Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Информационные технологии»

И. Л. Стефановский, Л. К. Титова

ПРОГРАММИРОВАНИЕ В INTERNET

Учебно-методическое пособие

Рекомендовано учебно-методическим объединением высших учебных заведений Республики Беларусь по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для студентов высших учебных заведений, обучающихся по специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» направления 1-40 05 01-01 «Информационные системы и технологии (в проектировании и производстве)»

Электронный аналог печатного издания

Гомель 2015

УДК 004.738.5:004.42(075.8)
ББК 32.973я73
С79

Рецензенты: зав. каф. информационно-вычислительных систем Белорусского торгово-экономического университета потребительской кооперации д-р техн. наук, проф. *А. Н. Семенюта*;
зав. каф. автоматизированных систем обработки информации Гомельского государственного университета имени Ф. Скорины канд. техн. наук, доц. *В. Д. Левчук*

Стефановский И. Л.

С79 Программирование в Internet : учеб.-метод. пособие / И. Л. Стефановский, Л. К. Титова ; М-во образования Респ. Беларусь, Гомел. гос. техн. ун-т им. П. О. Сухого. – Гомель : ГГТУ им. П. О. Сухого, 2015. – 112 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

ISBN 978-985-535-247-2.

Изложены основные темы курса «Программирование в Internet». Рассмотрены вопросы развития глобальной компьютерной сети Интернет, а также технологии разработки приложений, ориентированной на работу в Интернет.

Для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» направления 1-40 05 01-01 «Информационные системы и технологии (в проектировании и производстве)».

УДК 004.738.5:004.42(075.8)
ББК 32.973я73

ISBN 978-985-535-247-2

ПРЕДИСЛОВИЕ

Современный специалист в области разработки программного обеспечения должен уметь пользоваться технологиями Интернет и владеть основами создания Web-приложений различного уровня сложности.

Дисциплина «Программирование в Internet» знакомит студентов с основными концепциями, технологиями и тенденциями развития глобальной компьютерной сети Интернет, а также с технологией разработки приложений, ориентированной на работу в Интернет и получению навыков решения задач Интернет-программирования.

Цель дисциплины – изучение современных методов программирования приложений, использующих в своей работе среду Internet, создания интернет-сайтов, наполненных динамически изменяющимся контентом.

Задачи дисциплины – овладение технологией создания динамических Web-страниц, обучение программированию в Internet на стороне клиента и сервера.

В результате изучения дисциплины «Программирование в Internet» студенты должны *знать*:

– основные принципы и технологии организации глобальной компьютерной сети Интернет;

– основы построения и функционирования прикладных сервисов Интернет;

– основные технологии прикладного программирования для Интернет;

уметь:

– проектировать и создавать динамические веб-сайты;

– формулировать и решать задачи проектирования веб-ориентированных Интернет/Интранет-приложений с использованием современных технологий клиентского и серверного программирования.

Настоящее учебно-методическое пособие предназначено для студентов специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» и соответствует программе курса «Программирование в Internet». Знания и умения, приобретенные студентами при изучении указанной дисциплины, могут быть использованы при решении различных практических задач.

Глава 1. Виды динамических HTML-документов. CSS 3.0

Динамический HTML (dynamic HTML или DHTML) – это набор средств, которые позволяют создавать более интерактивные Web-страницы без увеличения загрузки сервера. DHTML не является языком разметки страниц. Это термин, применяемый для обозначения HTML-страниц с динамически изменяемым содержимым. Реализация DHTML основывается на HTML, каскадных таблицах стилей (Cascade Style Sheets – CSS) и языке сценариев (JavaScript или VBScript). Эти три компонента DHTML связаны между собой объектной моделью документа (Document Object Model — DOM), являющейся интерфейсом прикладного программирования (API).

DOM связывает воедино эти три компонента, обеспечивая для HTML-документа возможность динамического изменения своего содержимого без перезагрузки страницы.

Объектная модель документа делает все элементы страницы программируемыми объектами. С ее помощью через языки сценариев можно получить доступ и управлять всеми элементами документа. Каждый элемент HTML доступен как индивидуальный объект, следовательно, можно изменять значение того или иного параметра любого тэга HTML-страницы, и, как следствие, документ действительно становится динамическим. Любое действие пользователя объектной моделью документа трактуется как событие, которое может быть перехвачено и обработано процедурой сценария.

Каскадные таблицы стилей можно сравнить со стилевыми файлами любого текстового редактора. С их помощью определяется внешний вид отображаемого HTML-документа: цвет шрифта и фона документа, сам шрифт, разбивка текста и др. Для каждого элемента, задаваемого определенным тэгом HTML, можно определить свой стиль отображения в окне браузера. Например, заголовки первого уровня будут отображаться шрифтом Arial 16pt синего цвета, заголовки второго уровня – Arial 14pt красного цвета, основной текст – Times New Roman 10pt черного цвета с одинарным интервалом между строками. Можно создать таблицу стилей и использовать ее для всех документов, расположенных на сервере, что придаст стройность и строгость всему Web-сайту.

Объектная модель документа делает все элементы страницы программируемыми объектами. С ее помощью через языки сценариев

можно получить доступ и управлять всем, что есть в документе. Каждый элемент HTML доступен как индивидуальный объект, а это означает, что можно изменять значение любого параметра любого тэга HTML-страницы, и, как следствие, документ действительно становится динамическим. Любое действие пользователя (щелчок кнопкой мыши, перемещение мыши в окне браузера или нажатие клавиши клавиатуры) объектной моделью документа трактуется как событие, которое может быть перехвачено и обработано процедурой сценария.

DHTML – достаточно новая технология и не все браузеры поддерживают объектную модель документа и каскадные таблицы стилей. Однако DHTML использует стандартные тэги HTML и поэтому пользователи браузеров, не поддерживающих DOM, практически увидят все, что задумано разработчиком динамической страницы, но только в статическом виде.

Каскадные таблицы стилей впервые были реализованы в Internet Explorer 3.0. При реализации Internet Explorer 4.0 были приняты во внимание рекомендации REC-CSS1 Консорциума W3 относительно каскадных таблиц стилей, датированные 17 декабря 1996 г. К настоящему времени они пересмотрены и известны как рекомендации по каскадным таблицам стилей, уровень 1, документ REC-CSS1-19990111 от 11 января 1999 г. В мае 1998 г. Консорциум издал рекомендации по каскадным таблицам стилей, уровень 2, документ REC-CSS2-19980512, часть из которых реализована в Internet Explorer 4.01.

Каскадные таблицы стилей, уровень 1, представляют собой простую технологию определения и присоединения стилей к документам HTML. Стиль, говоря простым языком, – это все то, что определяет внешний вид документа HTML при его отображении в окне браузера: шрифты и цвета заголовков разных уровней, шрифт и разрядка основного текста, задаваемого в тэге абзаца <P>, и т. д. Стиль задается по определенным правилам, а таблица стилей – набор правил отображения, применяемых в документе, к которому присоединена соответствующая таблица стилей. Таблица стилей – это шаблон, который управляет форматированием тэгов HTML в Web-документе.

Почему в название таблиц стилей включено определение «каскадные»? Дело в том, что рекомендации Консорциума W3 позволяют использовать несколько таблиц стилей для управления форматированием одного документа HTML, а браузер по определенным правилам выстраивает приоритетность применения этих таблиц. Они выстраиваются неким «каскадом», по которому и «прокатывается» документ.

Правила приоритетности и разрешения возникающих конфликтов будут описаны ниже в данном разделе.

Для разработки таблицы стилей достаточно немного ориентироваться в языке HTML и быть знакомым с базовой терминологией настольных издательских систем. Таблица стилей представляет собой набор правил форматирования элементов HTML. Эти правила достаточно просты и легко запоминаемы. Например, если необходимо, чтобы в документе все заголовки первого уровня отображались синим цветом и шрифтом с кеглем (размером) 16 пунктов, то в таблице следует задать правило:

H1 {color: blue;
font-size: 16pt}

Любое правило каскадных таблиц стилей состоит из двух частей: селектора и определения. Селектором может быть любой тэг HTML, для которого определение задает, каким образом необходимо его форматировать. Само определение, в свою очередь, также состоит из двух частей: свойства и его значения, разделенных знаком двоеточия (:). Назначение свойства очевидно из его названия. В приведенном правиле селектором является элемент H1, а определение, записанное в фигурных скобках, задает значения двух свойств заголовка первого уровня: цвет шрифта (свойство color) определен как синий (значение blue) и размер шрифта (свойство font-size) определен в 16 пунктов (значение 16pt). В одном правиле можно задать несколько определений, разделенных символом *точка с запятой* (;), как это демонстрируется в приведенном примере.

Созданная только что таблица стилей влияет на форматирование элемента определенного типа: заголовок первого уровня. Ее комбинация с другими таблицами стилей определяет окончательное представление документа при его просмотре в окне браузера.

Синтаксис правил каскадных таблиц стилей не чувствителен к регистру. Селекторы, свойства и их значения можно задавать как строчными, так и прописными буквами, или в смешанном порядке. Однако каскадные таблицы стилей чувствительны к синтаксису задания правил и правильности названий свойств, значений и селекторов. Та или иная грамматическая ошибка приводит к тому, что правило пропускается анализатором браузера, и никакого предупреждающего сообщения не появляется.

Встраивание таблиц стилей в документ

Чтобы таблица стилей могла воздействовать на внешнее представление документа, браузер должен знать о ее существовании. Для этого ее необходимо связать с HTML-документом.

Существует четыре способа связывания документа и таблицы стилей:

1) связывание – позволяет использовать одну таблицу стилей для форматирования многих страниц HTML;

2) внедрение – позволяет задавать все правила таблицы стилей непосредственно в самом документе;

3) импортирование – позволяет встраивать в документ таблицу стилей, расположенную на сервере;

4) встраивание в тэги документа – позволяет изменять форматирование конкретных элементов страницы.

Связывание позволяет хранить таблицу стилей в отдельном файле и присоединять ее к документам с помощью тэга <LINK>, задаваемого в разделе <HEAD>:

<LINK REL="stylesheet" TYPE="text/css" HREF="mystyles.css">

Связываемый файл содержит набор правил каскадных таблиц стилей, определяющих форматирование документа, и должен иметь расширение CSS.

Связывание позволяет разработчику применить одинаковый набор правил форматирования к группе HTML-документов, что приводит к единообразному отображению различных документов и придает некоторую системность серверу разработчика.

При внедрении таблицы стилей в документ правила, ее составляющие, задаются в стилевом блоке, ограниченном тэгами <STYLE TYPE="text/css"> и </STYLE>, который должен размещаться в разделе <HEAD> документа:

```
<HEAD>
<STYLE TYPE="text/css">
<!---
B { text-transform: uppercase; }
P { background-color: lightgrey;
text-align:center; }
 --->
</STYLE>
</HEAD>
```

Обычно браузеры, не поддерживающие какие-либо тэги, игнорируют их, интерпретируя их содержимое в том виде, в каком оно задано, что может приводить к ошибкам. Поэтому следует задавать содержимое тэгов, которые потенциально не обрабатываются старыми версиями браузеров, заключенных в тэг комментария <!-- ... -->.

В приведенном выше примере встроенная таблица стилей определяет отображение всех абзацев в документе (элемент P) на сером фоне с центрированными строками. Полужирный текст, определяемый любым элементом B (тэг <B>) документа, будет отображаться прописными буквами, даже если в документе он задан строчными.

В тэге <STYLE> можно импортировать внешнюю таблицу стилей с помощью свойства @import таблицы стилей:

@import: url(mystyles.css);

Его следует задавать в начале стилевого блока или связываемой таблицы стилей перед заданием остальных правил. Значением свойства @import является URL-адрес файла таблицы стилей.

Последний способ задания значений свойств таблицы стилей предназначен для оперативного форматирования определенного элемента документа и называется *внедрение*. Каждый тэг HTML имеет параметр STYLE, в котором можно задать значения его свойств в соответствии с синтаксисом каскадных таблиц стилей. Например, в следующем примере задается форматирование заголовка первого уровня, определяющее его отображение шрифтом красного цвета:

```
<H1 STYLE="color:red">Заголовок отображается шрифтом
                      красного цвета </H1>
```

Если связанные, внедренные и импортируемые таблицы стилей влияют на форматирование всех элементов документа, для которых определены в таблицах правила, то встраивание определений стилей в конкретный тэг влияет на отображение только элемента, определяемого данным тэгом.

Рекомендуется избегать встраивания в тэги документа определений форматирования, так как подобная техника лишает разработчика преимуществ задания таблиц стилей в отдельном файле или в головной части документа, где их можно легко и быстро откорректировать, в случае необходимости. При форматировании отдельных тэгов придется просмотреть весь документ целиком, что требует достаточно большой и кропотливой работы.

Все способы встраивания таблиц стилей свободно сочетаются в одном документе. Например, можно разработать главную таблицу

стилей для всех документов и связывать ее с каждым HTML-документом. Импортируемая или внедряемая таблица стилей будет уточнять форматирование элементов конкретного документа, а встраиваемые в тэг определения форматирования будут уточнять их отображение.

Группирование и наследование

Правила каскадных таблиц стилей состоят из селектора и определения. Для уменьшения размеров таблиц стилей можно группировать разные селекторы в виде списка элементов страницы HTML, разделенных запятыми, если для них задается одно правило. Например, следующие правила:

```
H1 {font-family: Arial}
H2 {font-family: Arial}
H3 {font-family: Arial}
```

можно сгруппировать и задать в виде одного правила со списком селекторов

```
H1, H2, H3 {font-family: Arial}
```

Аналогично группируются определения, только в списке они разделяются точками с запятой (;). Следующие правила форматирования заголовка первого уровня:

```
H1 {font-weight: bold}
H1 {font-size: 14pt}
H1 {font-family: Arial}
```

можно задать в виде одного правила, сгруппировав определения:

```
H1 {
font-weight: bold;
font-size: 14pt;
font-family: Arial;
}
```

Некоторые свойства имеют собственный синтаксис группирования, связанный с заданием значений нескольких свойств в одном. Например, предыдущий пример при использовании свойства font запишется так:

```
H1 {font: bold 14pt Arial}
```

При задании таблицы стилей можно свободно комбинировать все три правила группирования для уменьшения ее размеров.

В HTML некоторые элементы могут содержать другие. Как будет отображаться элемент, расположенный внутри другого элемента

страницы, если для последнего задано правило форматирования, а для вложенного элемента нет? Например, пусть цвет шрифта абзаца определен как синий (p {color: blue}). Как будет отображаться выделенный элемент текста, задаваемый тэгом <EM>, если для него не определено правило форматирования?

В подобных случаях вложенный элемент наследует правила форматирования элемента-родителя. В нашем примере выделенный элемент будет также отображаться синим цветом. Другие свойства ведут себя аналогично свойству color, например font-family, font-size.

Некоторые свойства не наследуются вложенными элементами от своих родителей, например свойство background, но по умолчанию вложенные элементы будут отображаться с фоном родительского элемента.

Наследование полезно при задании значений свойств, применяемых к документу по умолчанию. Для этого достаточно задать все свойства для элемента, порождающего все остальные элементы страницы HTML. Таким элементом является тело документа, определяемое тэгом <BODY> :

```
BODY {
color: black;
font-family: "Times New Roman";
background: url(texture.gif) white;
 }
```

Приведенные правила задают форматирование документа по умолчанию: черным шрифтом гарнитуры Times New Roman с фоном, задаваемым графическим файлом texture.gif, или на белом фоне, если файл не доступен.

Приведенное задание правил форматирования по умолчанию будет работать всегда, даже если разработчик пропустит в документе тэг <BODY>, что допускается стандартом языка HTML, так как синтаксический анализатор HTML всегда вставляет пропущенный тэг <BODY>.

Селекторы

Правила каскадных таблиц стилей, в которых в качестве селектора используются тэги HTML, влияют на отображение всех элементов заданного типа в документе. Следующее правило отображает без подчеркивания все ссылки (тэг <A>) в документе:

```
<STYLE TYPE="text/css">
<!---
A { text-decoration:none; }
 -->
</STYLE>
```

А что делать, если необходимо некоторые ссылки отобразить по-другому? Можно задать для них правило форматирования непосредственно в тэге, а можно применить параметр CLASS, добавленный в HTML 4.0 в качестве стандарта для всех тэгов. Значением параметра CLASS является ссылка на класс, задаваемый в таблице стилей.

CSS 3 – это новый стандарт оформления HTML документов, значительно расширяющий возможности стандарта CSS2.1. Многие возможности, которые были труднодоступны в CSS2.1, т. е. требовали использования дополнительных внешних программ (таких как Adobe Photoshop), скриптов (таких как JavaScript) или специальных «хитростей» могут легко достигаться в CSS 3 за счет использования новых свойств оформления. Более привлекательный дизайн сайтов с использованием минимально необходимого кода — вот результат, получаемый с помощью CSS 3.

В CSS 3 можно:

– создавать элементы со сглаженными углами;

– создавать линейные и сферические градиенты;

– более гибко оформлять фоновую картинку элементов;

– добавлять к элементам и к тексту элементов тени;

– использовать небезопасные шрифты (не боясь при этом, что они будут не поддерживаться браузером пользователя);

– создавать анимацию и различные эффекты переходов;

– задавать цвета несколькими новыми способами и др.

Пример:

```
background:-webkit-linear-gradient(top,#E567B1,#84004D);
background:-moz-linear-gradient(top,#CB0077,black);
background:-o-linear-gradient(top,#CB0077,black);
box-shadow:3px 3px 10px 1px #000000;
```

Новые свойства CSS 3 поддерживаются только в современных браузерах: IE9+, Firefox 3.6+, Opera 10+, Chrome 12+, Safari 5+. Большинство модулей все еще продолжает совершенствоваться и модифицироваться, и ни один браузер не поддерживает все модули.

Глава 2. Программирование на стороне клиента. JavaScript

2.1. Преимущества и ограничения программ, работающих на стороне клиента. Язык JavaScript. Основы синтаксиса

JavaScript – новый язык для составления скриптов, разработанный фирмой Netscape, это язык программирования, построенный на алгоритмическом принципе. Основная задача Javascript – создавать последовательность действий, которые будут приводить к определенному результату. Веб-сайты на JavaScript придают функциональность и дополнительные визуальные эффекты веб-странице.

С помощью *сценариев JavaScript* можно выполнять следующие задачи:

– динамически изменять вид и содержимое страницы;

– проверять и передавать данные, введенные пользователем в поля формы;

– создавать анимационные эффекты с текстом и отображать графику в виде слайд-шоу;

– создавать галерею фото или картин;

– воспроизводить звуковое сопровождение страницы в соответствии с контекстом;

– отображать сообщение в строке состояния браузера, а также время и дату;

– создавать навигационную панель, как один из вариантов;

– делать Web-страницу интерактивной и вести диалог с пользователем с помощью выходящих диалоговых окон при активации событий.

Сценарии языка JavaScript привязаны к html-коду страницы и могут выполняться только при участии приложения обозревателя.

Код скрипта JavaScript размещается непосредственно на HTML-странице. Для того чтобы увидеть, как это делается, рассмотрим следующий простой пример:

```
<html>
<body>
<br>
Это обычный HTML документ.
<br>
  <script language="JavaScript">
```

```
  document.write("А это JavaScript!")
 </script>
<br>
Вновь документ HTML.
</body>
</html>
```

С первого взгляда пример напоминает обычный файл HTML. Единственное новшество здесь – конструкция:

```
 <script language="JavaScript">
  document.write("А это JavaScript!")
 </script>
```

Это действительно код JavaScript. Для того чтобы увидеть, как этот скрипт работает, необходимо записать данный пример как обычный файл HTML и загрузить его в браузер, имеющий поддерку языка JavaScript. В результате получим три строки текста:

Это обычный HTML документ.
А это JavaScript!
Вновь документ HTML.

Все, что стоит между тэгами <script> и </script>, интерпретируется как код на языке JavaScript. Здесь виден пример использования инструкции document.write() – одной из наиболее важных команд, используемых при программировании на языке JavaScript. Команда document.write() используется, когда необходимо что-либо написать в текущем документе (в данном случае таким является наш HTML-документ). Таким образом, наша небольшая программа на JavaScript в HTML-документе пишет фразу "А это JavaScript!".

А как будет выглядеть страница, если браузер не воспринимает JavaScript? Браузеры, не имеющие поддержки JavaScript, «не знают» и тэга <script>. Они игнорируют его и печатают все стоящие вслед за ним коды как обычный текст. Иными словами, можно увидеть, как код JavaScript, приведенный в нашей программе, окажется вписан открытым текстом прямо посреди HTML-документа. На этот случай имеется специальный способ скрыть исходный код скрипта от старых версий браузеров – для этого используется тэг комментария из HTML – <!-- -->.

В результате новый вариант нашего кода будет выглядеть как:

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

```
Это обычный HTML документ.
<br>
 <script language="JavaScript">
 <!-- скрывает код от старых браузеров
       document.write("А это JavaScript!")
   -->
 </script>
<br>
Вновь документ HTML.
</body>
</html>
```

В этом случае браузер без поддержки JavaScript будет печатать:

Это обычный HTML документ.
Вновь документ HTML.

А браузер без поддержки JavaScript без HTML-тэга комментария напечатал бы:

Это обычный HTML документ.
document.write("А это JavaScript!")
Вновь документ HTML.

Обратите внимание, что нельзя полностью скрыть исходный код JavaScript. То, что здесь продемонстрировано, имеет целью предотвратить распечатку кода скрипта на старых браузерах – однако тем не менее можно увидеть этот код посредством пункта меню *View document source*.

События и обработчики событий являются очень важной частью для программирования на языке JavaScript. События главным образом инициируются теми или иными действиями пользователя. Например, если пользователь щелкает по некоторой кнопке, происходит событие Click, если указатель мыши пересекает какую-либо ссылку гипертекста – происходит событие MouseOver. Существует несколько различных типов событий. Мы можем заставить нашу JavaScript-программу реагировать на некоторые из них, и это может быть выполнено с помощью специальных программ обработки событий. Так, в результате щелчка по кнопке, может создаваться выпадающее окно. Это означает, что создание окна должно быть реакцией на событие щелка – Click. Программа-обработчик событий, которую мы должны использовать в данном случае, называется onClick. Она сообщает компьютеру, что нужно делать, если произойдет данное со-

бытие. Приведенный ниже код представляет простой пример программы обработки события onClick:

```
<form>
<input type="button" value="Click me" onClick="alert('Yo')">
</form>
```

В данном примере для начала создается некая форма с кнопкой (это делается при помощи языка HTML). onClick="alert('Yo')" в тэге <input> определяет, что происходит, когда нажимают на кнопку. Таким образом, если имеет место событие Click, компьютер должен выполнить вызов alert('Yo'). Это и есть пример кода на языке JavaScript (обратите внимание, что в этом случае мы не пользуемся тэгом <script>). Функция alert() позволяет создавать выпадающие окна. Таким образом, при щелчке на кнопке наш скрипт создает окно, содержащее текст 'Yo'.

