll() класса DatabeseHandler.

Осталось только разобраться, как данные из логики приложения вывести на экран.

Вывод данных на экран. Листинг 14.8

```
<?php
require_once 'configs/config.php';
require_once 'class/database_handler.php';
require_once 'presentation/catalog.php';
echo "<pre>";
print_r(Catalog::GetDepartments());
echo "</pre>";
?>
```

Где Catalog::GetDepartments() – это массив данных

Пример процедуры с входящими данными:

Процедура catalog_get_department_details с входящим параметром inDepartmentId. Листинг 14.9

```
<?php
CREATE PROCEDURE catalog_get_department_details(IN inDepartmen-
tId INT)
BEGIN
  SELECT name, description
  FROM   department
  WHERE  department_id = inDepartmentId;
END$$
?>
```

Вызов процедуры с передачей данных. Листинг 14.10

```
<?php
  // Retrieves complete details for the specified department
  public static function GetDepartmentDetails($departmentId)
  {
    // Build SQL query
    $sql = 'CALL cata-
log_get_department_details(:department_id)';
```

```
    // Build the parameters array
    $params = array (':department_id' => $departmentId);

    // Execute the query and return the results
    return DatabaseHandler::GetRow($sql, $params);
  }
?>
```

15. Логика представления. Smarty.

Задача Smarty заключается в отделении уровня представления (html, JavaScript, CSS и всё иное, что можно вывести в браузере) от уровня логики приложения (в нашем случае это PHP).

Сперва необходимо установить сам шаблонизатор. Для этого переместите папку «smarty» в корневую директорию вашего сайта. Я использовал версию 3.0.8. Более новую версию можно бесплатно скачать с официального сайта Smarty.

Рассмотрим файл index.php

Пример использования шаблонизатора, файл index.php . Листинг 15.1

```
<?php
require('libs/Smarty.class.php');

$smarty = new Smarty;

$smarty->assign("Name","Изучаем SMARTY");
$smarty-> as-
sign("FirstName",array("Иван","Мария","Виктория","Людмила"));
$smarty-> as-
sign("LastName",array("Чай","Крузанская","Прокопова","Иванова"));

$smarty->display('index.tpl');
?>
```

Как видите, первой строчкой у нас идет подключение класса smarty.class.php. В принципе, это и есть сам шаблонизатор.

Далее мы создаем новый объект класса Smarty. И используя метод assign() устанавливаем переменные. Из листинга видно, что можно создавать как обычные переменные ($Name), так и массив значений.

Важно! Переменные должны быть инициализированы до подключения шаблона index.tpl

А вот и сам шаблон index.tpl . Листинг 15.2

```
{config_load file="test.conf" section="setup"}
{include file="header.tpl" title=foo}

<h1>
{$FirstName[0]} - {$LastName[0]}
</h1>
<h1>
{$FirstName[1]} - {$LastName[1]}
</h1>
<h1>
{$FirstName[2]} - {$LastName[2]}
</h1>
<h1>
{$FirstName[3]} - {$LastName[3]}
</h1>
{* End departments list *}

{include file="footer.tpl"}
```

Переменные

{$foo} <-- отображение простой переменной (не массив и не объект)

{$foo[4]} <-- отображает 5-й элемент числового массива

{$foo.bar} <-- отображает значение ключа "bar" ассоциативного массива, подобно PHP $foo['bar']

{$foo->bar} <-- отображает свойство "bar" объекта

{$foo->bar()} <-- отображает возвращаемое значение метода "bar" объекта

{#foo#} <-- отображает переменную "foo" конфигурационного файла

{$smarty.config.foo} <-- синоним для {#foo#}

Также доступно множество других комбинаций:

{$foo.bar.baz}

{$foo[4].baz} и т.д.

{* отображает серверную переменную "SERVER_NAME" ($_SERVER['SERVER_NAME'])*}

{$smarty.server.SERVER_NAME}

Переменные запроса, такие как $_GET, $_SESSION и т.д. доступны через зарезервированную переменную $smarty.

Конструкция {if},{elseif},{else}

Квалификатор	Альтернативы	Пример синтаксиса	Описание	Эквивалент PHP
==	eq	$a eq $b	равно	==
!=	ne, neq	$a neq $b	не равно	!=
>	gt	$a gt $b	больше	>
<	lt	$a lt $b	меньше	<
>=	gte, ge	$a ge $b	больше или равно	>=
<=	lte, le	$a le $b	меньше или равно	<=
===		$a === 0	проверка идентичности	===
!	not	not $a	отрицание	!
%	mod	$a mod $b	остаток от деления	%

Пример использования конструктора if . Листинг 15.3

```
{if $name eq 'Fred'}
       Welcome Sir.
{elseif $name eq 'Wilma'}
    Welcome Ma'am.
{else}
    Welcome, whatever you are.
{/if}

{* пример с логикой "или" *}
{if $name eq 'Fred' or $name eq 'Wilma'}
       ...
{/if}

{* то же самое, что и выше *}
{if $name == 'Fred' || $name == 'Wilma'}
       ...
{/if}

{* скобки разрешены *}
{if ( $amount < 0 or $amount > 1000 ) and $volume >= #min-
VolAmt#}
       ...
{/if}

{* вы также можете использовать функции php *}
{if count($var) gt 0}
       ...
{/if}

{* проверка на массив *}
{if is_array($foo) }
 ...
{/if}
```

```
{* проверка на существование *}
{if isset($foo) }
 ...
{/if}

{* проверяет чётность значений *}
{if $var is even}
       ...
{/if}
{if $var is odd}
       ...
{/if}
{if $var is not odd}
       ...
{/if}

{* проверяет, делится ли $var на 4 без остатка *}
{if $var is div by 4}
       ...
{/if}

{*
  проверяет, является ли $var чётным двум, например
  0=чётно, 1=чётно, 2=нечётно, 3=нечётно, 4=чётно, 5=чётно и
т.д.
*}
{if $var is even by 2}
       ...
{/if}

{* 0=чётно, 1=чётно, 2=чётно, 3=нечётно, 4=нечётно, 5=нечётно и
т.д. *}
{if $var is even by 3}
       ...
{/if}
```

Вот еще пример использования конструктора if . Листинг 15.4

```
{if isset($name) && $name = 'Blog'}
    {* сделать что-нибудь *}
{elseif $name == $foo}
    {* сделать что-нибудь другое *}
{/if}

{if is_array($foo) && count($foo) > 0)
    {* выполнить цикл foreach *}
{/if}
```

Конструкция {section},{sectionelse}

Секции используются для обхода массивов данных (так же, как и {foreach}). Каждый тэг {section} должен иметь пару {/section}. Обязательными параметрами являются name и loop. Имя цикла {section} может быть любым, состоящим из букв, цифр и знаков подчеркивания.

Пример использования конструкции section, где $LastName – обычный массив данных. Листинг 15.5

```
<div class="box">
 {section name=i loop=$LastName}
  <p class="box-title">ИМЯ:  {$LastName[i]}</p>
 {/section}
</div>
```

Пример использования конструкции section, где $LastName – ассоциативный массив данных (например, множество строк из таблицы данных). Листинг 15.6

```
<div class="box">
 {section name=i loop=$FirstName}
  <p class="box-title">ИМЯ:  {$FirstName[i].name}</p>
 {/section}
</div>
```

Переменная loop определяет только количество итераций.

Вы можете получать доступ к любой переменной из шаблона внутри секции.

Определение переменных в php. Листинг 15.7

```
<?php
$id = array(1001,1002,1003);
$smarty->assign('custid',$id);

$fullnames = array('John Smith','Jack Jones','Jane Munson');
$smarty->assign('name',$fullnames);

$addr = array('253 N 45th', '417 Mulberry ln', '5605 apple st');
$smarty->assign('address',$addr);
?>
```

Этот пример предполагает, что $custid, $name и $address все являются массивами, содержащими одинаковое количество значений.