В команде document.write( ) были использованы двойные кавычки ("), а в конструкции alert( ) – только одинарные. В большинстве случаев используются оба типа кавычек. Однако в последнем примере мы написали onClick="alert('Yo')" – т. е. использовали и двойные, и одинарные кавычки. Если бы написали onClick="alert("Yo")", то компьютер не смог бы разобраться в нашем скрипте, поскольку становится неясно, к какой из частей конструкции имеет отношение функция обработки событий onClick, а к какой – нет. Поэтому в данном случае необходимо использовать оба типа кавычек. Порядок использования кавычек не имеет значения, т. е. можно по аналогии написать onClick='alert("Yo")'.

В данном примере выпадающее окно содержит текст, который передан функции JavaScript с помощью метода alert. Такое ограничение накладывается по соображениям безопасности. Такое же выпадающее окно можно создать и с помощью метода prompt(), который выводит сообщение в окне с текстовым полем и двумя кнопками: «ОК» и «ОТМЕНА».

В большинстве программ на языке JavaScript пользуются *функциями.* Функции представляют собой лишь способ связать вместе нескольких команд. Напишем скрипт, печатающий некий текст три раза подряд. Для начала рассмотрим простой подход:

```
<html>
<script language="JavaScript">
<!-- hide
document.write("Добро пожаловать на мою страницу!<br>");
```

```
document.write("Это JavaScript!<br>");
document.write("Добро пожаловать на мою страницу!<br>");
document.write(Это JavaScript!<br>");
document.write("Добро пожаловать на мою страницу!<br>");
document.write("Это JavaScript!<br>");
// -->
</script>
</html>
```

Такой скрипт напишет следующий текст три раза.

Добро пожаловать на мою страницу!
Это JavaScript!

Если посмотреть на исходный код скрипта, то видно, что для получения необходимого результата определенная часть его кода была повторена три раза. Но это неэффективно. Рассмотрим следующий скрипт для решения этой же задачи:

```
<html>
<script language="JavaScript">
<!-- hide
function myFunction() {
  document.write("Добро пожаловать на мою страницу!<br>");
  document.write("Это JavaScript!<br>");
}
myFunction();
myFunction();
myFunction();
  -->
</script>
</html>
```

В этом скрипте мы определили некую функцию, состоящую из следующих строк:

```
function myFunction()
{
  document.write("Добро пожаловать на мою страницу!<br>");
  document.write("Это JavaScript!<br>");
}
```

Команды скрипта, находящиеся внутри фигурных скобок – { } –, принадлежат функции myFunction(). Это означает, что обе команды document.write() теперь связаны воедино и могут быть выполнены при

вызове указанной функции. Действительно, в нашем примере есть три вызова этой функции. Можно увидеть, что мы написали строку myFunction() три раза сразу после того, как дали определение самой функции, т. е. как раз и сделали три вызова. В свою очередь, это означает, что содержимое этой функции было выполнено трижды.

Функции могут также использоваться совместно с процедурами обработки событий. Рассмотрим следующий пример:

```
<html>
<head>
<script language="JavaScript">
<!-- hide
function calculation() {
  var x= 12;
  var y= 5;
  var result= x + y;
  alert(result);
}
  -->
</script>
</head>
<body>
<form>
<input type="button" value="Calculate" onClick="calculation( )">
</form>
</body>
</html>
```

Здесь при нажатии на кнопку осуществляется вызов функции calculation(). Как можно заметить, эта функция выполняет некие вычисления, пользуясь переменными x, y и result. Переменную можем определить с помощью ключевого слова var. Переменные могут использоваться для хранения различных величин – чисел, строк текста и т. д. Так, строка скрипта var result= x + y; сообщает браузеру о том, что необходимо создать переменную result и поместить туда результат выполнения арифметической операции x + y (т. е. 5 + 12). После этого в переменный result будет размещено число 17. В данном случае команда alert(result) выполняет то же самое, что и alert(17). Иными словами, мы получаем выпадающее окно, в котором написано число 17.

2.2. Объектная модель HTML-страницы

Объектная модель документа (Document Object Model – DOM) является стандартом, предложенным веб-консорциумом, и регламентирует способ представления содержимого документа (в частности веб-страницы) в виде набора объектов. Под содержимым понимается все, что может находиться на веб-странице: рисунки, ссылки, абзацы, текст и т. д.

Объектная модель документа как объектная модель определяет:

– интерфейсы и объекты, используемые для представления документа и манипулирования с ним;

– семантику (смысл) этих интерфейсов и объектов, включая и поведение, и атрибуты;

– «родственные» связи и взаимодействие между этими интерфейсами и объектами.

В отличие от объектной модели браузера (BOM), которая уникальна для каждого браузера, объектная модель документа является стандартом и должна поддерживаться всеми браузерами. И хотя на практике поддержка DOM реализована не в полной мере, тем не менее необходимо стремиться следовать требованиям этого стандарта как производителям браузеров, так и разработчикам веб-сайтов.

Следует заметить, что DOM может применяться не только к веб-страницам, но и к любым другим документам. В частности, она может использоваться с любыми словарями XML, причем одним из таких словарей является HTML, а точнее, XHTML.

DOM является развивающимся стандартом и разбит на три уровня. Первый уровень является первой версией стандарта и пока что единственной законченной. Он состоит из двух разделов: первый является ядром и определяет принципы манипуляции со структурой документа (генерация и навигация), а второй посвящен представлению в DOM элементов HTML, определяемых одноименными тегами.

Второй и третий уровни описывают модель событий, дополняют таблицы стилей, проходы по структуре.

В DOM документ представляется в виде древовидной структуры (рис. 2.1), являющейся одной из наиболее употребительных структур в программировании. Это обеспечивает унифицированный способ навигации по документу.

```
<html>
<head>
     <tytle>Пример представления HTML-документа в виде дерева</tytle>
<head>
<body>
      <h1>Представление документа в виде дерева</h1>
      <p1>Абзац 1</p>
      <p1>Абзац 1</p>
</body>
</html>
```

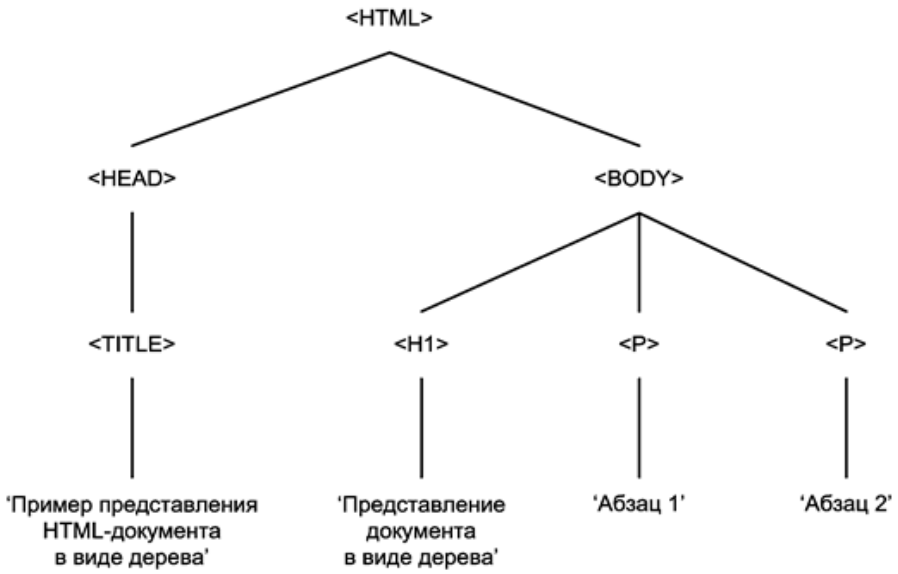

Рис. 2.1. Представление HTML-документа
в виде древовидной структуры

В модели DOM к элементу можно обратиться непосредственно по его идентификатору id, воспользовавшись методом getElementById объекта Document:

```
<html>
<head>
<title>Основы DOM</title>
</head>
<body>
<h1 id = "head">Основы DOM</h1>
<p>A Text</p>
<script language = "JavaScript">
var a = document.getElementById("head");
alert(a);
```

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

Для получения коллекции всех элементов, соответствующих какому-либо тегу, используется метод объекта Document – getElementsByTagName. Например,

```
var a = document.getElementsByTagName("TD")
```

присвоит переменной **a** коллекцию всех элементов <td>. Обратите внимание, что имя элемента следует писать прописными буквами ("TD").

Рассмотрим пример использования метода getElementsByTagName:

```
<html>
<head>
<title>Основы DOM</title>
</head>
<body>
<h1 id = "head">Основы DOM</h1>
<table border = "2">
<tr>
<td>1,1</td>
<td>1,2</td>
</tr>
<tr>
<td>2,1</td>
<td>2,2</td>
</tr>
</table>
<script language = "JavaScript">
var a = document.getElementsByTagName("TD");
a.item(0).style.color = "red";
a.item(3).style.fontFamily = "arial";
a.item(3).style.color = "green";
</script>
</body>
</html>
```

Чтобы воспользоваться преимуществом древовидной структуры, принятой в DOM для представления документа, следует использовать навигационные атрибуты (рис. 2.2), представленные в табл. 2.1.

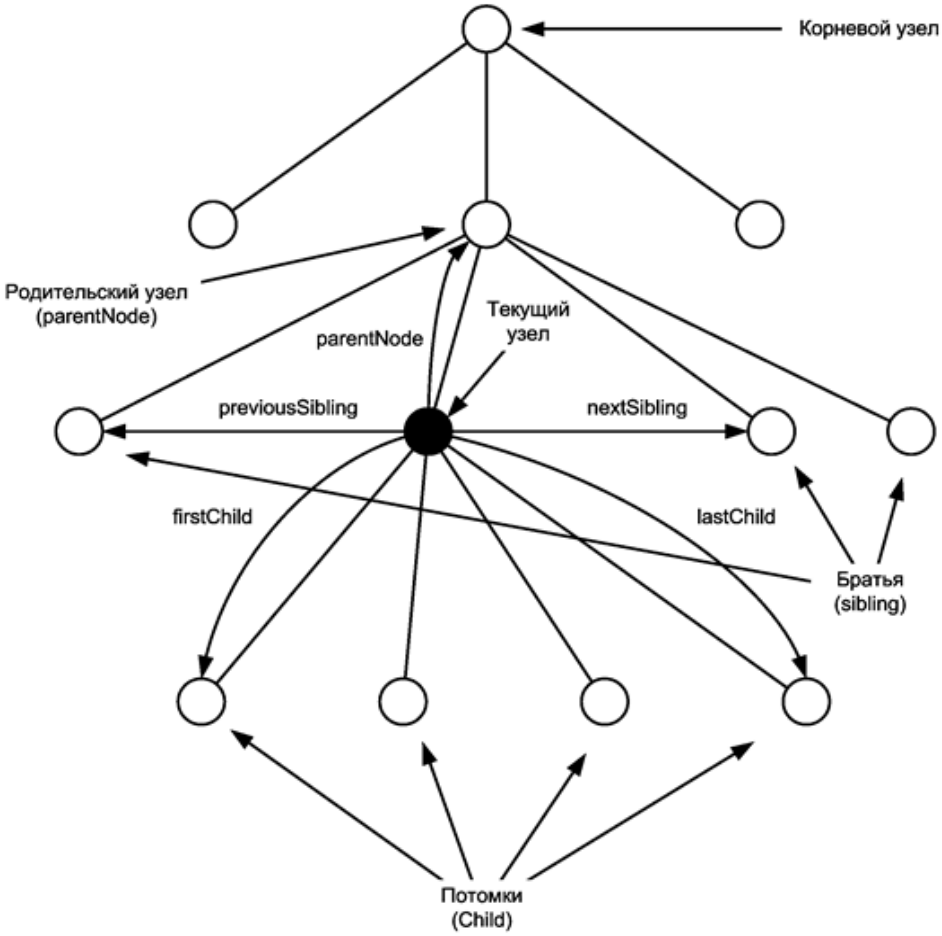

Рис. 2.2. Навигационные атрибуты объекта Node

Таблица 2.1

Навигационные атрибуты объекта Node

Атрибут	Описание
firstChild	Возвращает первый узел-потомок
lastChild	Возвращает последний узел-потомок
previousSibling	Возвращает предыдущий соседний узел, имеющий с текущим одного родителя
nextSibling	Возвращает следующий соседний узел, имеющий с текущим одного родителя
parentNode	Возвращает родительский узел
ownerDocument	Возвращает корневой узел документа, содержащий текущей узел
nodeName	Возвращает имя узла
nodeValue	Возвращает значение узла в текстовом формате
nodeType	Возвращает тип узла в виде числа

В следующем примере осуществляется проход по древовидной структуре документа:

```
<html>
<head>
<title>Навигация по документу</title>
</head>
<body>
<h1>Изучение навигации по документу</h1>
<p>Абзац 1</p>
<p>Абзац 2</p>
<script language = "JavaScript">
var temp = document.documentElement;
temp = temp.firstChild;
alert(temp.tagName);
if(temp.nextSibling == 3)
temp = temp.nextSibling.next.Sibling;
else
temp = temp.nextSibling;
alert(temp.tagName);
temp = temp.firstChild;
alert(temp.tagName);
temp.style.color = "red";
if(temp.nextSibling == 3)
temp = temp.nextSibling.nextSibling;
else
temp = temp.nextSibling;
alert(temp.tagName);
temp.style.color = "blue";
temp = temp.parentNode;
alert(temp.tagName);
</script>
</body>
</html>
```

Это динамическая генерация веб-страниц средствами DHTML на основе DOM.

Иногда требуется динамически формировать веб-страницы, например, в случае создания чатов, форумов, либо динамически создаваемых веб-страниц, содержимое которых хранится в базе данных. DOM позволяет решить такую задачу.

Для создания объектов у объекта Document имеются следующие методы, представленные в табл. 2.2.

Таблица 2.2

Методы объекта Document, позволяющие создавать объекты

Метод	Описание
createElement(имя_элемента)	Создает новый узел элемента с указанным именем
createTextNode(текст)	Создает текстовый узел с указанным текстом
createAttribute(имя_атрибута)	Создает новый узел атрибута с указанным именем

Вновь созданные объекты добавляются в структуру документа при помощи методов объекта Node (табл. 2.3).

Таблица 2.3

Методы объекта Node, добавляющие и удаляющие элементы документа

Метод	Описание
appendChild(новый_узел)	Добавляет объект Node в конец списка узлов-потомков
cloneNode(потомок-опция)	Создает объект Node, идентичный указанному в аргументе. В качестве аргумента можно использовать и все узлы-потомки одновременно
hasChildNodes()	Возвращает true, если узел имеет потомков
insertBefore(новый_узел, текущий_узел)	Вставляет объект Node в список потомков перед узлом, указанным в качестве второго параметра
removeChild(узел-потомок)	Удаляет узел-потомок, указанный в качестве параметра
replaceChild(новый_потомок, старый_потомок)	Заменяет старого потомка на нового

Приведем пример динамической генерации документа средствами DOM (рис. 2.3).

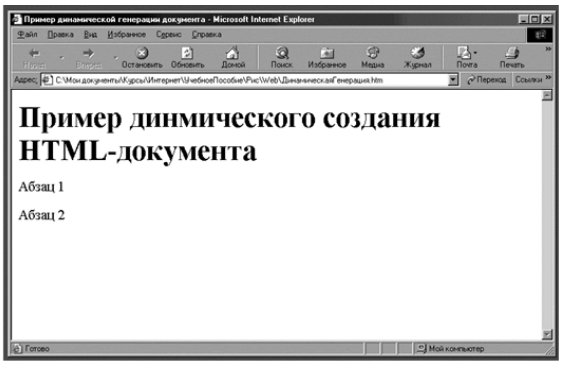

Рис. 2.3. Динамически сгенерированная веб-страница

```
<html>
<head>
<title>Пример динамической генерации документа</title>
</head>
<body>
<script language = "JavaScript">
    var newText;
    var newElem;
    newText = document.createTextNode("Пример
    динамического создания HTML-документа");
    newElem = document.createElement("H1");
    newElem.appendChild(newText);
    document.body.appendChild(newElem);
    newText = document.createTextNode("Абзац");
    newElem = document.createElement("P");
    newElem.appendChild(newText);
    document.body.appendChild(newElem);
</script>
</body>
</html>
```

Для чтения и установки атрибутов используются методы объекта Element (табл. 2.4).

Таблица 2.4

Методы объекта Element

Метод	Описание
getAttribute(имя_атрибута)	Возвращает значение атрибута
setAttribute(имя_атрибута, значение)	Устанавливает значение атрибута
removeAttribute(имя_атрибута)	Устанавливает значение атрибута по умолчанию, затирая текущее значение

Ниже приведен пример на задание атрибутов. И хотя применение атрибутов физического форматирования не рекомендовано к применению (для этих целей используются стили), они были выбраны в качестве примера, так как наглядно демонстрируют идею задания атрибутов методами DOM.

```
<html>
<head>
<title>Пример динамического создания HTML-документа</title>
</head>
<body>
<script language = "JavaScript">
    var newText;
    var newElem;
    newText = document.createTextNode("Пример динамического
    создания HTML-документа");
    newElem = document.createElement("H1");
    newElem.appendChild(newText);
    newElem.setAttribute("align", "center");
    document.body.appendChild(newElem);
    alert(newElem.getAttribute("align"));
    newText = document.createTextNode("Абзац");
    newElem = document.createElement("P");
    newElem.appendChild(newText);
    newElem.setAttribute("align", "right");
    document.body.appendChild(newElem);
    alert(newElem.getAttribute("align"));
    newElem.removeAttribute("align");
     </script>
</body>
</html>
```

2.3. Событийная модель DHTML. Применение DHTML

Особенностью программ, создаваемых для среды веб, является то, что они управляются *событиями.* Чтобы узнать, какое событие произошло, в DOM имеется объект события event (табл. 2.5). Объект event является локальным и его следует явным образом передавать в обработчик события.

Таблица 2.5

Свойства объекта event

Свойство	Описание
bubbles	Указывает возможность «всплывания» события (передачи управления вверх по иерархической структуре)
cancelable	Указывает возможность отмены действия события, заданного по умолчанию
currentTarget	Указывает событие, обрабатываемое в данный момент
eventPhase	Указывает фазу возбуждения события
target (только NN 6)	Указывает элемент, вызвавший событие
timestamp (только NN 6)	Указывает время возникновения события
type	Указывает имя события

Приведем пример динамически изменяемого текста.

```
<html>
<head>
<script language = "JavaScript">
      function onMouseover()
      {
      var temp = document.getElementById("DynamicText");
      temp.firstChild.nodeValue = "Указатель мыши на тексте";
      temp.style.color = "red";
      }
      function onMouseout()
      {
      var temp = document.getElementById("DynamicText");
      temp.firstChild.nodeValue = "Динамический текст";
      temp.style.color = "black";
      }
</script>
<title>Динамический текст</title>
</head>
<body>
<p id = "DynamicText" onmouseover = "return onMouseover()"
```

```
onmouseout = "return onMouseout()">Динамический текст</p>
</body>
<html>
```

Свойства объекта mouse приведены в табл. 2.6.

Таблица 2.6

Свойства объекта mouse

Свойство	Описание
altKey	Указывает, была ли нажата клавиша <Alt> в момент возникновения события
button	Указывает, какая клавиша мыши была нажата
clientX	Сообщает горизонтальную координату указателя мыши в окне браузера на момент возникновения события
clientY	Сообщает вертикальную координату указателя мыши в окне браузера на момент возникновения события
ctrKey	Указывает, была ли нажата клавиша <Ctrl> в момент возникновения события
metaKey	Указывает, была ли нажата метаклавиша в момент возникновения события
relatedTarget	Указывает цель события
screenX	Сообщает горизонтальную координату указателя мыши в окне браузера, вычисленную от начала экранных координат, на момент возникновения события
screenY	Сообщает вертикальную координату указателя мыши в окне браузера, вычисленную от начала экранных координат, на момент возникновения события
shiftKey	Указывает, была ли нажата клавиша <Shift> в момент возникновения события

Пример, определяющий координаты нахождения курсора мыши:

```
<html>
<head>
<title>Определение координат курсора</title>
<script language = "JavaScript">
        function onClick(e)
    {  alert("X = " + e.clientX + "; " + "Y = " + e.clientY);  }
```

```
</script>
</head>
<body onclick = "onClick(event)">
<p>Щелкните мышью на экране<p>
</body>
</html>
```

Глава 3. Программирование на стороне сервера

3.1. Заголовки запроса и ответа. CGI. Переменные окружения

CGI – (Common Gateway Interface – общий интерфейс маршрутизации) служит для обеспечения связи внешней прикладной программы с Web-сервером. Обычно Web-серверы возвращают статическую информацию, т. е. набираете некоторый URL (в своем браузере), в котором задаете имя ресурса. Web-сервер анализирует ваш запрос и отсылает вам то, что хотели (или сообщение об ошибке, если указанный вами ресурс недоступен), вот вообщем-то и весь World Wide Web. Во всем этом есть одна проблема: информация, которую посылает сервер, по своей природе статична. Сервер не может послать больше того, что у него есть на момент вашего запроса. Для того чтобы изменить эту ситуацию к лучшему, был введен стандарт CGI. CGI расширяет возможности Web-сервера тем, что информация, поступаемая от Web-сервера, приобретает динамический характер. Делается это следующим образом: если в URL (в броузере) указать не какой-нибудь статический ресурс, а специальную программу (CGI-скрипт), сервер, проанализировав такой запрос, возьмет и запустит ее. Программа, в свою очередь (пользуясь всеми ресурсами сервера), выдаст некоторую динамическую информацию (например, страничку). Web-сервер после окончания работы CGI-скрипта отправит эту информацию броузеру, предварительно снабдив ее нужным для протокола HTTP заголовком.

CGI-скрипт – это программа, поддерживающая интерфейс CGI. Она может быть написана на любом языке программирования (Perl, C/C++, Fortran, Unix Schell, Visual Basic, Delphi, Pascal, Apple Script, Java, Oberon), все будет зависеть от сервера, на котором будет работать данная программа. Как выполнимый модуль, она обычно располагается в директории /cgi-bin сервера или в директории /scripts (на самом деле это зависит от настроек Web-сервера, просто ему надо

знать, из какой директории надо запускать программы, а из какой пересылать находящиеся там данные). Существуют также такие CGI-скрипты, которые взаимодействуют с ППО (прикладным программным обеспечением) сервера (СУБД, электронные таблицы, деловая графика), а по результатам взаимодействия создают ответ и передают его серверу, который в свою очередь отсылает его пользователю. Такие CGI-скрипты называются *шлюзами*. Они выполняют роль переходника между пользователем и ППО сервера.