Переменная loop команды {section}. Листинг 15.8

```
{section name=customer loop=$custid}
<p>
    id: {$custid[customer]}<br />
    name: {$name[customer]}<br />
    address: {$address[customer]}
</p>
{/section}
```

Результат выполнения данного примера. Листинг 15.9

```
<p>
    id: 1000<br />
    name: John Smith<br />
    address: 253 N 45th
</p>
<p>
    id: 1001<br />
    name: Jack Jones<br />
    address: 417 Mulberry ln
```

```
</p>
<p>
    id: 1002<br />
    name: Jane Munson<br />
    address: 5605 apple st
</p>
```

Секции могут иметь вложенность любой глубины. Используя вложенные секции, вы можете обращаться к сложным структурам данных, таким как многомерные массивы.

{sectionelse}

{sectionelse} будет выполнена в том случае, если массив, передаваемый в loop не содержит значений.

Пример использования {sectionelse}. Листинг 15.10

```
{section name=customer loop=$custid}
    id: {$custid[customer]}<br />
{sectionelse}
    there are no values in $custid.
{/section}
```

Все свойства {section}. Листинг 15.11

```
<table>
<tr>
<th>index</th><th>id</th>
<th>index_prev</th><th>prev_id</th>
<th>index_next</th><th>next_id</th>
</tr>
{section name=cus loop=$custid}
<tr>
<td>{$smarty.section.cus.index}</td><td>{$custid[cus]}</td>
<td>{$smarty.section.cus.index_prev}</td><td>{$custid[cus.index_prev]}
</td>
<td>{$smarty.section.cus.index_next}</td><td>{$custid[cus.index_next]}
</td>
</tr>
{/section}
</table>
```

Результатом выполнения этого примера будет таблица, содержащая следующее:

index	id	index_prev	prev_id	index_next	next_id
0	1001	-1		1	1002
1	1002	0	1001	2	1003
2	1003	1	1002	3	1004
3	1004	2	1003	4	1005
4	1005	3	1004	5	

{assign}

{assign} используется для установки значения переменной в процессе выполнения шаблона.

Доступ к переменным {assign} из PHP-скрипта.

Чтобы получить доступ к переменным {assign} из PHP-скрипта, используйте функцию **get_template_vars**(). Обратите внимание, что переменные доступны только во время и после выполнения шаблона, как видно из следующего примера:

Определение переменной foo в smarty-шаблоне. Листинг 15.12

```
{* index.tpl *}
{assign var="foo" value="Smarty"}
```

Доступ к переменной в PHP-скрипте. Листинг 15.13

```
<?php

// это не выведет ничего, ведь шаблон ещё не был выполнен
echo $smarty->get_template_vars('foo');

// получаем шаблон в переменную-пустышку
$dead = $smarty->fetch('index.tpl');

// это выведет 'smarty', так как шаблон уже выполнен
echo $smarty->get_template_vars('foo');

$smarty->assign('foo','Even smarter');

// это выведет 'Even smarter'
echo $smarty->get_template_vars('foo');

?>
```

Плагины Smarty

Структура web-страниц генерируется с помощью шаблонов Smarty. Если какой-то компонент стрницы сложнее и требует вставки PHP-кода, то мы можем использовать шаблоны Smarty, в которые входят следующее:

- Дизайн-шаблон Smarty (с расширением .tpl)
- Подключаемая функция Smarty (PHP-файл)
- Объект представвления (это класс возвращающий данные в Smarty)

Подключаемые файлы и функции Smarty должны соответствовать строгим правилам именования function.имя_функции.php. Пусть в нашем случае будет function.load_presentation_object.php. Функция с данным именем должна находиться в папке plagins. Для того чтобы воспользоваться данной фукцией, ее необходимо подключить в .tpl- шаблоне

Плагин Smarty. Листинг 15.14

```
{load_presentation_object filename="plagin_podrazdels" id=5}
```

Обратите внимание на имя плагина load_presentation_object, оно должно совпасть с именем подключаемого php-файла function.load_presentation_object.php. Помимо подключения данной функции, мы передаем пары значений: filename="plagin_podrazdels" и id=5

А вот и само содержимое функции function.load_presentation_object.php.