Рассмотрим структуру CGI. Каким образом сервер передает CGI-скрипту информацию об HTTP-запросе? Перед запуском CGI-скрипта Web-сервер создает для него переменные окружения (Environment variables) или как их еще называют переменные среды и записывает в них всю информацию о текущем запросе.

Переменные окружения CGI

REQUEST_METHOD – это одна из самых главных переменных, используемая для определения метода запроса HTTP.

Пример: REQUEST_METHOD=GET

QUERY_STRING – это строка запроса при методе GET.

Пример: QUERY_STRING= name=Vasya&age=19&hobby=games

CONTENT_LENGTH – длина в байтах тела запроса. Соответствует полю Content-Length в протоколе HTTP.

Пример: CONTENT_LENGTH=31

CONTENT_TYPE – MIME тип тела запроса (для форм кодированых вышеуказаным образом он application/x-www-form-urlencoded). Соответствует полю Content-Type в HTTP протоколе

GATEWAY_INTERFACE – версия протокола CGI.

Пример: GATEWAY_INTERFACE=CGI/1.1

REMOTE_ADDR – IP-Адрес клиента, делающего данный запрос.

Пример: REMOTE_ADDR=192.168.3.1

REMOTE_HOST – если запрашивающий клиент имеет доменное имя, то эта переменная содержит его, в противном случае – тот же самый IP-адресс, что и в REMOTE_ADDR.

Пример: REMOTE_HOST=uni-vologda.ac.ru

SCRIPT_NAME – имя скрипта, использованое в запросе. Для получения реального пути на сервере используйте SCRIPT_FILENAME.

Пример: SCRIPT_NAME=/~rva/guestbook.cgi

SCRIPT_FILENAME – имя файла скрипта на сервере.

Пример: SCRIPT_FILENAME=/students/rva/cgi-bin/guestbook.cgi

SERVER_NAME – имя сервера, чаще всего доменное как www.microsoft.com, но в редких случаях за неимением такового может быть IP-адресом как 192.168.3.1.

Пример: SERVER_NAME=www.vologda.ru

SERVER_PORT – TCP-Порт сервера, использующийся для соединения. По умолчанию HTTP-порт 80, хотя может быть в некоторых случаях быть другим.

Пример: SERVER_PORT=80

SERVER_PROTOCOL – версия протокола сервера.

Пример: SERVER_PROTOCOL=HTTP/1.1

SERVER_SOFTWARE – програмное обеспечение сервера.

Пример: Apache/1.0

AUTH_TYPE, REMOTE_USER – эти переменные определены в том случае, если запрошенный ресурс требует аутентификации пользователя.

Переменные заголовка HTTP-запроса

За исключением тех строк из заголовка HTTP-запроса, которые уже названы, сервер присоединяет строкам префикс HTTP_ и заменяет знаки '-' на '_':

HTTP_ACCEPT – давая запрос на сервер, браузер обычно расчитывает получить информацию определеного формата. Для этого он в заголовке запроса указывает поле Accept:. Отсюда скрипту поступает список тех MIME, которые браузер готов принять в качестве ответа от сервера.

Пример: HTTP_ACCEPT=text/html,text/plain,image/gif

HTTP_USER_AGENT – браузер обычно посылает на сервер и информацию о себе, чтобы, базируясь на знании особеностей и недостатков конкретных браузеров, CGI-скрипт мог выдать информацию с учетом этого. Например, разные браузеры могут поддерживать или не поддерживать какие-то HTML тэги.

Пример: HTTP_USER_AGENT=Mozila/2.01 Gold(Win95;I)

И другие переменные. Всего их около 30.

CGI-скрипт получает доступ к значениям этих переменых через функции операционной системы (в разных операционных системах это реализуется по-разному), тем самым CGI-скрипт получает исчерпывающую информацию об HTTP-запросе. А тело запроса поступает на STDIN (стандартный поток ввода) скрипта. Размер CONTENT_LENGTH – байт.

Заголовок ответа CGI программы

Возникает вопрос, каким образом CGI-скрипт должен посылать информацию Web-серверу с точки зрения CGI.

Все выводимые данные должны помещаться в STDOUT скрипта. При этом должен обязательно присутствовать CGI-заголовок (CGI-Header). В CGI-заголовоке могут быть следующие поля:

Content-Type: – должно обязательно присутствовать, если есть тело ответа. Определяет MIME-тип ответа.

Например: Content-Type:text/html

Location: – должно содержать URL-ресурса, на который перенаправляется запрос, как правило, в этом случае больше ничего не указывается (так как запрос перенаправляется на другой сервер).

Например: Location:http://www.idsoftware.com/index.html

Status: – содержит код завершения работы CGI-скрипта. Если не указан, то подразумевается 200 Ok.

Например: Status:404 Not found.

Далее после CGI заголовка на STDOUT посылается пустая строка, которая отделяет заголовок от тела ответа и посылается собственно тело ответа, тип которого был указан в Content-Type (рисунок, текст HTML или др.).

Web-сервер, получив через STDOUT информацию, поступившую от CGI-скрипта, формирует на базе ее HTTP-ответ, который и посылается клиенту (броузеру).

3.2. Создание форм. Методы передачи параметров (GET, POST)

PHP-сценарии обработки HTML-форм

В основном для передачи параметров из форм используются методы POST и GET. Главное отличие методов POST и GET заключается в способе передачи информации. В методе GET параметры передаются через адресную строку (URL), т. е. в HTTP-заголовке запроса, в то время как в методе POST параметры передаются через тело HTTP-запроса и никак не отражаются в адресной строке.

Кнопки – тег <BUTTON>

Тег <BUTTON> создает на веб-странице кнопки и по своему действию напоминает результат, получаемый с помощью тега <INPUT> (с параметром type="button | reset | submit"). В отличие от этого тега, <BUTTON> предлагает расширенные возможности по соз-

данию кнопок. Например, на подобной кнопке можно размещать любые элементы HTML, в том числе изображения. Используя стили, можно определить вид кнопки путем изменения шрифта, цвета фона, размеров и других параметров.

Теоретически тег <BUTTON> должен располагаться внутри формы, устанавливаемой элементом <FORM>. Тем не менее браузеры не выводят сообщение об ошибке и корректно работают с тегом <BUTTON>, если он встречается самостоятельно. Однако, если необходимо результат нажатия на кнопку отправить на сервер, помещать <BUTTON> в контейнер <FORM> обязательно. Закрывающий тег </BUTTON> обязателен.

Параметры:

Disabled – блокирует доступ и изменение элемента;

type – тип кнопки;

value – значение кнопки, которое будет отправлено на сервер или прочитано с помощью скриптов.

<button> Кнопка с текстом </button>

Параметр DISABLED блокирует доступ и изменение кнопки. В таком случае она отображается серой и недоступной для активации пользователем. Кроме того, такая кнопка не может получить фокус путем нажатия на клавишу Tab, мышью или другим способом. Тем не менее такое состояние кнопки можно изменять через скрипты.

```
<form>
<p><button> Активная кнопка </button>
<button disabled> Неактивная кнопка </button></p>
</form>
```

Параметр TYPE определяет тип кнопки, который устанавливает ее поведение в форме. По внешнему виду кнопки разного типа никак не различаются, но у каждой такой кнопки свои функции.

Значение по умолчанию: button.

Аргументы:

– button – обычная кнопка;

– reset – кнопка для очистки введенных данных формы и возвращения значений в первоначальное состояние;

– submit – кнопка для отправки данных формы на сервер.

```
<form method='post'>
<p><input type="text" name="user"></p>
<p><button type="reset">Очистить форму</button>
<button type="submit">Отправить форму</button></p>
```

```
</form>
<?php
    echo $_POST['user']; // Если выбран хоть 1 элемент
?>
```

Параметр VALUE определяет значение кнопки, которое будет отправлено на сервер. На сервер отправляется пара «имя=значение», где имя задается параметром name тега <BUTTON>, а значение – параметром value. Значение может как совпадать с текстом на кнопке, так быть и самостоятельным. Также параметр value применяется для доступа к данным через скрипты.

```
<form action="/html/example/handler.php">
 <p><input type="text" name="user"></p>
 <p><button value="11111010101" name="hidden" type="submit">
       Отправить форму</button></p>
</form>
```

Кнопка (input type=button)

```
<form>
 <input type=button name="save" value="Привет!">
</form>
```

Кнопка с изображением (input type=image)

```
<button>
<img src="/images/umbrella.gif" alt="Зонтик" style="vertical-align: middle">
```

Кнопка с рисунком

Кнопки с изображениями аналогичны по действию кнопке submit, но представляют собой рисунок. Для этого задаем type=image и src="image.gif".

```
<form>
 <input type="image" src="image.gif" name="sub" />
</form>
```

Когда пользователь щелкнет где-нибудь на изображении, соответствующая форма будет передана на сервер с двумя дополнительными переменными – sub_x и sub_y. Они содержат координаты нажатия пользователя на изображение. Опытные программисты могут заметить, что на самом деле имена переменных, отправленных браузером, содержат точку, а не подчеркивание, но PHP автоматически конвертирует точку в подчеркивание.

Кнопка отправки формы (input type=submit)

Служит для отправки формы сценарию. При создании кнопки для отправки формы необходимо указать 2 атрибута: type="submit" и value="Текст кнопки". Атрибут name необходим, если кнопка не одна, а несколько, и все они созданы для разных операций, например, кнопки «Сохранить», «Удалить», «Редактировать» и т. д. После нажатия на кнопку сценарию передается строка имя=текст кнопки.

<input type="submit" name="Submit" value='Отправить'>

PHP-сценарий не требуется.

Массив кнопок (submit) для выбора варианта действий

```
<form method='post'>
 <input type=submit name="save" value="first">
 <input type=submit name="save" value="pref">
 <input type=submit name="save" value="next">
 <input type=submit name="save" value="last">
</form>
<?php
$action = $_POST['save'];
switch ($action)
 {
  case 'first': echo 'Первый';      break;
  case 'pref' : echo 'Предыдущий'; break;
  case 'next' : echo 'Следующий';  break;
  case 'last' : echo 'Последний';  break;
 }
?>
```

Кнопка сброса формы (Reset)

При нажатии на кнопку сброса (reset) все элементы формы будут установлены в то состояние, которое было задано в атрибутах по умолчанию, причем отправка формы не производится.

<input type="reset" name="Reset" value="Очистить форму">

PHP-сценарий не требуется.

Флажок (checkbox)

Флажки checkbox предлагают пользователю ряд вариантов и разрешают произвольный выбор (ни одного, одного или нескольких из них).

```
<form method='post'>
 <input type='checkbox' name='chb[0]' value="white">Белый<br>
 <input type='checkbox' name='chb[1]' value="green">Зеленый<br>
 <input type='checkbox' name='chb[2]' value="blue">Синий<br>
 <input type='checkbox' name='chb[3]' value="red">Красный<br>
 <input type='checkbox' name='chb[4]' value="black">Черный<br>
 <input type='submit' name="Submit" value='Отправить'>
</form>
<?php
if (!empty($_POST['chb']))
 {
  $chb = $_POST['chb'];
  foreach($chb as $index => $go)
   {
    echo $index." – > ".$go."<br>";
   };
 };
```

Переключатель (radio)

Переключатели radio предлагают пользователю ряд вариантов, но разрешает выбрать только один из них.

Пример 1

```
<form method='post'>
 <input name="mycolor" type="radio" value="white">Белый<br>
 <input name="mycolor" type="radio" value="green" checked>Зеленый<br>
 <input name="mycolor" type="radio" value="blue">Синий<br>
 <input name="mycolor" type="radio" value="red">Красный<br>
 <input name="mycolor" type="radio" value="black">Черный<br>
 <input type='submit' name="Submit" value='Отправить'>
</form>
<?php echo $_POST['mycolor'];
?>
```

Пример 2

```
<form method=GET>
 // первый набор кнопок
 <input type='radio' name='rdi[0]' value='1'>
 <input type='radio' name='rdi[0]' value='2'>
```

```
 <input type='radio' name='rdi[0]' value='3'><br>
 // второй набор кнопок
 <input type='radio' name='rdi[1]' value='1'>
 <input type='radio' name='rdi[1]' value='2'>
 <input type='radio' name='rdi[1]' value='3'><br>
 // третий набор кнопок
 <input type='radio' name='rdi[2]' value='1'>
 <input type='radio' name='rdi[2]' value='2'>
 <input type='radio' name='rdi[2]' value='3'><br>
 <input type='submit' value='Отправить'>
</form>
<?php
$rdi = $_GET['rdi'];
while(list($key,$val) = @each($rdi))
 echo "ключ – $key, значение – $val<br>\n";
?>
```

Текстовое поле (text)

При создании обычного текстового поля размером size и максимальной допустимой длины maxlength символов, атрибут type принимает значение text. Если указан параметр value, то поле будет отображать указанный в переменной value. При создании поля не забывайте указывать имя поля, так как этот атрибут является обязательным.

```
<form method='post'>
 <input type="text" name="txtName" size="40" maxlength="35" value="Текст по умолчанию">
 <input type='submit' name="Submit" value='Отправить'>
</form>
<?php echo $_POST['txtName']; ?>
```

Поле для ввода пароля (password)

Полностью аналогичен текстовому полю, за исключением того, что символы, набираемые пользователем, не будут отображаться на экране.

```
<form method='post'>
 <input type="password" name="txtName" size="40" maxlength="35">
 <input name="Submit" type='submit' value='Отправить'>
</form>
```

```
<?php echo $_POST['txtName']; ?>
```

Скрытое текстовое поле (hidden)

Позволяет передавать сценарию необходимую служебную информацию, не отображая ее на странице.

```
<form method='post'>
 <input type="hidden" name="txtName" value="Это скрытый текст">
 <input name="Submit" type='submit' value='Отправить'>
</form>
<?php echo $_POST['txtName']; ?>
```

Выпадающий список (select)

Тэг <select> представляет собой выпадающий или раскрытый список, при этом одновременно могут быть выбраны одна или несколько строк, но будет передано значение последней выбранной кнопки.

Список начинается с парных тегов <select> </select>. Теги <option> </option> позволяют определить содержимое списка, а параметр value определяет значение строки. Если в теге <option> указан параметр selected, то строка будет изначально выбранной. Параметр size задает, сколько строк будет занимать список. Если size равен 1, то список будет выпадающим. Если указан атрибут multiple, то разрешено выбирать несколько элементов из списка. Но эта схема практически не используется, а при size = 1 не имеет смысла.

```
<form method='post'>
    <select name="color"> <!--выпадающий список-->
    <!--<select name="color" size=3> <!--список с прокруткой-->
    <option value="white">Белый</option>
    <option value="green">Зеленый</option>
    <option value="blue">Синий</option>
    <option value="red">Красный</option>
    <option value="black">Черный</option>
    </select>
    <input name="Submit" type='submit' value='Отправить'>
</form>
<?php
f (!empty($_POST['color'])) { echo $_POST['color']; };
?>
```

Предположим, необходимо создать выпадающий список с предсказуемой последовательностью, например, список с годами с 2000 по 2050.

В данном случае используется следующий прием:

```
<form method='post'>
            <select name=years>
            <?php
            $year = 2000;
            for ($i = 0; $i <= 50; $i++) // Цикл от 0 до 50
             {
              $new_years = $year + $i; // Формируем новое значение
              echo                                                    '<option
    value='.$new_years.'>'.$new_years.'</option>'; //строка
             }
            ?>  </select>
             <input name="Submit" type='submit' value='Отправить'>
</form>
<?php
    if (!empty($_POST['years']))
     {   echo $_POST['years'];   };
?>
```

Многострочное поле ввода текста (textarea)

Многострочное поле ввода текста позволяет отправлять не одну строку, а сразу несколько. При необходимости можно указать атрибут readonly, который запрещает редактировать, удалять и изменять текст, т. е. текст будет предназначен только для чтения. Если необходимо, чтобы текст был изначально отображен в многострочном поле ввода, то его необходимо поместить между тэгами <textarea> </textarea>.

Существует параметр wrap – задание переноса строк. Возможные значения:

– off – отключает перенос строк;

– virtuals – показывает переносы строк, но отправляет текст как он введен;

– physical – переносы строк оставляются в исходном виде.

По умолчанию тег <textarea> создает пустое поле шириной в 20 символов, состоящее из двух строк.

```
<form method='post'>
 <textarea name="text" cols="40" rows="5">
```

```
Первоначально вставленный текст
</textarea>
<br>
<!--
<textarea name="text" cols="40" rows="5" readonly>
Первоначально вставленный текст
</textarea>
<br>
-->
<input name="Submit" type='submit' value='Отправить'>
</form>
<?php echo $_POST['text']; ?>
```

Для того чтобы в многострочном текстовом поле соблюдалось html-форматирование (перенос строк по средством тега
 или <br\>), используйте функцию nl2br():

```
<form method='post'>
<textarea name="text" cols="40" rows="5">
Первоначально вставленный строка 1
Первоначально вставленный строка 2
Первоначально вставленный строка 3
</textarea>
<input name="Submit" type='submit' value='Отправить'>
</form>
<?php echo nl2br($_POST['text']); ?>
```

Кнопка для загрузки файлов (browse)

Служит для реализации загрузки файлов на сервер. При создании текстового поля также необходимо указать тип поля type как "file".

```
<form method="post">
Загрузить файл:<br>
<input name="filename" type="file"><br>
<input type="submit" value="Отправить">
</form>
<?php echo ($_POST['filename']); ?
```

Способов, предоставляемых протоколом HTTP, немного. Это важная информация. Никаких других способов нет. На практике используются два: GET – это когда данные передаются в адресной стро-

ке, например, когда пользователь жмет ссылку. POST – когда он нажимает кнопку в форме.

Метод GET

Чтобы передать данные методом GET, не надо создавать на HTML-странице форму (использовать формы для запросов методом GET не рекомендуется) – достаточно ссылки на документ с добавлением строки запроса, которая может выглядеть как переменная=значение, пары объединяются с помощью амперсанда «&», а к URL страницы строка присоединяется с помощью вопросительного знака «?».

Но можно не использовать пары ключ=значение, если необходимо передать всего одну переменную. Для этого надо после знака вопроса написать ЗНАЧЕНИЕ (не имя) переменной.

Преимущество передачи параметров таким способом заключается в том, что клиенты, которые не могут использовать метод POST (например, поисковые машины), все же смогут просто, пройдя по ссылке, передать параметры скрипту и получить содержимое.

Недостаток в том, что, просто изменив параметры в адресной строке, пользователь может повернуть ход сценария непредсказуемым образом, это создает огромную дыру в безопасности в сочетании с неопределенными переменными и register_globals on или кто-нибудь может узнать значение важной переменной (например ID-сессии), просто посмотрев на экран монитора.

Для чего следует использовать метод GET:

– для доступа к общедоступным страницам с передачей параметров (повышение функциональности);

– передача информации, не влияющей на уровень безопасности.

Для чего не следует использовать метод GET:

– для доступа к защищенным страницам с передачей параметров;

– для передачи информации, влияющей на уровень безопасности;

– для передачи информации, не подлежащей модифицированию пользователем (некоторые передают текст SQL-запросов).

Метод POST

Передать данные методом POST можно только с помощью формы на HTML странице. Основное отличие POST от GET в том, что данные передаются не в заголовке запроса, а в теле, следовательно, пользователь их не видит. Модифицировать можно только изменив саму форму.

Преимущество:

– большая безопасность и функциональность запросов с помощью форм методом POST.

Недостаток:

– меньшая доступность.

Для чего следует использовать метод POST:

– для передачи большого объема информации (текст, файлы);

– для передачи любой важной информации;

– для ограничения доступа (например, использовать для навигации только форму – возможность, доступная не всем программам-роботам или грабберам-контента).

3.3. HTTP-cookie. Атрибуты cookie. Сессии. Идентификация пользователя

PHP поддерживает HTTP-cookie. Формально, cookie – это небольшой фрагмент данных, которые веб-браузер пересылает веб-серверу в HTTP-запросе при каждой попытке открыть очередную страницу сайта. Обычно cookie создаются веб-сервером и присылаются в браузер при первом запросе к сайту. Cookie также могут быть созданы самой загруженной web-страницей. Далее хранятся на компьютере пользователя в виде текстового файла до тех пор, пока либо не закончится их срок, либо они будут удалены скриптом или пользователем.

На практике cookie обычно используются для:

– аутентификации пользователя;

– хранения персональных предпочтений и настроек пользователя;

– отслеживания состояния сессии доступа пользователя;

– ведения статистики о пользователях.

Cookie могут использоваться сервером для опознания ранее аутентифицированных пользователей. Это происходит следующим образом.

Пользователь вводит имя пользователя и пароль в текстовых полях страницы входа и отправляет их на сервер.

Сервер получает имя пользователя и пароль, проверяет их и, при их правильности, отправляет страницу успешного входа, прикрепив cookie с неким идентификатором сессии. Эта cookie может быть действительна не только для текущей сессии браузера, но может быть настроена и на длительное хранение.

Каждый раз, когда пользователь запрашивает страницу с сервера, браузер автоматически отправляет cookie с идентификатором сес-

сии серверу. Сервер проверяет идентификатор по своей базе идентификаторов и, при наличии в базе такого идентификатора, «узнает» пользователя.

Этот метод широко используется на многих сайтах, например, на Yahoo!, в Википедии и в Facebook.

Многие браузеры (в частности, Opera, FireFox), путем редактирования свойств cookie, могут управлять поведением веб-сайтов. Изменив срок истечения непостоянных (сессионных) cookie можно, например, получить формально-неограниченную сессию после авторизации на каком-либо сайте. Возможность редактирования cookie стандартными средствами отсутствует в Internet Explorer. Но, воспользовавшись иными механизмами, например, JavaScript, пользователь может изменить cookie-файл. Более того, существует возможность заменить сессионные cookie постоянными (с указанием срока годности).

Однако серверное программное обеспечение может отслеживать такие попытки. Для этого сервер выдает cookie на определенный срок и записывает дату окончания cookie у себя каждый раз, когда пользователь обращается к серверу. Если cookie, присланный браузером, имеет дату годности отличную от той, что хранится на сервере, значит, имеет место попытка подмены даты годности cookie. Сервер может отреагировать, например, запросив у пользователя повторную авторизацию.

Устанавливаются cookie функцией setcookie(). Cookie являются частью HTTP header'a, поэтому функция setcookie() обязана вызываться до отправления любого вывода браузеру. Это то же самое ограничение, что и для header(). Можно использовать функции буферизации вывода для задержки вывода скрипта до тех пор, пока не определите, устанавливать ли cookie и отправлять ли какие-нибудь header'ы.

Любая cookie, отправляемая вам с клиента, будет автоматически конвертирована в PHP-переменную точно так же, как GET и POST-данные, в зависимости от переменных конфигурации register_globals и variables_order. Если необходимо присвоить несколько переменных одной cookie, просто добавьте [] в имя cookie.

В PHP 4.1.0 и позднее автоглобальный массив $_COOKIE всегда будет установлен любой cookie, отправленной клиентом. $HTTP_COOKIE_VARS также устанавливается в более ранних версиях PHP, когда установлена переменная конфигурации track_vars.

Синтасис функции setcookie **(PHP 3, PHP 4):**

int setcookie (string name [, string value [, int expire [, string path [, string domain [, int secure]]]]])

Функция setcookie() определяет cookie, посылаемую наряду с другой информации в HTML заголовке. Cookies должны быть посланы до отправления других заголовков (это особенность cookies, а не PHP). Поэтому эта функция должна быть вызвана до каких-либо тегов. Вызов этой функции необходимо помещать перед тэгами <html> или <head>.

Все аргументы, кроме name, являются необязательными. Если имеется только аргумент name, cookie с этим именем будет удалена с удаленного клиента. Заместить любой аргумент можно пустой строкой (""). Аргументы expire и secure – это целые числа (integer), и они не могут быть пропущены с помощью пустой строки. В них используйте нуль (0). Аргумент expire – это обычное Unix time integer, возвращаемое функциями time() или mktime(). Аргумент secure указывает, что данная cookie должна передаваться только через секретное HTTPS-соединение.

После того как cookie установлены, доступ к ним может быть получен при загрузке следующей страницы через массив $_COOKIE (который вызывается $HTTP_COOKIE_VARS в версиях PHP до 4.1.0).

Особенности

Cookie будут невидимы до тех пор, пока не будет загружена следующая страница.

Cookie обязаны быть удалены с теми же параметрами, с которыми были установлены.

В PHP 3 множественные вызовы setcookie() в том же скрипте могут быть выполнены в реверсном порядке. Если вы пытаетесь удалить одну cookie до вставки другой, вы должны сделать вставку до удаления. В PHP 4 множественные вызовы setcookie() выполняются в порядке вызова.

Рассмотрим примеры отправки cookie.

Отправка кук функцией setcookie()

```
setcookie ("TestCookie", $value);
setcookie ("TestCookie", $value,time()+3600);
                    /*период действия – 1 час */
setcookie ("TestCookie", $value,time()+3600, "/~rasmus/",
".utoronto.ca", 1);
```

При удалении cookie вы должны убедиться, что дата окончания действия прошла, чтобы переключить механизм в вашем браузере.

Удаление cookie с помощью setcookie()

```
// установить дату окончания действия на один час назад
setcookie ("TestCookie", "", time() – 3600);
setcookie ("TestCookie", "", time() – 3600, "/~rasmus/", ".utoronto.ca", 1);
```

Для просмотра содержимого тестовой cookie в скрипте можно использовать один из следующих примеров:

```
echo $TestCookie;
echo $_COOKIE["TestCookie"];
```

Можно также установить cookie массива, используя нотацию в имени cookie. Это дает эффект установки стольких cookie, сколько элементов в этом массиве, но, когда cookie получается скриптом, значения помещаются в массив с именем cookie:

```
setcookie ("cookie[three]", "cookiethree");
setcookie ("cookie[two]", "cookietwo");
setcookie ("cookie[one]", "cookieone");
```

Глава 4. Программирование на стороне сервера на PHP

4.1. Введение в PHP

Главная область применения PHP – написание скриптов, работающих на стороне сервера. Таким образом, PHP способен выполнять все то, что выполняет любая другая программа CGI, например, обрабатывать данные форм, генерировать динамические страницы или отсылать и принимать cookies. Но PHP способен выполнять намного больше.

Существуют три основных области применения PHP:

1. Создание скриптов для выполнения на стороне сервера. PHP традиционно и наиболее широко используется именно таким образом. Для этого вам будут необходимы три вещи. Интерпретатор PHP (в виде программы CGI или серверного модуля), веб-сервер и браузер. Для того чтобы можно было просматривать результаты выполнения PHP-скриптов в браузере, нужен работающий веб-сервер и установленный PHP. Просмотреть вывод PHP-программы можно в браузере, получив PHP-страницу, сгенерированную сервером. В случае, если вы просто экспериментируете, вполне можете использовать свой домашний компьютер вместо сервера.

2. Создание скриптов для выполнения в командной строке. Можно создать PHP-скрипт, способный запускаться без сервера или браузера. Все, что вам потребуется, – парсер PHP. Такой способ использования PHP идеально подходит для скриптов, которые должны выполняться регулярно, например, с помощью cron (на платформах *nix или Linux) или с помощью планировщика задач (Task Scheduler) на платформах Windows. Эти скрипты также могут быть использованы в задачах простой обработки текстов.

3. Создание оконных приложений, выполняющихся на стороне клиента. Возможно, PHP является не самым лучшим языком для создания подобных приложений, но, если вы очень хорошо знаете PHP и хотели бы использовать некоторые его возможности в своих клиентских приложениях, вы можете использовать PHP-GTK для создания таких приложений. Подобным образом вы можете создавать и кроссплатформенные приложения. PHP-GTK является расширением PHP и не поставляется вместе с основным дистрибутивом PHP.

PHP доступен для большинства операционных систем, включая Linux, многие модификации Unix (такие как HP-UX, Solaris и OpenBSD), Microsoft Windows, Mac OS X, RISC OS и др. Также в PHP включена поддержка большинства современных веб-серверов, таких как Apache, IIS и др. В принципе, подойдет любой веб-сервер, способный использовать бинарный файл FastCGI PHP, например, lighttpd или nginx. PHP может работать в качестве модуля или функционировать в качестве процессора CGI.

Таким образом, выбирая PHP, вы получаете свободу выбора операционной системы и веб-сервера. Более того, у вас появляется выбор между использованием процедурного или объектно-ориентированного программирования (ООП) или же их сочетания.

PHP способен генерировать не только HTML. Доступно формирование изображений, файлов PDF и даже роликов Flash (с использованием libswf и Ming), создаваемых «на лету». PHP также способен генерировать любые текстовые данные, такие как XHTML и другие XML-файлы. PHP может осуществлять автоматическую генерацию таких файлов и сохранять их в файловой системе вашего сервера вместо того, чтобы отдавать клиенту, организуя таким образом серверный кэш для вашего динамического контента.

Одним из значительных преимуществ PHP является поддержка широкого круга баз данных. Создать скрипт, использующий базы данных, – невероятно просто. Можно воспользоваться расширением,

специфичным для отдельной базы данных (таким как mysql) или использовать уровень абстракции от базы данных, такой как PDO, или подсоединиться к любой базе данных, поддерживающей Открытый Стандарт Соединения Баз Данных (ODBC), с помощью одноименного расширения ODBC. Для других баз данных, таких как CouchDB, можно воспользоваться с URL или сокетами.

PHP также поддерживает «общение» с другими сервисами через такие протоколы, как LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (на платформах Windows) и др. Кроме того, существует возможность работать с сетевыми сокетами напрямую. PHP поддерживает стандарт обмена сложными структурами данных WDDX практически между всеми языками веб-программирования. Обращая внимание на взаимодействие между различными языками, следует напомнить о поддержке объектов Java и возможности их использования в качестве объектов PHP.

PHP имеет много возможностей по обработке текста, включая регулярные выражения Perl (PCRE) и много других расширений и инструментов для обработки и доступа к XML документам. В PHP обработка XML-документов стандартизирована и происходит на базе мощной библиотеки libxml2, расширив возможности обработки XML добавлением новых расширений SimpleXML, XMLReader и XMLWriter. Есть еще много других расширений, которые можно просмотреть как в алфавитном порядке, так и по категориям.

4.2. Основы синтаксиса

В отличие от традиционных скриптовых языков (таких как Perl), PHP-программа представляет собой HTML-страницу со вставками кода.

Для сравнения:

Perl-скрипт:

```
#!/usr/local/bin/perl
print "Content-type: text/html\n\n";
print "<html>\n<head><title>Hello World</title></head>\n";
print "<body><h1>Hello World!</h1></body>\n";
print "</html>";
```

PHP-скрипт:

```
<html>
<head><title>Hello World</title></head>
```

```
<body><h1>Hello World!</h1></body>
</html>
```

Как видите, простейшая программа на PHP – это обычная HTML-страница. О выводе заголовка *Content-type: text/html* PHP тоже позаботился самостоятельно.

Непосредственно PHP-код (который – не HTML) размещается между тэгами **<?** и **?>**. Вообще универсальный (т. е. гарантированно работающий при любой конфигурации PHP), но более длинный способ спецификации PHP-кода – тэги **<?php** ... **?>**. Такая длинная форма записи используется при совмещении XML и PHP, так как тэг **<?** ... **?>** используется в стандарте XML. За распознавание тэга **<?** как начала PHP-блока отвечает директива short_open_tag файла php.ini (по умолчению – включена). Если вы хотите разрабатывать скрипты, работающие независимо от данной настройки, используйте длинный открывающий тэг <?php. В дальнейшем будем использовать сокращенную форму.

Рассмотрим простой пример.

```
<html>
<head><title>Hello World</title></head>
<body><h1>Hello World!</h1>
<p>Текущая дата:
<?
  echo date("d.m.Y");
?>
</body>
</html>
```

Для выполнения примеров скопируйте их в файл, расположенный в каталоге, соответствующий директиве DocumentRoot файла конфигурации Apache httpd.conf (например, в файл с именем test.php), и выполните их, обратившись к сохраненному скрипту (test.php) из адресной строки браузера (http://localhost/test.php). Apache + PHP должен быть установлен.

Если сегодня – 27 апреля 2013 г., в результате исполнения скрипта браузер получит следующий HTML-код:

```
<html>
<head>
<title>Hello World  </title>  </head>
<body>
   <h1>Hello World!</h1>
   <p>Текущая дата:
```

```
    27.04.2013
</body>
</html>
```

Строки 5–7 предыдущего листинга программы – вставка PHP-кода. На строках 5 и 7 расположены, соответственно, открывающий и закрывающий тэг. Их совершенно необязательно располагать на отдельных строках – это сделано по соображениям удобства чтения.

В строке 6 расположен оператор **echo**, используемый для вывода в браузер. Выводит же он результат выполнения функции **date** – в данном случае это текущая дата.

Строка 6 является законченным выражением. Каждое выражение в PHP заканчивается точкой с запятой – **;**. Именно точкой с запятой, а не переводом строки – не забывайте об этом, особенно если вы раньше программировали на Visual Basic или ASP.

Заметим, что тэг </body> расположен на той же строке, что и текст, сформированный функцией date(), хотя в исходном коде </body> находится на отдельной строке. Дело в том, что PHP отбрасывает перевод строки, следующий сразу после закрывающего тэга ?> – это сделано специально, чтобы в фрагментах HTML, где лишние пробелы нежелательны, не было необходимости жертвовать читабельностью скрипта, записывая закрывающий PHP-тэг на одной строке с последующим HTML-кодом. Если же пробел необходим – вставьте после ?> пустую строку.

Переменные в PHP начинаются со знака **$**, за которыми следует произвольный набор латинских букв, цифр и знака подчеркивания: _, при этом цифра не может следовать сразу за знаком **$**.

Регистр букв в имени переменной имеет значение: $A и $a – это две разные переменные.

Для присваивания переменной значения используется оператор равно «=».

Пример:

```
<?
    $a = 'test';
    $copyOf_a = $a;
    $Number100 = 100;
    echo $a;
    echo $copyOf_a;
    echo $Number100;
?>
```

Данный код выведет: testtest100.

Следите за тем, какие имена даете переменным: вряд ли вы через полгода вспомните, для чего используется переменная $a21 или $zzz. А вот для чего используется переменная $username, понять довольно легко.

В строке 2 переменной *$a* присваивается строковое значение 'test'. Строки в PHP записываются в кавычках – одинарных или двойных (различие между записями в неодинаковых кавычках мы рассмотрим чуть позже). Также справедливо высказывание, что переменная *$a* инициализируется значением 'test': в PHP переменная создается при первом присваивании ей значения; если переменной не было присвоено значение – переменная не определена, т. е. ее просто не существует.

В строке 3 переменная *$copyOf_a* инициализируется значением переменной $a; в данном случае (смотрим строку 2) это значение – строка 'test'. В строке с номером 4 переменной с именем *$Number100* присваивается числовое значение 100.

Как видите, в PHP существует типизация, т. е. язык различает типы данных – строки, числа и т. д. Однако при этом PHP является языком со слабой типизацией – преобразования между типами данных происходят автоматически по установленным правилам. Например, программа *<? echo '100' + 1; ?>* выведет число 101: строка автоматически преобразуется в число при использовании в числовом контексте (в данном случае – строка '100', при использовании в качестве слагаемого преобразуется в число 100, так как операция сложения для строк не определена).

Такое поведение (а также отсутствие необходимости или возможности явно определять переменные) роднит PHP с Perl и Basic, однако, возможно, будет встречено «в штыки» приверженцами строгих языков, таких как Си и Pascal. Конечно, необходимость четкого определения переменных и требование явного приведения типов уменьшает число возможных ошибок программирования, однако PHP, прежде всего, – интерпретируемый язык для быстрой разработки скриптов, и нестрогость синтаксиса с избытком компенсируется скоростью кодинга. А о неинициализированной переменной PHP всегда услужливо сообщит – если, конечно, ему этого не запрещать.

Рассмотрим пример:

```
<?
    $greeting = 'Привет';
```

```
    $name = 'Вася';
    $message = "$greeting, $name!";
    echo $message;
?>
```

Особого внимания заслуживает четвертая строка. Внутри двойных кавычек указаны переменные, определенные в предыдущих строках. Если выполнить эту программу, в окне браузера отобразится строка *Привет*, *Вася*!. В этом и заключается основная особенность двойных кавычек: имена переменных, указанных внутри пары символов **"**, заменяются на соответствующие этим переменным значения.

Помимо этого внутри двойных кавычек распознаются специальные управляющие комбинации, состоящие из двух символов, первый из которых – обратный слэш (\). Наиболее часто используются следующие управляющие символы:

r – возврат каретки (CR);

n – перевод строки (NL);

" – двойная кавычка;

*$* – символ доллара ($);

\\ – собственно, обратный слэш (\).

Символы \r и \n обычно используются вместе, в виде комбинации \r\n – так обозначается перевод строки в Windows и многих TCP/IP-протоколах.

Оставшиеся три пункта из приведенного списка применения обратного слэша являются примерами экранирования – отмены специального действия символа. Так, двойная кавычка обозначала бы конец строки, символ доллара – начало имени переменной, а обратный слэш – начало управляющей комбинации. При экранировании символ воспринимается «как он есть», и никаких специальных действий не производится.

Экранирование (не обратный слэш, а сам принцип) в PHP используется во многих случаях.

Если в данном случае кавычки заменить на одинарные, в браузере отобразится именно то, что внутри них написано ($greeting, $name!). Комбинации символов, начинающиеся с \, в одинарных кавычках также никак не преобразуются, за двумя исключениями: **\'** – одинарная кавычка внутри строки; \\ – обратный слэш.

Немного изменим наш последний пример:

```
<?
    $greeting = 'Привет';
```

```
    $name = 'Вася';
    $message = $greeting . ',' . $name. '!';
    echo $message;
?>
```

В данном случае мы не стали пользоваться двойными кавычками: в строке 4 имена переменных и строковые константы записаны через оператор конкатенации (объединения строк). В PHP конкатенация обозначается точкой – «**.**». Результат выполнения этой программы аналогичен предыдущему примеру.

Рекомендуется использовать именно этот способ записи – на то есть достаточно причин:

– имена переменных более четко визуально отделены от строковых значений, что лучше всего заметно в редакторе с подсветкой кода;

– интерпретатор PHP обрабатывает такую запись немного быстрее;

– PHP сможет более четко отследить опечатку в имени переменной.

Вы не совершите ошибку, подобную следующей: *$message = "$greetingVasya"* – PHP в данном случае выведет не *"ПриветVasya"*, а пустую строку, так как *$greetingVasya* распознается как имя переменной, а таковой у нас нет.

Однако двойные кавычки весьма популярны, и вы их встретите во множестве скриптов, доступных в сети.

Кстати, в последних двух примерах совершенно нет необходимости в определении переменной $message: строки 4 и 5 можно сократить до *echo $greeting . ',' . $name . '!';*. А если переменная $message нам может понадобиться в нижеследующем коде – можно написать *echo $message = $greeting . ',' . $name . '!';*, и это сработает. Связано это с тем, что результатом выражения, содержащего присваивание, является присвоенное значение. Это особенно удобно при присваивании одного и того же значения нескольким переменным. Например, если переменным $a и $b нужно присвоить одно и то же значение (предположим, число с плавающей запятой 10.34), можно написать *$a = $b = 10.34;*.

В PHP предусмотрено множество встроенных функций для работы со строками. Описание их можно найти в официальной документации.

Помимо строк и чисел существует еще один простой, но важный тип данных – булевый (bool), к которому относятся два специальных значения: true (истина) и false (ложь). При автоматическом приведении типов, false соответствует числу 0 и пустой строке (**''**), true – все-

му остальному. Булевые значения часто применяются совместно с условными операторами, о которых мы поговорим дальше.

4.3. Управляющие конструкции

Условные операторы

if

Часто возникает необходимость выполнения разного кода в зависимости от определенных условий. Рассмотрим пример:

```
<?
    $i = 10;
    $j = 5 * 2;
    if ($i == $j)
    echo 'Переменные $i и $j имеют одинаковые значения';
    else
    echo 'Переменные $i и $j имеют различные значения';
?>
```

Здесь используется оператор if...else – условный оператор.

```
        В общем виде он выглядит так:
        if (условие)
      выражение_1;
 else
      выражение_2;
```

В приведенном примере условием является результат сравнения значений переменных $i и $j. Оператор сравнения «==» – два знака равенства. Поскольку 5 * 2 равняется 10, и, соответственно, 10 равняется 10, выполнится строка 5, и мы увидим, что переменные имеют равные значения. Измените, например, строку 2 на **$i = 11**, и вы увидите, что выполнится оператор echo из строки 7 (так как условие ложно). Помимо «==» есть и другие операторы сравнения:

!= – не равно;

< – меньше;

> – больше;

<= – меньше или равно;

>= – больше или равно.

Можно поэкспериментировать, изменяя оператор сравнения и значения переменных. (Для логической правильности вывода на экран потребуется, конечно, изменить и тексты, выводимые операторами echo.)

Не путайте оператор сравнения «==» с оператором присваивания «=». Если вы допустите такую ошибку, условие всегда будет верным, если присваивается значение, соответствующее булевому true, и всегда ложным – если значение соответствует false.

Если требуется выполнить действие, что **if** условие выполняется, блок **else ...** можно опустить:

```
<?
    $i = 10;
    $j = 5 * 2;
    if ($i == $j)
    echo 'Переменные $i и $j имеют одинаковые значения';
?>
```

В этом случае, если условие ложно, в браузер не выведется ничего.

Отступы перед строками **echo ...** сделаны для удобства чтения.

Рассмотрим следующий пример:

```
<?
    $i = 10;
    $j = 11;
    if ($i > $j)
    $diff = $j – $i;
    echo '$j больше, чем $i; разность между $j и $i составляет ';
    $diff; //НЕВЕРНО!
?>
```

Вопреки возможным ожиданиям, строка 6 выполнится, хотя условие ($i > $j) ложно. Дело в том, что к if(...) относится лишь следующее выражение – строка 5. Строка 6 выполняется в любом случае – действие if(..) на нее уже не распространяется. Для получения нужного эффекта следует воспользоваться блоком операторов, который задается фигурными скобками:

```
<?
    $i = 10;
    $j = 11;
    if ($i > $j)
    {
    $diff = $j – $i;
    echo '$j больше, чем $i; разность между $j и $i составляет ';
    $diff;
    }
?>
```

Фигурные скобки можно использовать даже если внутри только один оператор. Рекомендуется поступать именно так – меньше шансов ошибиться. На производительности это никак не отражается, зато повышает читабельность.

Часто нужно ввести дополнительные условия (если так... а если по-другому... иначе) или даже (если так… а если по-другому… а если еще по-другому... иначе):

```
<?
    $i = 10;
    $j = 11;
    if ($i > $j)
    {
    echo '$i больше, чем $j';
    }
     else if ($i < $j)
    {
    echo '$i меньше, чем $j';
    }
    else
    { // ничего, кроме равенства, не остается :)
    echo '$i равно $j';
    }
?>
```

Для дополнительных «развилок» используется оператор *if ... else if ... else*. Как и в случае с if, блок *else* может отсутствовать. Все операторы echo по правилам заключаются в фигурные скобки.

В данной программе присутствуют // – это комментарий. Это информация для человека, PHP ее игнорирует. Комментарии бывают двух видов: однострочный, как здесь – начинается с // и распространяется до конца строки, и многострочный – считается все, что расположено между парами символов /* и */.

Комментарий вида // – один из немногих случаев, когда инструкция заканчивается переводом строки. Отметим, что PHP в большинстве случаев безразличны переводы строк: все предыдущие примеры вполне можно было бы записать в одну строку.

Бывает необходимость осуществления «развилки» в зависимости от значения одной и той же переменной или выражения. Можно написать:

```
if ($i==1) {
  // код, соответствующий $i==1
} else if ($i==2) {
  // код, соответствующий $i==2
} else if ($i==3) {
  // код, соответствующий $i==3...
}
```

Но существует более удобный для этого случая оператор – *switch*. Выглядит это так:

```
<? $i = 1;
    switch ($i) {
    case 1:
        echo 'один';
        break;
    case 2:
        echo 'два';
        break;
    case 3:
        echo 'три';
        break;
    default:
    echo 'я умею считать только до трех! ;)';
    }
?>
```

Операторы, находящиеся между case-ами, не нужно заключать в фигурные скобки – каждое ответвление заканчивается оператором break. Специальное условие default соответствует «всему остальному» Условие default может отсутствовать.

Циклы

Любой язык программирования содержит операторы организации циклов для повторного выполнения фрагментов кода.