Плагин Smarty. Листинг 15.15

```
<?php
// Внутри файла функция должны иметь вид: smarty_function_имя
function smarty_function_load_presentation_object($params,
$smarty)
{
Echo $params['filename']; // это переданный из Smarty параметр.
}
?>
```

16. XML.

Язык Extensible Markup Language (XML) можно назвать и языком разметки, и форматом хранения текстовых данных. Это подмножество языка Standard Generalized Markup Language (SGML); он предоставляет текстовые средства для описания древовидных структур и их применения к информации. XML служит основой для целого ряда языков и форматов, таких как Really Simple Syndication (RSS), Mozilla XML User Interface Language (XUL), Macromedia Maximum eXperience Markup Language (MXML), Microsoft eXtensible Application Markup Language (XAML) и open source-язык Java XML UI Markup Language (XAMJ).

Структура XML

Базовым блоком данных в XML является элемент. Элементы выделяются начальным тегом, таким как <book>, и конечным тегом, таким как </book>. Каждому начальному тегу должен соответствовать конечный тег. Если для какого-то начального тега отсутствует конечный тег, XML-документ оформлен неправильно, и синтаксический анализатор (парсер) не сможет проанализировать его надлежащим образом. Названия тегов обычно отражают тип элемента. Можно ожидать, что элемент book содержит название книги, например, «Большой американский роман» (см. листинг 1). Текст, содержащийся между тегами, включая пробелы, называется символьными данными.

Пример XML-документа. Листинг 16.1

```
<books>
 <book>
  <title>Большой американский роман</title>
  <characters>
   <character>
    <name>Клифф</name>
    <desc>отличный парень</desc>
   </character>
   <character>
    <name>Миловидная Женщина</name>
    <desc>редкая красавица</desc>
   </character>
   <character>
    <name>Преданный Пес</name>
    <desc>любит поспать</desc>
   </character>
  </characters>
  <plot>
   Клифф встречает Миловидную Женщину. Преданный Пес спит,
   но просыпается, чтобы облаять почтальона.
  </plot>
  <success type="bestseller">4</success>
  <success type="bookclubs">9</success>
  </book>
 </books>
```

Имена XML-элементов и атрибутов могут состоять из латинских букв верхнего (A-Z) и нижнего (a-z) регистров, цифр (0-9), некоторых специальных и неанглийских символов, а также трех знаков пунктуации: дефиса, знака подчеркивания и точки. Другие символы в именах не допускаются.

XML чувствителен к регистру. В приведенном примере <Book> и <book> описывают два разных элемента. Оба имени приемлемы. Однако описание двух разных элементов именами <Book> и <book> нельзя считать разумным решением ввиду высокой вероятности опечаток.

Каждый документ XML содержит один и только один корневой элемент. Корневой элемент — это единственный элемент XML-документа, для кото-

рого нет родительского элемента. В приведенном выше примере корневым является элемент <books>. Большинство XML-документов содержат родительские и дочерние элементы. Элемент <books> имеет один дочерний элемент <book>. У элемента <book> четыре дочерних элемента: <title>, <characters>, <plot> и <success>. У элемента <characters> три дочерних элемента, каждый из которых является элементом <character>. У каждого элемента <character> по два дочерних элемента, <name> и <desc>.

Кроме вложенных элементов, что создает отношения родительский-дочерний, XML-элементы могут иметь атрибуты. Это пары имя-значение, присоединенные к начальному тегу элемента. Имена отделяются от значений знаком равенства, =. Значения заключаются в одинарные или двойные кавычки. В листинге 1 элемент <success> имеет два атрибута: "bestseller" и "bookclubs". XML-разработчики практикуют разные подходы к использованию атрибутов. Большую часть информации, содержащейся в атрибуте, можно поместить в дочерний элемент. Некоторые разработчики настаивают на том, чтобы информация атрибутов состояла не из данных, а из метаданных, то есть сведений о данных. Сами данные должны содержаться в элементах. На самом деле решение о том, использовать ли атрибуты, зависит от природы данных и от того, как они извлекаются из XML.

Достоинства XML

Одно из достоинств XML состоит в его относительной простоте. XML-документ можно составить в простом текстовом редакторе или текстовом процессоре, не прибегая к специальным инструментам или ПО. Базовый синтаксис XML состоит из вложенных элементов, некоторые из которых имеют атрибуты и содержание. Обычно элемент начинается открывающим тегом <тег> и заканчивается соответствующим закрывающим тегом </тег >. XML чувствителен к регистру и не игнорирует пробелы и табуляции. Он очень похож на HTML, но, в отличие от HTML, позволяет присваивать тегам имена для лучшего описания свох данных. К числу преимуществ XML относится самодокументирование, читабельный для людей и компьютеров формат, поддержка Unicode, что позволяет создавать документы на разных языках, и простые требования к синтаксису и синтаксическому анализу. К сожалению, в PHP5 поддержка UTF-8 сопряжена с проблемами; это один из тех недостатков, которые привели к разработке PHP6.

Недостатки XML

XML многословен и избыточен, что порождает документы большого объема, занимающие много дискового пространства и сетевых ресурсов. Предполагается, что он должен быть читабелен для людей, но трудно представить себе человека, пытающегося прочесть файл XML с 7 млн. узлов. Простейшие синтаксические анализаторы функционально не способны поддерживать широ-

кий набор типов данных; по этой причине редкие или необычные данные, каких бывает много, становятся серьезным источником затруднений.

Правильно построенный XML

XML-документ считается построенным правильно, если в нем соблюдены правила XML-синтаксиса. Неправильно построенный формат в техническом смысле не является XML-документом. Например, такой HTML-тег, как
, в XML неприемлем; соответствующий правильный тег выглядит как
. Корневой элемент можно представить себе как бесконечный шкаф с документами. У вас всего один шкаф, но почти нет ограничений на тип и количество его содержимого. В вашем шкафу помещается бесконечное количество ящиков и папок для документов.

PHP5 и XML

Поддержка XML присутствовала в PHP с самых ранних версий, но в PHP5 она существенно улучшена. Поддержка XML в PHP4 была ограниченной, в частности, предлагался только включенный по умолчанию парсер на базе SAX, а поддержка DOM не соответствовала стандарту W3C. В PHP5 разработчики PHP XML, можно сказать, изобрели колесо, обеспечив соответствие общепринятым стандартам.

Новое в поддержке XML в версии PHP5

PHP5 содержит полностью переписанные и новые расширения, включая парсер SAX, DOM, SimpleXML, XMLReader, XMLWriter и процессор XSLT. Теперь все эти расширения основаны на libxml2.

Наряду с улучшенной по сравнению с PHP4 поддержкой SAX, в PHP5 реализована поддержка DOM в соответствии со стандартом W3C, а также расширение SimpleXML. По умолчанию включены и SAX, и DOM, и SimpleXML. Тем, кто знаком с DOM по другим языкам, станет проще реализовать аналогичную функциональность в PHP

DOM в действии

Модель DOM, реализованная в PHP5, — это та же спецификация W3C DOM, с которой вы имеете дело в браузере и с которой работаете посредством JavaScript. Используются те же методы, так что способы кодирования покажутся вам знакомыми. Листинг 2 иллюстрирует использование DOM для создания XML-строки и XML-документа, отформатированных в целях читабельности.

Применение DOM. Листинг 16.2

```
<?php

//Создает XML-строку и XML-документ при помощи DOM
$dom = new DomDocument('1.0');

//добавление корня - <books>
$books = $dom->appendChild($dom->createElement('books'));

//добавление элемента <book> в <books>
$book = $books->appendChild($dom->createElement('book'));

// добавление элемента <title> в <book>
$title = $book->appendChild($dom->createElement('title'));

// добавление элемента текстового узла <title> в <title>
$title->appendChild(
                $dom->createTextNode('Great American Novel'));

//генерация xml
$dom->formatOutput = true; // установка атрибута formatOutput
                           // domDocument в значение true
// save XML as string or file
$test1 = $dom->saveXML(); // передача строки в test1
$dom->save('test1.xml'); // сохранение файла
?>
```

Это приводит к созданию выходного файла, приведенного в листинге 3.

Выходной файл. Листинг 16.3

```
<?xml version="1.0"?>
<books>
  <book>
    <title>Great American Novel</title>
  </book>
</books>
```

18.Защита PHP.