Цикл *while*:

```
<?
    $i = 1;
    while($i < 10) {
        echo $i . "<br>\n";
        $i++;
    }
?>
```

Цикл while (строка 3) работает следующим образом. Сначала проверяется истинность выражения в скобках. Если оно не истинно, тело цикла (все, что расположено между последующими фигурными скобками) не выполняется. Если же оно истинно, после выполнения кода, находящегося в теле цикла, опять проверяется истинность выражения и т. д.

В теле цикла (строки 4, 5) выводится текущее значение переменной $i, после чего значение $i увеличивается на единицу.

Переменную, используемую подобно $i в данном примере, часто называют переменной-счетчиком цикла, или просто счетчиком.

$i++*, операция инкрементирования (увеличения значения на 1) – сокращенная запись для *$i=$i+1; аналогичная сокращенная запись – *$i+=1*. По последнему правилу можно сокращать любые бинарные операции (например, конкатенация: *$s .= 'foo'* – аналог *$s = $s . 'foo'*); однако аналогично инкрементированию можно записать только декрементирование (уменьшение значения на 1):

$i--.

Возможна также запись ++$i (и --$i); различие в расположении знаков операции проявляется только при непосредственном использовании результата этого вычисления: если $i равна 1, в случае $j=$i++ переменная $j получит значение 1, если же $j=++$i, $j будет равняться двум. Из-за этой особенности операция ++$i называется преинкрементом, а $i++ – постинкрементом.

Если бы мы не увеличивали значение $i, выход из цикла никогда бы не произошел («вечный цикл»).

Запишем тот же пример в более краткой форме:

```
<?
    $i = 1;
    while($i < 10) {
    echo $i++ . "<br>\n";
    }
?>
```

И еще один вариант:

```
<?
    $i = 0;
    while(++$i < 10) {
    echo $i . "<br>\n";
    }
?>
```

Заметим, что все эти три программы работают одинаково. В зависимости от начального значения счетчика удобнее та или иная форма записи.

Цикл *do...while* аналогичен циклу while, отличаясь от него тем, что условие находится в конце цикла. Таким образом, тело цикла do..while выполняется хотя бы один раз.

Пример:

```
<?
    $i = 1;
    do {
    echo $i . "<br>\n";
    } while ($i++ < 10);
?>
```

Цикл *for* – достаточно универсальная конструкция. Он может выглядеть как просто, так и очень запутанно.

Рассмотрим классический вариант его использования:

```
<?
    for ($i=1; $i<10; $i++) {
    echo $i . "<br>\n";
    }
?>
```

Этот скрипт выводит в браузер числа от 1 до 9.

Синтаксис цикла for в общем случае:

for*(выражение_1; выражение_2; выражение_3)*,

где выражение_1 выполняется перед выполнением цикла, выражение_2 – условие выполнения цикла (аналогично while), а выражение_3 выполняется после каждой итерации цикла.

Перепишем «общий случай» цикла for в переложении на цикл while:

```
        //  цикл for
for (выражение_1; выражение_2; выражение_3)
{
тело_цикла
}
        //  цикл while
выражение_1;
while (выражение_2)
{
```

```
тело_цикла
выражение_3;
}
```

Обратим внимание на следующий цикл:

```
<?
$i=0;
for ($i++; --$i<10; $i+=2) {
echo $i . "<br>\n";
}
?>
```

Операторы break и continue. Вложенные циклы

Может возникнуть необходимость выхода из цикла при определенном условии, проверяемом в теле цикла. Для этого служит оператор break, с которым мы уже встречались, рассматривая switch.

```
<?
    $i = 0;
    while (++$i < 10) {
    echo $i . "<br>\n";
    if ($i == 5) break;
    }
?>
```

Этот цикл выведет только значения от 1 до 5. При $i==5 сработает условный оператор if в строке 5, и выполнение цикла прекратится.

Оператор continue начинает новую итерацию цикла. В следующем примере с помощью continue «пропускается» вывод числа 5:

```
<?
    for ($i=0; $i<10; $i++) {
    if ($i == 5) continue;
    echo $i . "<br>\n";
    }
?>
```

Операторы break и continue можно использовать совместно со всеми видами циклов.

Циклы могут быть вложенными: внутри одного цикла может располагаться другой цикл и т. д. Операторы break и continue имеют необязательный числовой параметр, указывающий, к какому по порядку вложенности циклу – считая снизу вверх от текущей позиции – относятся (на самом деле, break – это сокращенная запись break 1; аналогично и с continue). Пример выхода из двух циклов сразу:

```
<?
    for ($i=0; $i<10; $i++) {
    for ($j=0; $j<10; $j++) {
    if ($j == 5) break 2;
    echo '$i=' . $i . ', $j=' . $j . "<br>\n";
    }
    }
?>
```

Цикл foreach

Для перебора элементов массива предусмотрен специальный цикл foreach:

```
<?
    $languages = array(
    1 => 'Assembler',
    'C++',
    'Pascal',
    'scripting' => 'bash'
    );
    $languages['php'] = 'PHP';
    $languages[100] = 'Java';
    $languages[] = 'Perl';
?>
<table>
<tr>
<th>Индекс</th>
<th>Значение</th>
</tr>
<?
    foreach ($languages as $key => $value) {
    echo '<tr><td>' . $key . '</td><td>' . $value . '</td></tr>';
    }
?>
</table>
```

Этот цикл работает следующим образом: в порядке появления в коде программы элементов массива $languages, переменным $key и $value присваиваются, соответственно, индекс и значение очередного элемента, и выполняется тело цикла.

Если индексы нас не интересуют, цикл можно записать следующим образом: *foreach ($languages as $value).*

Конструкции list и each

В дополнение к уже рассмотренной конструкции array, существует дополняющая ее конструкция list, являющаяся своего рода антиподом array: если последняя используется для создания массива из набора значений, то list, напротив, заполняет перечисленные переменные значениями из массива.

Допустим, у нас есть массив *$lang = array('php', 'perl', 'basic').* Тогда конструкция *list($a, $b) = $lang* присвоит переменной $a значение 'php', а $b – 'perl'. Соответственно, *list($a, $b, $c) = $lang* дополнительно присвоит $c = 'basic'.

Если бы в массиве $lang был только один элемент, PHP бы выдал замечание об отсутствии второго элемента массива.

А если нас интересуют не только значения, но и индексы? Воспользуемся конструкцией each, которая возвращает пары индекс-значение.

```
<?
    $browsers = array(
    'MSIE' => 'Microsoft Internet Explorer 6.0',
    'Gecko' => 'Mozilla Firefox 0.9',
    'Opera' => 'Opera 7.50'
    );
    list($a, $b) = each($browsers);
    list($c, $d) = each($browsers);
    list($e, $f) = each($browsers);
    echo $a.':'.$b."<br>\n";
    echo $c.':'.$d."<br>\n";
    echo $e.':'.$f."<br>\n";
?>
```

В первую очередь может удивить тот факт, что в строках 8–10 переменным присваиваются разные значения, хотя выражения справа от знака присваивания совершенно одинаковые. Дело в том, что у каждого массива есть скрытый указатель текущего элемента. Изначально он указывает на первый элемент. Конструкция each же продвигает указатель на один элемент вперед.

Эта особенность позволяет перебирать массив с помощью обычных циклов while и for. Конечно, ранее рассмотренный цикл foreach удобнее, и стоит предпочесть его, но конструкция с использованием each довольно распространена, и вы можете ее встретить во множестве скриптов в сети.

```
<?
    $browsers = array(
    'MSIE' => 'Microsoft Internet Explorer 6.0',
    'Gecko' => 'Mozilla Firefox 0.9',
    'Opera' => 'Opera 7.50'
    );

    while (list($key,$value)=each($browsers)) {
    echo $key . ':' . $value . "<br>\n";
    }
?>
```

В завершение цикла указатель текущего элемента указывает на конец массива. Если цикл необходимо выполнить несколько раз, указатель надо принудительно сбросить с помощью оператора reset: *reset($browsers)*. Этот оператор устанавливает указатель текущего элемента в начало массива.

Мы рассмотрели только самые основы работы с массивами. В PHP существует множество разнообразных функций для работы с массивами.

Константы

В отличие от переменных, значение константы устанавливается единожды и не подлежит изменению. Константы не начинаются с символа $ и определяются с помощью оператора define:

```
<?
    define ('MY_NAME', 'Вася');
    echo 'Меня зовут ' . MY_NAME;
?>
```

Константы необязательно называть прописными буквами, но это общепринятое (и удобное) соглашение.

Поскольку имя константы не начинается с какого-либо спецсимвола, внутри двойных кавычек значение константы поместить невозможно (так как нет возможности различить, где имя константы, а где – просто текст).

4.4. Обработка запросов с помощью PHP

HTML-формы. Массивы $_GET и $_POST

Формы являются основным способом обмена данными между веб-сервером и браузером, т. е. обеспечивают взаимодействие с пользователем – собственно, для чего и нужно веб-программирование.

Итак, рассмотрим пример:

```
<html>
<body>
<?
    if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    echo '<h1>Привет, <b>' . $_POST['name'] . '</b></h1>!';
    }
?>
<form method="POST" action="<?=$_SERVER['PHP_SELF']?>">
    Введите Ваше имя: <input type="text" name="name">
    <br>
    <input type="submit" name="okbutton" value="OK">
</form>
</body>
</html>
```

Форма, приведенная в строках 8–12, содержит два элемента: name и okbutton. Атрибут method указывает метод отправки формы POST, а атрибут action, указывающий URL, на который отправляется форма, заполняется значением серверной переменной PHP_SELF – адресом выполняемого в данный момент скрипта.

<?=$_SERVER['PHP_SELF']?> – сокращенная форма записи для echo: <? echo $_SERVER['PHP_SELF']; ?>.

Предположим, в поле name мы ввели значение «Вася», и нажали кнопку «OK». При этом браузер отправляет на сервер POST-запрос.

Тело запроса: *name=Вася&okbutton=OK*. PHP автоматически заполняет массив $_POST:

$_POST['name'] = 'Вася'

$_POST['okbutton'] = 'OK'

В действительности, значение «Вася» отправляется браузером в urlencode-виде; для кодировки windows-1251 это значение выглядит как %C2%E0%F1%FF. Но, поскольку PHP автоматически осуществляет необходимое декодирование, мы можем «забыть» об этой особенности – пока не придется работать с HTTP-запросами вручную.

Так как в теле запроса указываются только имена и значения, но не типы элементов форм, PHP понятия не имеет, соответствует $_POST['name'] строке ввода, кнопке или списку.

Поскольку знать, что написано на кнопке submit, нам необязательно, в строке 11 можно удалить атрибут name, сократив описание кнопки до *<input type="submit" value="OK">*. В этом случае браузер отправит POST-запрос *name=Вася*.

А теперь – то же самое, но для GET-формы:

```
<html>
<body>
<?
    if (isset($_GET['name'])) {
    echo '<h1>Привет, <b>' . $_GET['name'] . '</b></h1>!';
    }
?>
<form action="<?=$_SERVER['PHP_SELF']?>">
    Введите Ваше имя: <input type="text" name="name">
    <br>
    <input type="submit" value="OK">
</form>
</body>
</html>
```

В строке 8 можно было бы с таким же успехом написать *<form method="GET">*: GET – метод по умолчанию. В этот раз браузер отправляет GET-запрос, который равносилен вводу в адресной строке адреса: *http://адрес-сайта/имя-скрипта.php?name=Вася*.

PHP с GET-формами поступает точно так же, как и с POST, с тем отличием, что заполняется массив $_GET.

Кардинальное отличие – в строке 4. Поскольку простой ввод адреса в строке браузера является GET-запросом, проверка *if ($_SERVER['REQUEST_METHOD'] == 'GET')* бессмысленна: все, что мы в этом случае выясним – что кто-то не отправил на наш скрипт POST-форму. Поэтому мы прибегаем к конструкции *isset()*, которая возвращает true, если данная переменная определена (т. е. ей было присвоено значение), и false – если переменная не определена. Если форма была заполнена – PHP автоматически присваивает $_GET['name'] соответствующее значение.

Способ проверки с помощью isset() – универсальный, его можно было бы использовать и для POST-формы. Более того, он предпочтительнее, так как позволяет выяснить, какие именно поля формы заполнены.

Рассмотрим более сложный пример:

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

```
    if (isset($_POST['name'], $_POST['year'])) {
    if ($_POST['name'] == '') {
    echo 'Укажите имя!<br>';
    } else if ($_POST['year'] < 1900 || $_POST['year'] > 2004) {
    echo 'Укажите год рождения! Допустимый диапазон значений: 1900..2004<br>';
    } else {
    echo 'Здравствуйте, ' . $_POST['name'] . '!<br>';
    $age = 2004 – $_POST['year'];
    echo 'Вам ' . $age . ' лет<br>';
    }
    echo '<hr>';
    }
?>
<form method="post" action="<?=$_SERVER['PHP_SELF']?>">
    Введите Ваше имя: <input type="text" name="name">
    <br>
    Введите Ваш год рождения: <input type="text" name="year">
    <input type="submit" value="OK">
</form>
</body>
</html>
```

Изменим последний пример, чтобы пользователю не нужно было повторно заполнять поля. Для этого заполним атрибуты value элементов формы только что введенными значениями.

```
<html>
<body>
<?
    $name = isset($_POST['name']) ? $_POST['name'] : '';
    $year = isset($_POST['year']) ? $_POST['year'] : '';

    if (isset($_POST['name'], $_POST['year'])) {
    if ($_POST['name'] == '') {
    echo 'Укажите имя!<br>';
    } else if ($_POST['year'] < 1900 || $_POST['year'] > 2004) {
    echo 'Укажите год рождения! Допустимый диапазон значений: 1900..2004<br>';
    } else {
    echo 'Здравствуйте, ' . $_POST['name'] . '!<br>';
```

```
    $age = 2004 – $_POST['year'];
    echo 'Вам ' . $age . ' лет<br>';
    }
    echo '<hr>';
    }
?>
<form method="post" action="<?=$_SERVER['PHP_SELF']?>">
        Введите Ваше имя: <input type="text" name="name"
    value="<?=$name?>">
        <br>
        Введите Ваш год рождения: <input type="text"
    name="year" value="<?=$year?>">
        <input type="submit" value="OK">
</form>
</body>
</html>
```

Несколько непонятными могут оказаться строки 4 и 5.

X = A ? B : C – сокращенная запись условия *if (A) X=B else X=C*. Строку 4 можно было бы записать так:

```
if (isset($_POST['name']))
  $name = $_POST['name'];
else
  $name = '';
```

Используемая в строках 21 и 23 конструкция *<?=$foo ?>* – это сокращение для *<? echo $foo ?>*.

Может возникнуть вопрос – почему бы не выбросить строки 4–5 и не написать:

```
    Введите Ваше имя: <input type="text" name="name"
    value="<?=$_POST['name']?>"><br>
    Введите Ваш год рождения: <input type="text" name="year"
    value="<?=$_POST['year']?>">
```

Дело в том, что, если эти POST-переменные не определены и если форму еще не заполняли, PHP выдаст предупреждения об использовании неинициализированных переменных (причем, вполне обоснованно: такое сообщение позволяет быстро находить труднообнаружимые опечатки в именах переменных, а также предупреждает о возможных «дырах» на сайте). Можно, конечно, поместить код с isset... прямо в форму, но получится слишком громоздко.

А теперь попробуем найти недочет в приведенном коде.

Введем в поле "имя" двойную кавычку и какой-нибудь текст, например, *"123*. Отправим форму, и взглянем на исходный код полученной страницы. Четвертая строка примет вид:

Введите Ваше имя: <input type="text" name="name" value=""123">

Для решения этой проблемы необходимо воспользоваться функцией *htmlspecialchars()*, которая заменит служебные символы на их HTML-представление (например, кавычку – на "):

```
<html>
<body>
<?
    $name = isset($_POST['name']) ? htmlspecial-
    chars($_POST['name']) : '';
    $year = isset($_POST['year']) ? htmlspecialchars($_POST['year'])
    : '';
    if (isset($_POST['name'], $_POST['year'])) {
    if ($_POST['name'] == '') {
    echo 'Укажите имя!<br>';
    } else if ($_POST['year'] < 1900 || $_POST['year'] > 2004) {
    echo 'Укажите год рождения! Допустимый диапазон значе-
    ний: 1900..2004<br>';
    }
    else
    {
    echo 'Здравствуйте, ' . $name . '!<br>';
    $age = 2004 – $_POST['year'];
    echo 'Вам ' . $age . ' лет<br>';
    }
    echo '<hr>';
    }
?>
<form method="post" action="<?=$_SERVER['PHP_SELF']?>">
    Введите Ваше имя: <input type="text" name="name"
    value="<?=$name?>">
    <br>
```

```
    Введите Ваш год рождения: <input type="text" name="year"
    value="<?=$year?>">
    <input type="submit" value="OK">
</form>
</body>
</html>
```

В данном случае HTML-код корректен.

Запомните – функцию *htmlspecialchars()* необходимо использовать всегда, когда выводится содержимое переменной, в которой могут присутствовать спецсимволы HTML.

Функция *phpinfo()* – выводит информацию о настройках PHP, значения всевозможных конфигурационных переменных.

phpinfo( ) – удобнейшее средство отладки. phpinfo(), помимо прочего, выводит значения всех $_GET, $_POST и $_SERVER- переменных. Поэтому, если переменная формы «потерялась», самый простой способ обнаружить, в чем дело – воспользоваться функцией phpinfo. Для того чтобы функция выводила только значения переменных, ее следует вызвать следующим образом: *phpinfo(INFO_VARIABLES);*, или – что абсолютно то же самое – phpinfo(32);.

Рассмотрим пример:

```
<html>
<body>
<form method="post" action="<?=$_SERVER['PHP_SELF']?>">
Введите Ваше имя: <input type="text" name="name">
    <input type="submit" value="OK">
</form>
<?
    phpinfo(32);
?>
</body>
</html>
```

Следующая ситуация: необходимо узнать IP-адрес посетителя. Вы помните, что соответствующая переменная хранится в массиве $_SERVER, но забыли, как именно переменная называется. Опять вызываем *phpinfo(32);*, ищем в табличке свой IP-адрес и находим его – в строке _SERVER["REMOTE_ADDR"].

4.5. Функции в PHP

Функция может быть определена с использованием такого синтаксиса:

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

Внутри функции может появляться любой правильный код PHP и другие функции и определения классов.

В PHP 3 функции обязаны быть определены до обращения к ним. Такого требования нет в PHP 4.

PHP не поддерживает перегрузку/overloading функций; также невозможно разопределить или переопределить ранее объявленную функцию.

PHP 3 не поддерживает переменное количество аргументов функции, хотя аргументы по умолчанию поддерживаются.

PHP 4 поддерживает и то, и другое (списки аргументов переменного размера, функции func_num_args(), func_get_arg() и func_get_args()).

Аргументы функции

Информация может передаваться в функцию через список аргументов, который является списком разделенных запятыми переменных и/или констант.

PHP поддерживает разбор аргументов по значению (по умолчанию), разбор по ссылке и значения по умолчанию. Списки аргументов Variable-length поддерживаются только в PHP 4 и позднее. Аналогичный эффект может быть достигнут в PHP 3 путем передачи функции массива аргументов:

```
function takes_array($input)
{
echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

Создание аргументов, передаваемых по ссылке

По умолчанию аргументы передаются в функцию по значению (поэтому, если изменять значение аргумента внутри функции, оно не

изменяется за пределами этой функции). Если дать функции возможность модифицировать свои аргументы, то необходимо передавать их по ссылке.

Для того чтобы передавать аргументы по ссылке, нужно ввести префикс-амперсанд (&) в имени аргумента в определении функции:

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

Значения по умолчанию аргументов

Функция может определить значения по умолчанию в стиле C++ для скадярных аргументов:

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

Результат будет таким:

```
Making a cup of cappuccino.
Making a cup of espresso.
```

Значение по умолчанию обязано быть константным выражением, а не (например) переменной или членом класса.

Обратите внимание, что при использовании аргументов по умолчанию любые значения должны находиться справа от любых значений не по умолчанию.

Рассмотрим следующий фрагмент кода:

```
function makeyogurt ($type = "acidophilus", $flavour)
{
return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt ("raspberry");  // не будет работать так, как
                                // ожидается
```

Результат работы этого примера:

```
Warning!: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Making a bowl of raspberry.
```

Теперь сравним со следующим кодом:

```
 function makeyogurt ($flavour, $type = "acidophilus")
{
return "Making a bowl of $type $flavour.\n";
}
 echo makeyogurt ("raspberry");   // работает как положено
```

Результат работы этого примера:

```
Making a bowl of acidophilus raspberry.
```

Списки аргументов переменного размера

В PHP 4 имеется поддержка списков аргументов переменного размера/variable-length в пользовательских функциях. Это достаточно легко делается функциями func_num_args(), func_get_arg() и func_get_args().

Возвращаемые значения

Значения из функций возвращаются с помощью необязательного оператора return. Может быть возвращен любой тип, в том числе список и объект. Этот оператор немедленно останавливает выполнение функции и передает управление обратно на строчку, с которой функция была вызвана.

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

Вы можете возвратить из функции несколько значений, но исходные результаты можно получить путем возвращения списка.

```
function small_numbers()
{
   return array (0, 1, 2);
}
list ($zero, $one, $two) = small_numbers();
```

Чтобы возвратить из функции ссылку, вы должны использовать операцию ссылки & в объявлении функции и в присвоении возвращаемого значения переменной:

```
function &returns_reference()
{    return $someref;
        }
$newref =& returns_reference();
```

4.6. Объекты и классы в PHP

Класс – это коллекция переменных и функций, работающих с этими переменными. Класс определяется с использованием следующего синтаксиса:

```
<?php
class Cart
{
   var $items;  //Элементы в нашей shopping cart
   // Добавить $num артикулов $artnr в cart
   function add_item ($artnr, $num)
   {
      $this->items[$artnr] += $num;
   }
     // Изъять $num артикулов $artnr из cart
    function remove_item ($artnr, $num)
   {
      if ($this->items[$artnr] > $num) {
         $this->items[$artnr] -= $num;
         return true;
      } else {
         return false;
      }
   }
}
?>
```

Здесь определен класс Cart, состоящий из ассоциативного массива артикулов /articles в карте /cart и двух функций для добавления и удаления элементов этой cart.

В PHP 4 только константные инициализаторы для var-переменных допустимы. Для инициализации переменных с неконстантными значениями необходима функция инициализации, которая

вызывается автоматически, когда конструируется из класса. Такая функция называется *конструктором*.

```
<?php
/* Это не будет работать в PHP 4 */
class Cart
{
   var $todays_date = date("Y-m-d");
   var $name = $firstname;
   var $owner = 'Fred ' . 'Jones';
   var $items = array("VCR", "TV");
}

/* Вот как это должно делаться */
class Cart
{
   var $todays_date;
   var $name;
   var $owner;
   var $items;
   function Cart()
   {
      $this->todays_date = date("Y-m-d");
      $this->name = $GLOBALS['firstname'];
   }
}
?>
```

Классы являются шаблонами реальных переменных. Переменная нужного типа создается операцией new.

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

Здесь создаются объекты $cart и $another_cart, оба от класса Cart. Функция add_item() объекта $cart вызывается для добавления одного элемента артикула номер 10 в $cart. Три элемента артикула номер 0815 добавляются в $another_cart.

И $cart, и $another_cart имеют функции add_item(), remove_item() и элементы переменных. Это разные функции и переменные. Можно представить эти объекты как директории файловой системы. В файловой системе может быть два разных файла README.TXT, если они находятся в разных директориях. Необходимо вводить полный путь к файлу, чтобы извлечь его из директории верхнего уровня. В терминологии PHP директория верхнего уровня – это глобальное пространство имен/global namespace, а разделителем имен служит ->. Таким образом, имена $cart->items и $another_cart->items именуют две различные переменные. Заметьте, что переменная именуется $cart->items, а не $cart->$items, т. е. имя переменной в PHP имеет только один знак dollar.

```
// корректно, один символ $
$cart->items = array("10" => 1);
// неверно, поскольку $cart->$items становится $cart->""
$cart->$items = array("10" => 1);
// корректно, но может и не быть тем, что вы предполагаете:
// $cart->$myvar becomes $cart->items
$myvar = 'items';
$cart->$myvar = array("10" => 1);
```

Внутри определения класса неизвестно, под каким именем объект будет доступен в вашей программе: на момент написания класса Cart не было известно, что объект будет называться $cart или $another_cart. Таким образом, невозможно написать $cart->items в самом классе Cart. Вместо этого, чтобы иметь возможность доступа к переменным и функциям внутри класса, можно использовать псевдопеременную $this, которая может читаться как 'моя собственная' или 'текущий объект'. То есть '$this->items[$artnr] += $num' можно прочитать как 'добавить $num к счетчику $artnr элемента моего собственного массива' или 'добавить $num к счетчику $artnr элемента массива внутри текущего объекта'.

4.7. Работа с массивами данных. Работа со строками

Массив представляет собой набор переменных, объединенных одним именем. Каждое значение массива идентифицируется индексом, который указывается после имени переменной-массива в квадратных скобках. Комбинацию индекса и соответствующего ему значения называют элементом массива.

```
<?
$i = 1024;
$a[1] = 'abc';
$a[2] = 100;
$a['test'] = $i – $a[2];

echo $a[1] . "<br>\n";
echo $a[2] . "<br>\n";
echo $a['test'] . "<br>\n";
?>
```

В приведенном примере, в строке 3, объявляется элемент массива *$a* с индексом *1*; элементу массива присваивается строковое значение 'abc'. Этой же строкой объявляется и массив $a, так как это первое упоминание переменной $a в контексте массива, массив создается автоматически. В строке 4 элементу массива с индексом 2 присваивается числовое значение 100. В строке же 5 значение, равное разности $i и $a[2], присваивается элементу массива $a со строковым индексом 'test'. Как видите, индекс массива может быть как числом, так и строкой.

В предыдущем примере массив создавался автоматически при описании первого элемента массива. Но массив можно задать и явно:

```
<?
$i = 1024;
$a = array( 1=>'abc', 2=>100, 'test'=>$i-100 );
print_r($a);
?>
```

Созданный в последнем примере массив $a полностью аналогичен массиву из предыдущего примера. Каждый элемент массива здесь задается в виде индекс**=>**значение. При создании элемента 'test' пришлось указать значение 100 непосредственно, так как на этот раз мы создаем массив «одним махом», и значения его элементов на этапе создания неизвестны PHP.

В строке 4 для вывода значения массива мы воспользовались функцией print_r(), которая очень удобна для вывода содержимого массивов на экран – прежде всего, в целях отладки.

Строки в выводе функции print_r разделяются обычным переводом строки \n, но не тэгом
. Для удобства чтения, строку print_r(..) можно окружить операторами вывода тэгов <pre>...</pre>:

```
 echo '<pre>';
 print_r($a);
echo '</pre>';
```

Если явно не указывать индексы, то здесь проявляется свойство массивов PHP, характерное для числовых массивов в других языках: очередной элемент будет иметь порядковый числовой индекс. Нумерация начинается с нуля.

Рассмотрим пример:

```
<?
$operating_systems = array( 'Windows', 'Linux', 'FreeBSD', 'OS/2');
$operating_systems[] = 'MS-DOS';
echo "<pre>";
print_r($operating_systems);
echo "</pre>";
?>
```

Результат:

```
Array
(
    [0] => Windows
    [1] => Linux
    [2] => FreeBSD
    [3] => OS/2
    [4] => MS-DOS
)
```

Здесь мы явно не указывали индексы: PHP автоматически присвоил числовые индексы, начиная с нуля. При использовании такой формы записи массив можно перебирать с помощью цикла for. Количество элементов массива возвращает оператор *count* (или его синоним *sizeof*):

```
<?
$operating_systems = array( 'Windows', 'Linux', 'FreeBSD', 'OS/2');
$operating_systems[] = 'MS-DOS';

echo '<table border=1>';
for ($i=0; $i<count($operating_systems); $i++) {
echo '<tr><td>' . $i . '</td><td>' . $operating_systems[$i] . '</td></tr>';
}
echo '</table>';
?>
```

Стили записи можно смешивать. Обратите внимание на то, какие индексы автоматически присваиваются PHP после установки некоторых индексов вручную.

```
<?
$languages = array(
1 => 'Assembler',
'C++',
'Pascal',
'scripting' => 'bash'
);
$languages['php'] = 'PHP';
$languages[100] = 'Java';
$languages[] = 'Perl';

echo "<pre>";
print_r($languages);
echo "</pre>";
?>
```

Результат:

```
Array
(
    [1] => Assembler
    [2] => C++
    [3] => Pascal
    [scripting] => bash
    [php] => PHP
    [100] => Java
    [101] => Perl
)
```

Синтаксис строк

Строковой литерал может специфицироваться тремя способами:

1) одинарными кавычками;
2) двойными кавычками;
3) heredoc-синтаксисом.

Одинарные кавычки

Простейший способ определить строку – это заключить ее в одинарные кавычки «'». Чтобы использовать одинарную кавычку внутри строки, как и во многих других языках, перед ней необходимо

поставить символ обратной косой черты «\», т. е. экранировать ее. Если обратная косая черта должна идти перед одинарной кавычкой либо быть в конце строки, необходимо продублировать ее «\\'».

Если внутри строки, заключенной в одинарные кавычки, обратный слэш «\» встречается перед любым другим символом (отличным от «\» и «'»), то он рассматривается как обычный символ и выводится, как и все остальные. Поэтому обратную косую черту необходимо экранировать, только если она находится в конце строки, перед закрывающей кавычкой.

В PHP существует ряд комбинаций символов, начинающихся с символа обратной косой черты. Их называют управляющими последовательностями. Переменные и управляющие последовательности для специальных символов, встречающиеся в строках, заключенных в одинарные кавычки, не обрабатываются.

Двойные кавычки

Если строка заключена в двойные кавычки «"», PHP распознает большее количество управляющих последовательностей для специальных символов.

При необходимости экранировать любой другой символ, обратная косая черта также будет напечатана.

Важным свойством строк в двойных кавычках является обработка переменных.

Heredoc

Другой способ определения строк – это использование heredoc-синтаксиса. Поддержка heredoc была добавлена в PHP 4. В этом случае строка должна начинаться с символа <<<, после которого идет идентификатор. Заканчивается строка этим же идентификатором. Закрывающий идентификатор должен начинаться в первом столбце строки. Кроме того, идентификатор должен соответствовать тем же правилам наименования, что и все остальные метки в PHP: содержать только буквенно-цифровые символы и знак подчеркивания и начинаться не с цифры или знака подчеркивания.

Heredoc-текст ведет себя так же, как и строка в двойных кавычках, при этом их не имея. Это означает, что нет необходимости экранировать кавычки в heredoc.

Разбор переменных

Когда строка специфицируется двойными кавычками или heredoc, переменные внутри нее разбираются.

Есть два типа синтаксиса: простой и сложный. Простой синтаксис более распространен и удобнее, он предоставляет способ разбора переменной, переменной массива или свойства объекта.

Сложный синтаксис был введен в PHP 4 и может распознаваться по фигурным скобкам {} вокруг выражения.

Простой синтаксис

Если обнаружен знак dollar ($), разборщик захватывает как можно больше лексем для образования правильного имени.

Если вы хотите явным образом специфицировать конец имени, то необходимо имя переменной заключить в фигурные скобки.

```
 $beer = 'Heineken';
echo "$beer's taste is great";    // не будет работать
echo "He drunk some $beers"; // работает, 's' это правильный символ
                                          // для имен переменных
echo "He drunk some ${beer}s"; // работает
```

Аналогично можно разобрать индекс массива или свойство объекта. В индексах массивов закрывающая квадратная скобка (]) обозначает конец индекса. Для свойств объекта применяются те же правила, что и для простых переменных, хотя со свойствами объекта невозможен трюк, как с переменными.

```
$fruits = array( 'strawberry' => 'red' , 'banana' => 'yellow' );
// заметьте, что это работает по-другому вне закавыченной строки
echo "A banana is $fruits[banana].";
echo "This square is $square->width meters broad.";
// не будет работать,  (см. сложный синтаксис)
echo "This square is $square->width00 centimeters broad.";
```

Для чего-либо более сложного необходимо использовать сложный синтаксис.

Сложный (фигурный) синтаксис

Он называется сложным не потому, что сложен, а потому, что можно включать таким способом сложные выражения.

Фактически, можно включать любое значение, которое находится в пространстве имен строки с этим синтаксисом. Можно записы-

вать выражение тем же способом, что и вне строки, а затем окружать его символами { и }. Поскольку невозможно заменить '{' мнемоникой, этот синтаксис будет распознаваться, только когда $ будет непосредственно после {. Используйте "{\$" или "\{$" для получения литерала "{$". Для пояснения рассмотрим пример:

```
great = 'fantastic';
echo "This is { $great}"; // не будет работать,
                          //выведет: This is { fantastic}
echo "This is {$great}";  // работает, выведет: This is fantastic
echo "Это квадрат шириной {$square->width}00 сантиметров.";
echo "Это работает: {$arr[4][3]}";
                          // Неверно по той причине,
                          // что $foo[bar] неверно вне строки.
echo "Это неправильно: {$arr[foo][3]}";
echo "Вы должны сделать это так: {$arr['foo'][3]}";
echo "Вы можете даже записать {$obj->values[3]->name}";
echo "Это значение переменной $name: {${$name}}";
```

Доступ к символу в строке

Можно получить доступ к символам в строке путем специфицирования смещения с базой 0 в фигурных скобках для нужного символа.

Для обеспечения обратной совместимости можно по-прежнему использовать скобки массива. Однако этот синтаксис не рекомендуется, начиная с PHP 4.

Рассмотрим некоторые примеры строк:

```
<?php
/* Присвоение строкового значения. */
$str = "This is a string";

/* Присоединение к ней. */
$str = $str . " with some more text";

/* Другой способ присоединения с мнемоникой для newline. */
$str .= " and a newline at the end.\n";

/* Эта строка будет выглядеть '<p>Number: 9</p>' */
$num = 9;
$str = "<p>Number: $num</p>";
```

```
/* А эта – '<p>Number: $num</p>' */
$num = 9;
$str = '<p>Number: $num</p>';

/* Получить первый символ строки */
$str = 'This is a test.';
$first = $str{0};

/* Получить последний символ строки */
$str = 'This is still a test.';
$last = $str{strlen($str)-1};
?>
```

4.8. Работа с файловой системой

Рассмотрим один из важнейших аспектов PHP – средства файлового ввода/вывода. Входные и выходные потоки данных интенсивно используются при разработке web-приложений. Не ограничиваясь простым чтением/записью файлов, PHP предоставляет в распоряжение программиста средства просмотра и модификации серверной информации, а также запуска внешних программ.

Проверка существования и размера файла

Прежде чем пытаться работать с файлом, желательно убедиться в том, что он существует. Для решения этой задачи обычно используются следующие функции:

file_exists( ), is_file( ), file_exists( ).

Функция f ile_exists ( ) проверяет, существует ли заданный файл. Если файл существует, функция возвращает TRUE, в противном случае возвращается FALSE. Синтаксис функции:

bool file_exists(string файл)

Пример проверки существования файла:

```
if (! file_exists ($filename)) :
print "File $filename does not exist!";
endif:
```

is_file( )

Функция *is_file( )* проверяет существование заданного файла и возможность выполнения с ним операций чтения/записи. В сущности, is_file( ) представляет собой более надежную версию file_exists( ), которая проверяет не только факт существования файла, но и то, поддерживает ли он чтение и запись данных. Синтаксис функции:

bool is_file(string файл)

Следующий пример показывает, как убедиться в существовании файла и возможности выполнения операций с ним:

```
$file = "somefile.txt";
if (is_file($file)) :
print "The file $file is valid and exists!";
else :
print "The file $file does not exist or it is not a valid file!";
endif:
```

Функция *filesize( )* возвращает размер (в байтах) файла с заданным именем или FALSE в случае ошибки. Синтаксис функции:

int filesize(string имя_файла)

Предположим, необходимо определить размер файла pastry.txt. Для получения нужной информации можно воспользоваться функцией *filesize( )*.

```
$fs = filesize("pastry.txt"); print "Pastry.txt is $fs bytes.";
```

Выводится следующий результат:

Pastry.txt is 179 bytes.

Открытие и закрытие файлов

Прежде чем выполнять операции ввода/вывода с файлом, необходимо открыть его функцией fopen( ).

Функция *fopen( )* открывает файл (если он существует) и возвращает целое число – так называемый файловый манипулятор (file handle). Синтаксис функции fopen( ):

int fopen (string файл, string режим [, int включение_пути])

Открываемый файл может находиться в локальной файловой системе, существовать в виде стандартного потока ввода/вывода или представлять файл в удаленной системе, принимаемой средствами HTTP или FTP.

Параметр файл может задаваться в нескольких формах, перечисленных ниже:

– если параметр содержит имя локального файла, функция fopen( ) открывает этот файл и возвращает манипулятор;

– если параметр задан в виде php://stdin, php://stdout или php://stderr, открывается соответствующий стандартный поток ввода/вывода;

– если параметр начинается с префикса http://, функция открывает подключение HTTP к серверу и возвращает манипулятор для указанного файла;

– если параметр начинается с префикса ftp://, функция открывает подключение FTP к серверу и возвращает манипулятор для указанного файла.

В этом случае следует обратить особое внимание на два обстоятельства: если сервер не поддерживает пассивный режим FTP, вызов fopen() завершается неудачей. Более того, FTP-файлы открываются либо для чтения, либо для записи.

При работе в пассивном режиме сервер ожидает подключения со стороны клиентов. При работе в активном режиме сервер сам устанавливает соединение с клиентом. По умолчанию обычно используется активный режим.

Если необязательный третий параметр включение_пути равен 1, то путь к файлу определяется по отношению к каталогу включаемых файлов, указанному в файле php.ini

Приведем пример открытия файла функцией fopen( ). Вызов функции die( ), используемый в сочетании с fopen( ), обеспечивает вывод сообщения об ошибке в том случае, если открыть файл не удастся:

```
$file = "userdata.txt"; // Некоторый файл
$fh = fopen($file, "a+") or die("File ($file) does not exist!");
```

Следующий фрагмент открывает подключение к сайту PHP (http://www.php.net):

```
$site = "http://www.php.net": // Сервер, доступный через HTTP
$sh = fopen($site., "r");          //Связать манипулятор с индексной
                                   //страницей Php.net
```

После завершения работы файл всегда следует закрывать функцией *fclose( )*. При успешном закрытии возвращается TRUE, при неудаче – FALSE. Синтаксис функции fclose( ):

```
int fclose(int манипулятор)
```

Функция fclose( ) успешно закрывает только те файлы, которые были ранее открыты функциями fopen( ) или fsockopen( ).

Пример закрытия файла:

```
$file = "userdata.txt";
if (file_exists($file)) :
$fh = fopen($file, "r");
// Выполнить операции с файлом
fclose($fh);
else :
print "File Sfile does not exist!";
endif;
```

С открытыми файлами выполняются две основные операции – чтение и запись.

Функция *is_writeable( )* позволяет убедиться в том, что файл существует и для него разрешена операция записи. Возможность записи проверяется как для файла, так и для каталога. Синтаксис функции:

bool is_writeable (string файл)

Функция *fwrite( )* записывает содержимое строковой переменной в файл, заданный файловым манипулятором. Синтаксис функции:

int fwrite(int манипулятор, string переменная [, int длина])

Если при вызове функции передается необязательный параметр – длина, запись останавливается либо после записи указанного количества символов, либо при достижении конца строки. Проверка возможности записи в файл продемонстрирована в следующем примере:

```
<?
// Информация о трафике на пользовательском сайте
$data = "08:13:00|12:37:12|208.247.106.187|Win98";
$filename = "somefile.txt";
// Если файл существует и в него возможна запись
if ( is_writeable($filename) ) :
// Открыть файл и установить указатель текущей позиции
//в конец файла
$fh = fopen($filename, "a+");
// Записать содержимое $data в файл
$ success – fwrite($fh, $data);
// Закрыть файл
fclose($fh); else :
print "Could not open Sfilename for writing";
endif;
?>
```

Функция *fputs( )* является псевдонимом fwrite( ) и может использоваться всюду, где используется fwrite( ).

Функция fputs( ) является псевдонимом fwrite( ) и имеет точно такой же синтаксис:

int fputs(int манипулятор, string переменная [, int длина])

Обычно предпочитают использовать fputs( ). Следует помнить, что это всего лишь вопрос стиля, никак не связанный с какими-либо различиями между двумя функциями.

Несомненно, чтение является самой главной операцией, выполняемой с файлами. Ниже описаны некоторые функции, повышающие

эффективность чтения из файла. Синтаксис этих функций практически точно копирует синтаксис аналогичных функций записи.

Функция *is_readable( )* позволяет убедиться в том, что файл существует и для него разрешена операция чтения. Возможность чтения проверяется как для файла, так и для каталога. Синтаксис:

bool is_readable (string filename)

Скорее всего, PHP будет работать под идентификатором пользователя, используемым web-сервером (как правило, «nobody»), поэтому для того, чтобы функция is_readable( ) возвращала TRUE, чтение из файла должно быть разрешено всем желающим. Следующий пример показывает, как убедиться в том, что файл существует и доступен для чтения:

```
if ( is_readable($filename) ) :       // Открыть файл и установить
                                      //указатель  текущей позиции в конец файла
$fh = fopen($filename, "r");
else :
print "$filename is not readable!";
endif;
```

Функция *fread( )* читает из файла, заданного файловым манипулятором, заданное количество байт. Синтаксис функции:

int fread(int манипулятор, int длина)

Манипулятор должен ссылаться на открытый файл, доступный для чтения (функция is_readable( )). Чтение прекращается после прочтения заданного количества байт или при достижении конца файла.

Рассмотрим текстовый файл pastry.txt, где чтение и вывод этого файла в браузере осуществляется следующим фрагментом:

```
$fh = fopen('pastry.txt', "r") or die("Can't open file!");
$file = fread($fh, filesize($fh));
print $file;
fclose($fh);
```

Используя функцию fllesize( ) для определения размера pastry.txt в байтах, можно гарантировать, что функция fread( ) прочитает все содержимое файла.

Текстовый файл pastry.txt

Recipe: Pastry Dough
1 1/4 cups all-purpose flour
3/4 stick (6 tablespoons) unsalted butter, chopped
2 tablespoons vegetable shortening 1/4 teaspoon salt
3 tablespoons water

Функция *fgetc( )* возвращает строку, содержащую один символ из файла в текущей позиции указателя, или FALSE при достижении конца файла. Синтаксис функции:

string fgetc (int манипулятор)

Манипулятор должен ссылаться на открытый файл, доступный для чтения (функции is_readable( )).

В следующем примере продемонстрированы посимвольное чтение и вывод файла с использованием функции fgetc( ):

```
$fh = fopen("pastry.txt", "r"); while (! feof($fh)) :
$char = fgetc($fh):
print $char; endwhile;
fclose($fh);
```

fgets( )

Функция *fgets( )* возвращает строку, прочитанную от текущей позиции указателя в файле, определяемом файловым манипулятором. Файловый указатель должен ссылаться на открытый файл, доступный для чтения (функция is_readable( )). Синтаксис функции:

string fgets (int манипулятор, int длина)

Чтение прекращается при выполнении одного из следующих условий:

– из файла прочитана длина – 1 байт;

– из файла прочитан символ новой строки (включается в возвращаемую строку);

– из файла прочитан признак конца файла (EOF).

Если необходимо организовать построчное чтение файла, передайте во втором параметре значение, заведомо превышающее количество байт в строке.

Пример построчного чтения и вывода файла:

```
$fh = fopen("pastry.txt", "r");
while (! feof($fh));
$line = fgets($fh, 4096);
print $line. "<br>";
endwhile;
fclose($fh):
```

fgetss( )

Функция *fgetss( )* полностью аналогична fgets( ) за одним исключением – она пытается удалять из прочитанного текста все теги HTML и PHP:

string fgetss (Int манипулятор, int длина [, string разрешенные_теги])

4.9. Авторизация доступа с помощью сессий

Рассмотрим задачу авторизации доступа пользователей к ресурсам системы, а также обсудим безопасность построенного решения.

Что такое *авторизация доступа*? Попробуем объяснить на примере. Вы хотите взять в библиотеке книгу. Но эта услуга доступна только тем, у кого есть читательский билет. Можно сказать, что с помощью этого билета производится «авторизация доступа» к библиотечным ресурсам. Библиотекарь после предъявления ему читательского билета знает, кто берет книгу, и в случае необходимости (например, книгу долго не возвращают) может принять меры (позвонить должнику домой). Библиотекарь имеет гораздо больше прав, чем обычный посетитель: он может давать или не давать книги определенному посетителю, может выставлять напоказ новинки и убирать в архив редко читаемые книги и т. п.

В информационных технологиях все примерно так же. В сети существует огромное количество ресурсов, т. е. множество «библиотек». У каждой из них свой «библиотекарь», т. е. человек или группа людей, отвечающих за содержание ресурса и предоставление пользователям информации. Их называют администраторами. Функции администратора, как правило, включают добавление новой информации, удаление и редактирование существующей, настройка способов отображения информации пользователю. А в функции пользователя (простого посетителя ресурса) входит только поиск и просмотр информации.

Как же отличить пользователя от администратора? В реальной библиотеке это как-то очевидно, но если роли библиотекаря и посетителя библиотеки перенести в виртуальную реальность, то эта очевидность исчезает. Библиотекарь, как и посетитель, имеет доступ к библиотечным ресурсам через Internet. А согласно протоколу HTTP все клиенты абсолютно равноправны. Как же понять, кто зашел на сайт? Обычный пользователь (посетитель) или администратор (библиотекарь)? Если это простой пользователь, то как сохранить это знание, чтобы не допустить посетителя в закрытые архивы сайта? То есть возникает вопрос, как идентифицировать клиента, который послал запрос, и сохранять сведения о нем, пока он находится на сайте?

Самый простой вариант – это регистрация человека в системе и выдача ему аналога читательского билета, а именно логина и пароля для входа в административную часть системы. Эта информация хранится на компьютере-сервере, и при входе в систему проверяется со-

ответствие введенных пользователем логина и пароля тем, что хранятся в системе. Правда, здесь по сравнению с реальной библиотекой ситуация изменяется: читательский билет требуется библиотекарю для входа в закрытую часть системы, а читатель может заходить на сайт свободно. В принципе можно регистрировать и простых посетителей. Тогда всех зарегистрированных пользователей нужно разделить на группы: библиотекари (администраторы) и читатели (простые пользователи), наделив их соответствующими правами. Мы не будем вдаваться в эти тонкости и воспользуемся самым простым вариантом, когда ввод логина и пароля требуется для доступа к некоторым страницам сайта.

Сам скрипт авторизации должен предоставлять форму для ввода логина и пароля, проверять их правильность и перенаправлять на секретную страничку, если проверка прошла успешно, и выдавать сообщение об ошибке в противном случае.