Рекомендованные значения параметров PHP

- register_globals = off. Считается, что не выключенный данный параметр - наибольшая дыра в PHP. У PHP есть довольно древний механизм, облегчающий доступ к параметрам, переданным методами GET и POST. Например, если имеется POST-параметр с именем myparam, то PHP автоматически создаст переменную $myparam и присвоит ей значение параметра myparam. Однако всегда есть риск того, что вы забудете инициализировать ту или иную глобальную переменную, и это станет потенциальной уязвимостью вашего сценария. Эта функция выключается в файле php.ini.
- safe_mode = off

- error_reporting = off. Вместо вывода ошибок в браузер, лучше использовать их протоколирование.
- Следующие функции должны быть отключены: system(), exec(), passthru(), shell_exec(), proc_open() и popen(). Данные функции позволяют выполнять команды непосредственно на сервере.
- allow_url_open = off. Нет никакой надобности обращаться к файлам по http.

Атаки типа SQL-инъекция (вставка вредоносного кода)

Если вы не используете в своих сценариях трехуровневую архитектуру данных (т.е. SQL-запросы находятся непосредственно в PHP-файлах), то вы подвергаете себя риску SQL-атаке. Поэтому всегда необходимо проверять вставляемые параметры перед тем, как их пропускать через функцию query_mysql(). Для проверки числа подойдет функция intval (). Для проверки строки – лучше использовать регулярные выражения.

XSS-атаки

XSS – это кросс-сайтовый скриптинг. В отличие от большинства атак, уязвимость работает на стороне клиента. Как правило, это вставка вредоносного кода через формы. По этой причине на большинстве форумов запрещено использовать чистый html-код, а вместо него используют теги форматирования, например [b] или [linkto]. Для предотвращения такого типа атак строку можно пропускать через следующие функции: htmlentities, htmlspecialchar(), str_replace – для замены html-символов.

Преобразованием специальных HTML-символов в литералы занимается функция htmllentities(). Хотя эта функция помогает предотвратить большинство атак, опытные хакеры могут преобразовывать свои вредоносные скрипты в шеснадцатиричный код или в UTF-8 вместо обычного ASCII-текста, что позволяет обойти наши фильтры. Вот пример передачи переменной var (методом GET) строки, заданной в шеснадцатиричном виде

Пример строки заданной в шеснадцатиричном виде. Листинг 18.1

```
<a href=”Index.php?var=%22%3e%3c%$53%43%52%49%50%54%3e%44%6f%73
%6f%6d%65%74%68%69%6e%67%6d%61%6c%69%63%69%6f%75%73%3c%2f%53%43
%52%49%50%54%3e”>
```

В браузере мы увидим следующий код

Выполненный шеснадцатиричный код. Листинг 18.2

```
<a href=”Index.php?var=”><SCRIPT>Dosomethingmalicious</SCRIPT>
```

Чтобы превратить такой поворот событий, необходимо дополнительно конвертировать знаки % и #, что предотвращает шеснадцатиричную XSS-атаку и конвентирует UTF-закодированные данные.

Выполненный шеснадцатиричный код. Листинг 18.3

```
$string = srt_replace("%", "&#37", $string);
$string = srt_replace("#", "&#35", $string);
```

Также можно лимитизировать длину строки.

Использование SafeHTML

SafeHTML позволяет распознать и удалить все опасные HTML-теги.

1. Загрузитие последнюю верскию SafeHTML по адресу http://pixel-apes.com/safehtml/?page=safehtml
2. Поместите файлы в каталог classes на вашем сервере.
3. Подключите класс SafeHTML (файл safehtml.php) к вашему сценарию.
4. Проверяйте данные методом $safehtml->parse()

Вот небольшой пример:

Проверяем данные с помощью SafeHTML. Листинг 18.4

```
require_once('classes/safehtml.php')
$data = "Текст для проверки";
$safehtml = new safehtml(); //создаем объект класса safehtml()
$safe_data = $safehtml->parse($data);
```

Методическое издание

Михалькевич Александр Викторович

PHP. Практика создания сайтов.

Авторская редакция
Компьютерная верстка Михалькевич А.В.

Подписано в печать 25.03.2012. Формат 60x84 1/16.
Бумага HP Office, Печать лазерная.
Усл. печ. л. . Уч.-изд. л. .
Тираж 1000 экз. Заказ.