```
<?
if (!isset($_GET['go'])){
   // проверяем, отправлены ли данные формой
   echo "<form>
   // форма для авторизации (ввода логина и пароля)
   Login: <input type=text name=login>
   Password: <input type=password
                    name=passwd>
   <input type=submit name=go value=Go>
   </form>";
}else {
// если форма заполнена, то сравниваем логин
// и пароль с правильными логином и  паролем
   if ($_GET['login']=="pit" &&
      $_GET['passwd']=="123") {
   Header("Location: secret_info.html");
   //и перенаправляем на секретную страницу
   }else echo "Неверный ввод, попробуйте еще раз<br>";
}
?>
```

Вроде бы все достаточно просто. Но допустим, у нас не одна секретная страничка, а несколько. Причем они связаны между собой перекрестными ссылками. Тогда возникает необходимость постоянно помнить пароль и логин посетителя сайта (если таковой имеет). Что-

бы решить эту проблему, можно в каждую страницу встроить скрипт, который будет передавать логин и пароль от страницы к странице в качестве скрытых параметров формы. Но такой способ не совсем безопасен: эти параметры можно перехватить и подделать. В PHP существует более удобный и безопасный метод решения проблемы хранения данных о посетителе в течение сеанса его работы с сайтом – это механизм сессий.

Сессии – это механизм, который позволяет создавать и использовать переменные, сохраняющие свое значение в течение всего времени работы пользователя с сайтом.

Эти переменные для каждого пользователя имеют различные значения и могут использоваться на любой странице сайта до выхода пользователя из системы. При этом каждый раз, заходя на сайт, пользователь получает новые значения переменных, позволяющие идентифицировать его в течение этого сеанса или сессии работы с сайтом. Отсюда и название механизма – сессии.

Задача идентификации пользователя решается путем присвоения каждому пользователю уникального номера, так называемого идентификатора сессии (SID, SessionIDentifier). Он генерируется PHP в тот момент, когда пользователь заходит на сайт, и уничтожается, когда пользователь уходит с сайта, и представляет собой строку из 32 символов (например, ac4f4a45bdc893434c95dcaffb1c1811). Этот идентификатор передается на сервер вместе с каждым запросом клиента и возвращается обратно вместе с ответом сервера.

Первое, что нужно сделать для работы с сессиями (если они уже настроены администратором сервера), это запустить механизм сессий. Если в настройках сервера переменная session.auto_start установлена в значение "0" (если session.auto_start=1, то сессии запускаются автоматически), то любой скрипт, в котором нужно использовать данные сессии, должен начинаться с команды:

session_start();

Получив такую команду, сервер создает новую сессию или восстанавливает текущую, основываясь на идентификаторе сессии, переданном по запросу. Как это делается? Интерпретатор PHP ищет переменную, в которой хранится идентификатор сессии (по умолчанию – это PHPSESSID) сначала в cookies, потом в переменных, переданных с помощью POST- и GET-запросов. Если идентификатор найден, то пользователь считается идентифицированным, производится замена всех URL и выставление cookies. В противном случае пользователь

считается новым, для него генерируется новый уникальный идентификатор, затем производится замена URL и выставление cookies.

Команду session_start() нужно вызывать во всех скриптах, в которых предстоит использовать переменные сессии, причем до вывода каких-либо данных в браузер. Это связано с тем, что cookies выставляются только до вывода информации на экран.

Получить идентификатор текущей сессии можно с помощью функции session_id().

Для наглядности сессии можно задать имя с помощью функции session_name([имя_сессии]). Делать это нужно еще до инициализации сессии. Получить имя текущей сессии можно с помощью этой же функции, вызванной без параметров: session_name();

Переименуем наш файл index.html, чтобы обрабатывались php-скрипты, например в Index.php, создадим сессию и посмотрим, какой она получит идентификатор и имя:

```
<?
session_start();
   // создаем новую сессию или
   // восстанавливаем текущую
echo session_id();
   // выводим идентификатор сессии
?>

<html>
<head><title>My home page</title></head>
... // домашняя страничка
</html>
<?
echo session_name();
   // выводим имя текущей сессии.
   // В данном случае это PHPSESSID
?>
```

Если проделать то же самое с файлом авторизации, то значения выводимых переменных (id сессии и ее имя) будут такими же, если перейти на него с index.php и не закрывать перед этим окно браузера (тогда идентификатор сессии изменится).

Однако от самих идентификатора и имени сессии нам пользы для решения наших задач немного. Мы же хотим передавать и сохранять в течение сессии наши собственные переменные (например, ло-

гин и пароль). Для того чтобы этого добиться, нужно просто зарегистрировать свои переменные:

session_register(имя_переменной1, имя_переменной2, ...);

Заметим, что регистрируются не значения, а имена переменных. Зарегистрировать переменную достаточно один раз на любой странице, где используются сессии. Имена переменных передаются функции session_register() без знака $. Все зарегистрированные таким образом переменные становятся глобальными (т. е. доступными с любой страницы) в течение данной сессии работы с сайтом.

Зарегистрировать переменную также можно, просто записав ее значение в ассоциативный массив $_SESSION, т. е. написав

$_SESSION['имя_переменной'] = 'значение_переменой';

В этом массиве хранятся все зарегистрированные (т. е. глобальные) переменные сессии.

Доступ к таким переменным осуществляется с помощью массива $_SESSION['имя_переменной'] (или $HTTP_SESSION_VARS ['имя_переменной'] для версии PHP 4.0.6 и более ранних). Если же в настройках php включена опция register_globals, то к сессионным переменным можно обращаться еще и как к обычным переменным, например так: $имя_переменной.

Если register_globals=off (отключены), то пользоваться session_register() для регистрации переменных, переданных методами POST или GET, нельзя, т. е. это просто не работает. И вообще, не рекомендуется одновременно использовать оба метода регистрации переменных, $_SESSION и session_register(). (Начиная с версии PHP 5.3.0, не рекомендуется для регистрации переменных сессии использовать функцию session_register(); более того, начиная с версии PHP 6.0.0, эта функция станет недоступна. Вместо этого для регистрации переменных сессии рекомендуется пользоваться массивом $_SESSION.)

Зарегистрируем логин и пароль, вводимые пользователем на странице авторизации.

```
<?
session_start();
   // создаем новую сессию или
   // восстанавливаем текущую
if (!isset($_GET['go'])){
  echo "<form>
   Login: <input type=text name=login>
```

```
    Password: <input type=password
                     name=passwd>
    <input type=submit name=go value=Go>
  </form>";
}else {
   $_SESSION['login']=$_GET['login'];
   // регистрируем переменную login
   $_SESSION['passwd']=$_GET['passwd'];
   // регистрируем переменную passwd
// теперь логин и пароль – глобальные
// переменные для этой сессии
    if ($_GET['login']=="pit" &&
       $_GET['passwd']=="123") {
       Header("Location: secret_info.php");
       // перенаправляем на страницу
       // secret_info.php
    }else echo "Неверный ввод,
              попробуйте еще раз<br>";
}
print_r($_SESSION);
    // выводим все переменные сессии
?>
```

Теперь, попав на любую страницу сайта, мы сможем работать с введенными пользователем логином и паролем, которые будут храниться в массиве $_SESSION. Таким образом, если изменить код секретной странички так:

```
<?php
session_start();
    // создаем новую сессию или
    // восстанавливаем текущую
print_r($_SESSION);
    // выводим все переменные сессии
?>
<html>
<head><title>Secret info</title></head>
<body>
<p>Здесь я хочу делиться секретами
с другом Петей.
```

</body>
</html>
Получим в браузере на секретной странице следующее:
Array ([login] => pit [passwd] => 123)
Здесь я хочу делиться секретами с другом Петей.

4.10. Регулярные выражения

Регулярное выражение (regular expression) – это технология, которая позволяет задать шаблон и осуществить поиск данных, соответствующих этому шаблону, в заданном тексте, представленном в виде строки.

Кроме того, с помощью регулярных выражений можно изменить и удалить данные, разбить строку по шаблону на подстроки и др.

Одно из распространенных применений – это проверка строки на соответствие каким-либо правилам. Например, следующее регулярное выражение предназначено для проверки того, что строка содержит корректный e-mail–адрес:

/^\w+([\.\w]+)*\w@\w((\.\w)*\w+)*\.\w{2,3}$/

В определении регулярного выражения несколько раз повторяется термин «шаблон». Что это такое?

Давайте подумаем, что представляет собой корректный e-mail-адрес. Это набор букв, цифр и символов подчеркивания, после которых идет специальный символ «собака» @, затем еще один такой же набор, содержащий имя сервера, точку (.) и две или три буквы, указывающие на зону домена, к которой принадлежит почтовый ящик (ru, com, org и т. д.). Приведенное выше регулярное выражение формализует данное описание на языке, понятном компьютеру и описывает не какой-то конкретный электронный адрес, а все возможные корректные электронные адреса. Таким образом, производится формальное задание множества правильных e-mail'ов с помощью шаблона регулярного выражения. Другие примеры шаблонов – это шаблоны MS Word и html-формы.

Механизм регулярных выражений задает правила построения шаблонов и осуществляет поиск данных по этому шаблону в указанной строке.

Регулярные выражения в PHP пришли из UNIX и Perl. В PHP существуют такие удобные и мощные средства работы со строками, как explode (разбиение строки на подстроки), strstr (нахождение подстроки), str_replace (замена всех вхождений подстроки).

Основное преимущество регулярных выражений заключается в том, что они позволяют организовать более гибкий поиск, т. е. найти то, о чем нет точного знания, но есть примерное представление. Например, нужно найти все семизначные номера телефонов, встречающиеся в тексте. Мы не ищем какой-то заранее известный нам номер телефона, мы знаем только, что искомый номер состоит из семи цифр. Для этого можно воспользоваться следующим регулярным выражением:

/\d{3}-\d{2}-\d{2}/m

В PHP существует два различных механизма для обработки регулярных выражений: POSIX-совместимые и Perl-совместимые (сокращенно PCRE). Их синтаксис во многом похож, однако Perl-совместимые регулярные выражения более мощные и к тому же работают намного быстрее. Начиная с версии PHP 4.2.0, PCRE входят в набор базовых модулей и подключены по умолчанию. POSIX-совместимые регулярные выражения включены по умолчанию только в версию PHP для Windows.

Основные функции для работы с Perl-совместимыми регулярными выражениями: preg_match(pattern, string, [result, flags]) и preg_match_all(pattern, string, result, [flags]), где:

– pattern – шаблон регулярного выражения;

– string – строка, в которой производится поиск;

– result – содержит массив результатов (нулевой элемент массива содержит соответствие всему шаблону, первый – первому «захваченному» подшаблону и т. д.);

– flags – необязательный параметр, определяющий то, как упорядочены результаты поиска.

Эти функции осуществляют поиск по шаблону и возвращают информацию о том, сколько раз произошло совпадение. Для preg_match() это 0 (нет совпадений) или 1, поскольку поиск прекращается, как только найдено первое совпадение. Функция preg_match_all() производит поиск до конца строки и поэтому находит все совпадения. Все точные совпадения содержатся в первом элементе массива result у каждой из этих функций (для preg_match_all() этот элемент – тоже массив).

Аналогом preg_match является булева функция POSIX-расширения ereg(string pattern, string string [, array regs])

Функция ereg() возвращает TRUE, если совпадение найдено, и FALSE – в противном случае.

Регулярное выражение представляет собой строку. Эта строка состоит из собственно регулярного выражения (шаблона), выделенного с помощью специального символа разделителя (это могут быть символы " / " , " | ", " { ", " !" и т. п.) и модификатора, влияющего на способ обработки РВ.

Мощь регулярных выражений порождена в основе своей их способностью включать в шаблон альтернативы и повторения. Они кодируются в шаблоне с помощью метасимволов. Метасимвол отличается от любого другого символа тем, что имеет специальное значение.

Одним из основных метасимволов является обратный слэш " \ ". Он меняет тип символа, следующего за ним, на противоположный, т. е. если это был обычный символ, то он МОЖЕТ превратиться в метасимвол, если это был метасимвол, то он теряет свое специальное значение и становится обычным символом (это нужно для того, чтобы вставлять в текст специальные символы как обычные). Например, символ d в обычном режиме не имеет никаких специальных значений, но \d есть метасимвол, означающий «любая цифра». Символ " ." в обычном режиме означает «любой единичный символ», а " \." означает просто точку.

Пример использования приведенных выше метасимволов:

/\d\d\d plus \d is \w\w\w/

Это регулярное выражение означает: трехзначное число, за которым следует подстрока plus, любая цифра, затем is и слово из трех словарных символов. В частности, данному регулярному выражению удовлетворяют строки:

" 123 plus 3 is sum ", " 213 plus 4 is 217 ".

Рассмотрим пример. Пусть имеется некий текст. Нам нужно найти всех перечисленных в нем людей со званиями.

```
<?
$str = "Доцент Смирнов совершил ".
     "открытие. Его учителем была ".
     "профессор Иванова. ".
      "Этим открытием Смирнов ".
      "завоевал себе степень ".
        "доктора. Раньше он был ".
        "только кандидат.";
$pattern = "/(профессор|доцент)".
          "\s[А-Я][А-Яа-я]+(\s|\.)/i";
// осуществляем поиск
```

```
$n = preg_match_all ($pattern, $str,
                          $res);
// выводим результаты
for ($i=0;$i<$n;$i++)
  echo htmlspecialchars($res[0][$i]).
                    "<br>";
?>
```

Метасимвол прямая черта "|" позволяет задавать альтернативные варианты.

В примере мы хотели найти всех профессоров или доцентов. Для этого было создано подвыражение " (профессор|доцент) ". После звания через пробел фамилия человека, которому оно принадлежит, – для этого существует комбинация " \s[А-Я][А-Яа-я]+ ". После фамилии идет либо опять пробел, либо точка, если это конец предложения. Получаем опять два альтернативных варианта: " (\s|\.) " (здесь точка экранируется обратным слэшем, чтобы она понималась как обычная точка, без специального значения).

Подвыражения (подшаблоны)

В регулярных выражениях подшаблоны выделяют, заключая в круглые скобки. Для их обозначения кроме термина «подшаблон» также используют термин «подвыражение». Подшаблоны могут быть вложенными. Выделение части регулярного выражения в виде регулярного подвыражения делает следующее:

– локализует множество альтернатив;

– устанавливает подшаблон как «захватывающий» подшаблон. Это значит, что, когда какая-то подстрока в тексте совпала с шаблоном, все подстроки, которые совпали с подшаблонами этого РВ, тоже возвращаются в качестве результата. Скобки, обозначающие начало подшаблона, пересчитываются слева направо (начиная с 1) для того, чтобы узнать, сколько подшаблонов нужно захватить.

Повторения (квантификаторы)

Повторения описываются с помощью так называемых *квантификаторов* (метасимволов, задающих количественные отношения). Существует два типа квантификаторов: общие (задаются с помощью фигурных скобок) и сокращенные (это исторически сложившиеся сокращения наиболее распространенных квантификаторов).

Квантификаторы могут следовать за любым из перечисленных элементов:

– одиночный символ (возможно, в комбинации с обратным слэшем);

– метасимвол «точка»;

– символьный класс;

– обратная ссылка (о них расскажем позднее);

– подшаблон.

Общие квантификаторы задают минимальное и максимальное числа дозволенных повторений элемента; эти два числа, разделенные запятой, заключаются в фигурные скобки. Числа не должны превышать 65 536 и первое число должно быть меньше или равно второму. Например,

x{1,3}

говорит о том, что символ **x** должен повторяться минимум один, а максимум три раза. Соответственно, этому шаблону удовлетворяют строки: x, xx, xxx.

Если второй параметр отсутствует, но запятая есть, то повторений может быть сколько угодно.

Таким образом, *[aeuoi]{2,}* значит, что любой из символов " a ", " e ", " u ", " o ", " i " в строке может повторяться два и более раз, а регулярное выражение \d{3} задает ровно три цифры.

Сокращенные квантификаторы задают наиболее используемые количественные отношения (повторения). Они придуманы для удобства, чтобы не перегружать и без того сложные выражения лишним синтаксисом.

Исходя из исторических традиций, три наиболее часто встречающихся квантификатора имеют следующие обозначения:

* эквивалентно {0,} – то есть это нуль и более повторений;

+ эквивалентно {1,} – то есть это одно и более повторений;

? эквивалентно {0,1} – то есть это нуль или одно повторение.

По умолчанию все квантификаторы «жадные», они стараются захватить как можно больше повторений элемента. То есть если указать, что символ должен повторяться один и более раз (например, с помощью *), совпадение произойдет со строкой, содержащей наибольшее число повторений указанного символа. Это может создать проблемы, например, при попытке выделить комментарии в программе на языке Си или PHP. Комментарии в Си и PHP записываются между символами /* и */, внутри которых тоже могут встречаться сим-

волы * и /. И попытка выявить Си-комментарии с помощью шаблона /* .* */ в строке

```
/* первый комментарий */
    не комментарий
/* второй комментарий */
```

не увенчается успехом из-за «жадности» элемента ".*" (будет найдена также строка «не комментарий»).

Для решения этой проблемы нужно написать знак вопроса после квантификатора. Тогда он перестанет быть «жадным» и попытается захватить как можно меньшее число повторений элемента, к которому он применен (квантификатор применяется к элементу, что стоит перед ним). Так что шаблон /* .*? */ успешно выделяет Си-комментарии.

В PHP существует опция PCRE_UNGREEDY, которая делает все квантификаторы «не жадными» по умолчанию и «жадными», если после них идет знак вопроса.

Модификаторы PCRE

Еще один немаловажный элемент регулярного выражения – это список применяемых к нему модификаторов. Модификаторы – это выдаваемая интерпретатору регулярных выражений инструкция по обработке данного выражения. Например, считать, что все символы регулярного выражения соответствуют как большим, так и маленьким буквам в строке, где производится поиск.

Рассмотрим основные модификаторы:

i(PCRE_CASELESS) – если указан этот модификатор, то буквы в шаблоне совпадают с буквами и верхнего, и нижнего регистра в строке;

m(PCRE_MULTILINE) – по умолчанию строка, подающаяся на вход интерпретатору регулярного выражения, рассматривается как состоящая из одной линии. Этот модификатор включает поддержку многострокового режима;

s(PCRE_DOTALL) – если установлен этот модификатор, то метасимвол точка " ." совпадает с любым символом, включая символ перевода строки;

x(PCRE_EXTENDED) – заставляет интерпретатор игнорировать пробелы между символами в шаблоне, за исключением пробелов, экранированных обратным слэшем или находящихся внутри символь-

ного класса, а также между неэкранированным символом # вне символьного класса и символом новой строки;

U(PCRE_UNGREEDY) – этот модификатор инвертирует «жадность» квантификаторов, т. е. они становятся «нежадными» по умолчанию и «жадными», если предшествуют символу "?"

Обратная ссылка

Вне определения символьного класса (это тот, что задается квадратными скобками) комбинация обратный слэш и цифра больше нуля (например, \1) называется обратной ссылкой и представляет собой ссылку на захваченное ранее регулярное подвыражение. Этих подвыражений ровно столько, сколько открывающихся круглых скобок (после которых нет знака вопроса) стоит левее данного элемента.

Обратная ссылка совпадает с конкретным выбранным значением подвыражения, на которое она ссылается, а не с любым возможным значением этого подвыражения. Таким образом, шаблон

(ответствен|надеж)ный проявляет \1ность

совпадет со строками «ответственный проявляет ответственность», «надежный проявляет надежность» и не совпадет со строкой «ответственный проявляет надежность».

Обратные ссылки могут использоваться внутри подвыражений. При первом использовании подвыражения ссылка внутри него не срабатывает, но при последующих повторениях подшаблона она работает, как описано выше.

Утверждения

Утверждение – это проверка символов, следующих до или после текущего символа. Простейшие утверждения закодированы последовательностями \A, \Z, ^, $ и т. д. Более сложные утверждения кодируются с помощью подшаблонов.

Существует два типа утверждений: те, что смотрят на текущую позицию в исходной строке («смотрящие вперед»), и те, что смотрят на символы перед текущей позицией («смотрящие назад»).

Утверждения, закодированные подшаблонами, сравниваются как обычные подшаблоны, за исключением того, что при их обработке не происходит изменения текущей позиции.

«Смотрящие вперед» утверждения ищут совпадения в строке за текущей позицией поиска и начинаются с (?= для позитивных утверждений и с (?! для негативных. Например,

\w+(?=;)

совпадает со словом, заканчивающимся точкой с запятой (не включая точку с запятой в результат поиска), и

foo(?!bar)

совпадает с любым появлением foo, после которого нет bar. Как все происходит? Берем строку и ищем в ней foo. Как только нашли, заглядываем вперед (текущая позиция при этом не меняется) и смотрим, идет ли далее bar. Если нет, то совпадение с шаблоном найдено, иначе продолжаем поиск.

Регулярное выражение

(?!foo)bar

не найдет все вхождения bar, перед которыми нет foo, потому что оно «смотрит вперед», а перед ним никаких символов нет. Поэтому в данном шаблоне ?!foo всегда верно.

«Смотрящие назад» утверждения ищут совпадения перед текущей позицией. Позитивные утверждения этого типа начинаются с (?<=, негативные – с (?<! . Смотрящим назад утверждениям позволено искать только строки фиксированной длины, т. е. в них нельзя использовать квантификаторы. Например,

(?<!foo)bar

находит все появления bar, перед которыми нет foo.

Условные подвыражения

Как в любом языке программирования, в регулярных выражениях существуют условные конструкции. Применяются они к подвыражениям. То есть можно заставить процессор регулярных выражений выбирать подшаблон в зависимости от условия или выбирать между двумя альтернативными шаблонами в зависимости от результата утверждения или от того, совпал ли предыдущий захваченный подшаблон.

Существуют две формы условных подвыражений:

(?(условие)шаблон_выполняемый_если_условие_верно)

(?(условие)шаблон_если_условие_верно
|шаблон_если_условие_неверно)

Существует два типа условий. Если текст между круглыми скобками состоит из последовательности цифр, то условие удовлетворяется, если захваченное подвыражение с этим номером ранее совпало.

(\( )? [^()]+ (?(1) \))

Первая часть этого регулярного выражения опционально совпадает с открывающейся круглой скобкой, и если этот символ присутствует, то устанавливает его как первое захваченное подвыражение.

Вторая часть совпадает с одним или более символами, не заключенными в круглые скобки.

Третья часть – это условное подвыражение, которое проверяет, совпало ли первое множество скобок или нет (попалась ли нам в строке открывающая круглая скобка). Если попалась, т. е. объект (строка) начинается с символа " (", то условие верно и вычисляется условный шаблон, а именно требуется наличие закрывающей круглой скобки. В противном случае подшаблон ни с чем не совпадает.

Если условие – не последовательность цифр, то оно должно быть утверждением. Это может быть позитивное или негативное "смотрящее вперед" или "смотрящее назад" утверждение.

(?(?=[^a-z]*[a-z])\d{2}-[a-z]{3}-\d{2}|\d{2}-\d{2}-\d{2})

Условие здесь – позитивное «смотрящее вперед» утверждение. Оно совпадает с любой последовательностью не букв, после которых идет буква. Другими словами, оно проверяет присутствие хотя бы одной буквы в строке для поиска. Если буква найдена, то производится сравнение по первому альтернативному варианту шаблона (\d{2}-[a-z]{3}-\d{2}), иначе – по второму (\d{2}-\d{2}-\d{2}). Этому шаблону удовлетворяют строки двух типов:

dd-aaa-dd или dd-dd-dd,

где d – любая цифра, a – любая буква.

4.11. Использование шаблонов в PHP

Шаблон в языке программирования – это текст с переменными внутри него. При обработке шаблона происходит замена переменных на их значения. Шаблоны нужны для того, чтобы отделить логику работы приложения от способа представления данных, т. е. от дизайна.

Чаще всего для того, чтобы работать с шаблонами, создают библиотеки классов. В принципе создавать свою библиотеку не обязательно, поскольку существует множество свободно распространяемых библиотек шаблонов, над функциональностью которых трудятся большие коллективы разработчиков, стараясь сделать их универсальными, мощными и быстрыми. Некоторые из таких библиотек мы и рассмотрим. Но для начала сформулируем задачу, на примере решения которой будем демонстрировать использование различных шаблонов.

Итак, задача: требуется сгенерировать web-страницу со списком статей, имеющихся в базе данных. Для простоты считаем, что статья имеет название title, автора author, краткое содержание abstract и полное содержание fulltext, которое представлено либо в виде текста в базе данных, либо в виде ссылки на файл. Список должен быть организован так, чтобы при щелчке мышью на названии статьи ее полное содержание появлялось в новом окне.

Шаблоны подстановки

Как можно решить такую задачу способом простой подстановки?

Нужно придумать шаблон для этой страницы и где-то его хранить (в файле или в базе данных). Очевидно, что мы не можем придумать шаблон для всей страницы, потому что не знаем, сколько статей в базе данных. В шаблоне же мы договорились использовать только html и метасимволы <!имя элемента>. Поэтому мы можем написать только шаблон для одной строки списка, который уже программно надо преобразовать в нужное количество строк.

```
<li><a href="<!fulltext>"
 target=new><!title></a>
(<!author>)<br><p><!abstract></p>
```

Кроме того, здесь есть еще одно препятствие – с отображением ссылки на полный текст статьи. Если мы будем действовать по правилу подстановки (менять все метасимволы на их значения из базы данных), то может получиться, что вместо <!fulltext> вставим не ссылку на текст, а сам текст. То есть для этого элемента нужна дополнительная проверка перед заменой и какие-то дополнительные действия в случае, если в поле fulltext содержится текст статьи, а не ссылка на файл. Не будем усложнять себе жизнь и договоримся, что в поле fulltext всегда содержится только ссылка на файл. Тогда задачу можно решить следующим образом:

```
<?
$li_tmpl = file_get_contents("tmpl.html");
// считываем шаблон строки из файла
// устанавливаем соединение и выбираем
// базу данных
$conn = mysql_connect("localhost", "user","123")
or die("Cant connect");
mysql_select_db("book");
```

```
$sql = "SELECT * FROM Articles";
$q = mysql_query($sql,$conn);
   // отправляем запрос
$num = mysql_num_rows($q);
for($i=0; $i<$num; $i++){
      $tmpl .= $li_tmpl;
      $tmpl = str_replace("<!title>",
      mysql_result($q,$i,"title"),$tmpl);
      $tmpl = str_replace("<!author>",
      mysql_result($q,$i,"author"),$tmpl);
      $tmpl = str_replace("<!abstract>",
      mysql_result($q,$i,"abstract"),$tmpl);
      $tmpl = str_replace("<!fulltext>",
      mysql_result($q,$i,"fulltext"),$tmpl);
}
echo $tmpl;
?>
```

Для решения задач, где требуется делать подстановку целых блоков или даже проверять условия, создают классы шаблонов, такие как FastTemplate и Smarty.

FastTemplate – это набор классов, позволяющих реализовать работу с шаблонами. Логику добавить в шаблон FastTemplate нельзя, вся она должна находиться в коде программы. Идея работы шаблонов FastTemplate заключается в том, что та или иная большая страница состоит из множества кусочков, самые маленькие из которых – обычные строки текста, и они получают имя и значение.

Файл шаблона FastTemplate – это обычный html-файл, в котором могут встречаться переменные особого вида, впоследствии обрабатываемые методами класса FastTemplate.

Синтаксис переменных в шаблонах FastTemplate описывается следующим выражением: {([A-Z0-9_]+)}

Это значит, что переменная должна начинаться с фигурной скобки "{". Второй и последующие символы должны быть буквами верхнего регистра от A до Z, цифрами или символами подчеркивания. Переменная вычисляется с помощью закрывающей фигурной скобки "}". Например:

```
{TITLE}
{AUTH20}
{TOP_OF_PAGE}
```

Как уже было изложено, основная идея FastTemplate – создание страницы с помощью вложенных шаблонов. Например, для решения нашей задачи можно создать три файла шаблона:

main.tpl (Этот шаблон будет выводить страницу в целом)

```
<html>
<head><title>{TITLE_}</title>
</head>
<body>
{MAIN}
</body>
</html>
```

list.tpl (будет описывать, как выводить список в целом)

```
<ul>
{LIST_ELEMENT}
</ul>
```

list_element.tpl (описывает непосредственно элемент списка)

```
<li><a href="{FULLTEXT}">{TITLE}</a>
({AUTHOR})
<br> <p> {ABSTRACT}
```

Шаблоны созданы – работа дизайнера выполнена. Теперь нужно научиться их обрабатывать, т. е. выполнить работу программиста. Создадим программу для обработки приведенных выше шаблонов.

Перед началом работы с шаблонами FastTemplate нужно подключить этот набор классов к нашей программе. В реальной жизни набор классов FastTemplate записан в один файл, как правило, с названием class.FastTemplate.php3, поэтому подключить его можно, например, с помощью команды:

```
include("class.FastTemplate.php3");
```

Следующий важный шаг – это создание объекта класса FastTemplate, с которым впоследствии мы будем работать:

```
$tpl = new FastTemplate("/path/to/templates");
```

В качестве параметра передается путь к месту, где находятся наши шаблоны.

Методы FastTemplate

Для работы с FastTemplate нужно знать четыре основных метода: define, assign, parse и FastPrint.

Метод define

Синтаксис:

```
define( array ( ключ => значение, ключ1 => значение1, ... ))
```

Метод define() связывает имя файла шаблона с более коротким именем, которое можно будет использовать в программе. То есть «ключ» – это имя, которое мы будем использовать в программе для ссылки на файл шаблона, имя которого записано в строке "значение". Реальные имена файлов шаблонов не рекомендуется использовать нигде, кроме метода define. При вызове метода define() происходит загрузка всех определенных в нем шаблонов.

```
$tpl->define( array (main => "main.tpl",
   list_f => "list.tpl", list_el=> "list_element.tpl" ));
```

Здесь мы задаем псевдонимы именам файлов шаблонов. Эти псевдонимы, т. е. переменные main, list_f и list_el, будут использоваться в программе вместо соответствующих имен файлов main.tpl, list.tpl и list_element.tpl.

Метод assign

Синтаксис:

```
assign( (пары ключ/значение) или
     ( array(пары ключ/значение) )
```

Метод assign() присваивает переменным значения, «ключ» – это имя переменной, а «значение» – значение, которое ей нужно присвоить. Чтобы переменная в шаблоне была заменена значением, это значение нужно задать ей с помощью метода assign(). Согласно синтаксису, этот метод можно использовать в двух различных формах. В FastTemplate есть только один массив, поэтому, если вы повторно задаете значение одному и тому же ключу, оно будет перезаписано.

Пример использования метода assign()

```
$tpl->assign(array(
   TITLE => "Установка и настройка ПО",
   TITLE => "Введение в PHP" ));
```

Здесь мы дважды устанавливаем значение переменной, доступной в файлах шаблона по имени TITLE. Эта переменная будет иметь последнее присвоенное ей значение, т.е. она равна строке "Введение в PHP".

Метод parse

Синтаксис:

```
parse (возвращаемая переменная,  источники шаблонов)
```

Метод parse() – самый основной в FastTemplate. Он устанавливает значение возвращаемой переменной равным обработанным шаб-

лонам из указанных источников. Метод может использоваться в трех формах: простой, составной и присоединяющей.

```
$tpl->parse(MAIN, "main");                    // простая форма
$tpl->parse(MAIN, array ("list_f", "main"));  // составная форма
$tpl->parse(MAIN, ".list_el");                // присоединяющая форма
```

В простой форме шаблон с псевдонимом "main" загружается (если еще не был загружен), все его переменные подставляются, и результат сохраняется как значение переменной MAIN. Если переменная {MAIN} появится в более поздних шаблонах, то вместо нее будет подставлено значение, полученное в результате обработки шаблона "main". Это позволяет создавать вложенные шаблоны.

Составная форма метода parse() создана для того, чтобы упростить вложение шаблонов друг в друга.

Следующие записи эквивалентны:

```
$tpl->parse(MAIN, "list_f");
$tpl->parse(MAIN, ".main");
```

это то же самое, что и

```
$tpl->parse(MAIN, array("list_f", "main"));
```

Когда используется составная форма, важно, чтобы каждый шаблон, идущий в списке после первого, содержал ту переменную, в которую передаются результаты обработки шаблона. В примере выше main должен содержать переменную {MAIN}, поскольку именно в нее передаются результаты обработки шаблона list_f. Если main не содержит переменной {MAIN}, то результаты обработки шаблона list_f будут потеряны.

Присоединяющий стиль позволяет добавлять результаты обработки шаблона к переменной результата. Точка перед псевдонимом файла шаблона говорит FastTemplate о том, что нужно присоединить результат обработки этого шаблона к возвращенным результатам, а не перезаписывать его. Такой стиль наиболее часто используется при построении таблиц с переменным числом рядов, получаемых, например, в результате запроса к базе данных.

Метод FastPrint

Синтаксис:

FastPrint(обработанная переменная)

Метод FastPrint() печатает содержимое переданной в него обработанной переменной. Если он вызван без параметров, то печатается

последняя использованная методом parse() переменная. Пример использования метода FastPrint()

```
$tpl->parse(MAIN, array("list_f", "main"));$tpl->FastPrint();
/* если продолжать предыдущий пример,
то эта функция напечатает значение переменной MAIN */
$tpl->FastPrint("MAIN");   // эта функция сделает тоже самое
```

Если нужно печатать не на экран, а, например, в файл, то получить ссылку на данные можно с помощью метода *fetch()*.

```
$data = $tpl->fetch("MAIN");
fwrite($fd, $data); // запись данных в файл
```

Теперь попробуем собрать воедино все изученные методы, чтобы решить нашу задачу.

```
<?php
include("class.FastTemplate.php3");
                  //подключаем класс  шаблонов FastTemplate
$tpl = new FastTemplate("c:/users/user/tasks/");
                        //создаем  объект FastTemplate
                 //задаем псевдонимы для имен файлов шаблонов
$tpl->define( array( main => "main.tpl",
               list_f => "list.tpl",  list_el=> "list_element.tpl" ));
// Присваиваем переменной TITLE_ значение "List of articles"
$tpl->assign(TITLE_, "List of articles");
      /* далее, как и раньше, устанавливаем соединение с базой
           и получаем из нее значения нужных элементов */
$conn = mysql_connect("localhost","user","123")
  or die("Cant connect");
mysql_select_db("book");
$sql = "SELECT * FROM Articles";
$q = mysql_query($sql,$conn);
$num = mysql_num_rows($q);
  for($i=0; $i<$num; $i++){
    $title = mysql_result($q,$i,"title");
    $author = mysql_result($q,$i,"author");
    $abs = mysql_result($q,$i,"abstract");
    $full = mysql_result($q,$i,"fulltext");
            // присваиваем полученные значения переменным,
```

```
            // использованным внутри шаблонов
        $tpl->assign(array(
                TITLE => $title,
                AUTHOR => $author,
                ABSTRACT => $abs,
                FULLTEXT => $full ));
    /* подставляем вместо переменных значения в шаблоне list_el и
    добавляем полученное к переменной LIST_ELEMENT */
      $tpl->parse(LIST_ELEMENT,".list_el");
    }               //подставляем значения в шаблоны list_f и main
    $tpl->parse(MAIN, array("list_f","main"));
    Header("Content-type: text/plain");
    $tpl->FastPrint(); // выводим обработанный шаблон на экран
    exit;
    ?>
```

Этот класс шаблонов появился еще до выхода PHP 4 для работы с PHP 3. Чтобы протестировать приведенные примеры, нужно скачать библиотеку классов FastTemplate и скопировать этот файл в свою рабочую директорию. Если вы работаете с PHP 4, то в файл class.FastTemplate.php3 нужно внести пару изменений, о которых написано в документации, поставляющейся вместе с этой библиотекой.

Шаблоны Smarty

Smarty – один из действующих проектов PHP, его официальный сайт – http://www.smarty.net/. Этот набор классов для обработки шаблонов – гораздо более мощный и функциональный, чем FastTemplate. Smarty от классов шаблонов типа FastTemplate прежде всего отличается тем, что он не отделяет полностью логику от содержания. Логика, касающаяся отображения данных, может присутствовать в шаблоне, считают разработчики Smarty. Поэтому в шаблоне Smarty могут быть условные операторы, операторы вставки файлов, операторы изменения переменных, циклы и т. п. Другая особенность Smarty – это компиляция шаблонов. Шаблоны переводятся в php-код, и интерпретатор PHP производит все необходимые действия по подстановке значений. Для ускорения работы скомпилированные шаблоны кэшируются.

Механизм Smarty гораздо более сложен, чем тот же FastTemplate, но зато и более функционален.

Вопросы для самопроверки

1. JavaScript. Язык клиентских сценариев. Объект navigator, свойства браузера, объект document и свойства документа.
2. JavaScript. Арифметические, логические и операторы сравнения.
3. JavaScript. Ввод и вывод данных средствами JavaScript. Типы данных.
4. JavaScript. Динамический HTML: программирование окон и фреймов, свойства документа, работа с формами.
5. JavaScript. Операторы цикла и условного перехода.
6. JavaScript. Переменные и их область действия. Операторы языка JavaScript.
7. JavaScript. Символы-разделители и переводы строк. Комментарии. Литералы. Идентификаторы.
8. JavaScript. События мыши и клавиатурные события. Фокусные события и другие события.
9. JavaScript. Сценарии обработки событий. Объект EVENT и его атрибуты.
10. JavaScript. Числа и работа с числами. Логические значения в JavaScript.
11. JavaScript. Гиперссылки (метки) и массив объектов links (anchors). Объект Date и его методы.
12. JavaScript. Доступ к значениям элементов форм. Объект frames и динамическое создание фрейма.
13. JavaScript. Объект Array. Обобщенные объекты.
14. JavaScript. Объект images и его свойства. Динамическое изменение изображений, создание простейших анимаций, подгрузка рисунков.
15. JavaScript. Объект Math, его свойства и методы. Объект String, его свойства и методы.
16. JavaScript. Свойства объекта document.body, метод document.write() и объект history. Объект window и параметры метода window.open().
17. Назначение DHTML.
18. Языки программирования, используемые совместно в DHTML.
19. DHTML.События, определяемые для всех элементов.
20. DHTML. События, определяемые для части элементов.
21. DHTML. Обработка сабытий.

22. Назначение языка JavaScript. Пример создания кнопки на языке JavaScript.

23. Понятие иерархии объектов на языке JavaScript.

24. JavaScript. Манипулирование объектами. Пример.

25. JavaScript. Управление объектами формы. Пример.

26. XML. Document Type Definition (DTD). Допустимый XML.

27. XML. Корректно сформированный и валидный документ.

28. XML. Применение и его преимущества. Эволюция XML.

29. XML. Синтаксис и составные части XML-документов. Приложения XML.

30. Атрибут style и элемент style. Стилевое оформление форм.

31. Динамический HTML: доступ к элементам страницы, модификация элементов и их атрибутов, работа со стилями.

32. Классическая архитектура службы WWW и ее составляющие.

33. Методы позиционирования компонентов HTML-страниц: таблицы, фреймы, CSS.

34. PHP: Простейшие типы данных. Массивы.

35. PHP. Типы данных и значения. Управляющие конструкции.

36. Создать отформатированный текст в HTML формате.

37. Создать нумерованный и маркированный списки в HTML формате.

38. Создать таблицу с помощью любого стиля, содержащую 3 строки и 4 столбца в формате HTML.

39. Создать страницу, содержащую 2 рисунка, находящихся по центру страницы и 3 ссылки на другие страницы.

40. Создать страницу с формой «Анкета для регистрации», содержащую текстовые поля, радио кнопки, флаги и кнопки обработки данных.

41. Создать страницу, содержащую 3 фрейма.

42. Средствами языка php создать страницу, выводящую Ф.И.О. студента и текущую дату.

43. PHP. Оператор swich.

44. PHP. Оператор continue и break.

45. PHP. Создать страницу, выводящую числа от 1 до 10 с помощью цикла for.

46. PHP. Создать страницу, выводящую числа от 1 до 10 с помощью цикла while.

47. PHP. Создать страницу, выводящую числа от 1 до 10 с помощью цикла do…while.

48. PHP. Создать страницу, выводящую элементы массива, присвоенные автоматически.

49. PHP. Создать страницу, выводящую элементы массива, присвоенные явно.

50. PHP. Создать страницу, выводящую элементы массива в таблицу.

51. PHP. Создать страницу, выводящую элементы массива, с помощью функции print_r.

52. PHP. Создать страницу, которая бы складывала два числа, вводящиеся через форму.

53. PHP. Создать страницу, которая бы вычитала два числа, вводящие через форму.

54. PHP. Создать страницу, которая бы умножала два числа, вводящие через форму.

55. PHP. Создать страницу, которая бы делила два числа, вводящие через форму.

56. PHP. Создать калькулятор.

57. PHP. Создать прототип интернет-магазина.

Литература

1. Наварро, Э. XHTML : учеб. курс / Э. Наварро. – СПб. : Питер, 2001.

2. Тиге, Дж. К. DHTML и CSS для Internet / Дж. Тиге. – М. : НТ Пресс, 2005.

3. Зельдман, Д. Web-дизайн по стандартам / Д. Зельдман. – М. : НТ Пресс, 2005.

4. Дилип Найк Dynamic HTML: Стандарты и протоколы Интернета «Русская редакция» ТОО «Channel Trading Ltd.», 2005.

5. Джамса, К. Эффективный самоучитель по креативному Web-дизайну. HTML, XHTML, CSS, JavaScript, PHP, ASP, ActiveX. Текст, графика, звук и анимация / К. Джамса. – М. : ДиаСофтЮП, 2005.

6. Олифер, В. Компьютерные сети. Принципы, технологии, протоколы / В. Олифер, Н. Олифер. – СПб. : Питер, 2003.

7. Котеров, Д. Самоучитель PHP 4. / Д. Котеров. – СПб. : БХВ, 2001.

8. Колесниченко, Д. Н. Самоучитель PHP 5 / Д. Н. Колесниченко. – СПб. : Наука и техника, 2004.

9. Мельников, Д. А. Информационные процессы в компьютерных сетях. Протоколы, стандарты, интерфейсы, модели / Д. А. Мельников. – М. : КУДИЦ-ОБРАЗ, 2001. – 256 с.

10. Романец, Ю. В. Защита информации в компьютерных системах и сетях / Ю. В. Романец, П. А. Тимофеев, В. Ф. Шаньгин. – 2-е изд., перераб., доп. – М. : Радио и связь, 2001. – 376 с.

Оглавление

Учебное электронное издание комбинированного распространения

Учебное издание

Стефановский Игорь Леонидович
Титова Людмила Константиновна

ПРОГРАММИРОВАНИЕ В INTERNET

Учебно-методическое пособие

Электронный аналог печатного издания

Редактор *Н. Г. Мансурова*
Компьютерная верстка *Н. Б. Козловская*

Подписано в печать 05.05.15.
Формат 60x84/$_{16}$. Бумага офсетная. Гарнитура «Таймс».
Цифровая печать. Усл. печ. л. 6,74. Уч.-изд. л. 6,25.
Изд. № 122.
http://www.gstu.by

Издатель и полиграфическое исполнение
Гомельский государственный
технический университет имени П. О. Сухого.
Свидетельство о гос. регистрации в качестве издателя
печатных изданий за № 1/273 от 04.04.2014 г.
246746, г. Гомель, пр. Октября, 48