Министерство образования Республики Беларусь

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

Институт повышения квалификации
и переподготовки

Кафедра «Информатика»

Н. В. Самовендюк

СЕРВЕРНЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ WEB-САЙТОВ

ПОСОБИЕ
по одноименной дисциплине
для слушателей специальности переподготовки
1-40 01 74 «Web-дизайн и компьютерная графика»
заочной формы обучения

Гомель 2019

УДК 004.42(075.8)
ББК 32.973.22я73
С32

Рекомендовано научно-методическим советом факультета автоматизированных и информационных систем ГГТУ им. П. О. Сухого
(*протокол № 9 от 26.12.2018 г.*)

Составитель *Н. В. Самовендюк*

Рецензент: доц. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *В. В. Комраков*

Серверные технологии разработки web-сайтов : пособие по одноим. дисциплине
С32 для слушателей специальности переподготовки 1-40 01 74 «Web-дизайн и компьютерная графика» заоч. формы обучения / сост. Н. В. Самовендюк. – Гомель : ГГТУ им. П. О. Сухого, 2019. – 160 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: http://elib.gstu.by. – Загл. с титул. экрана.

Рассмотрены основные возможности серверного языка PHP для решения типичных задач разработки web-сайтов, таких как прием и обработка на сервере данных форм, отправленных клиентом пользователя, взаимодействие с базами данных, сохранение данных пользователя на стороне сервера с использованием сеансов, загрузка и обработка файлов.

Для слушателей специальности 1-40 01 74 «Web-дизайн и компьютерная графика» заочной формы обучения ИПКиП.

УДК 004.42(075.8)
ББК 32.973.22я73

СОДЕРЖАНИЕ

Введение

Серверные языки программирования нужны для реализации бизнес-логики работы любого web-сайта или web-приложения, разработчик при помощи языка программирования описывает возможные сценарии использования сайта или приложения.

Языков программирования, используемых для серверной web-разработки, достаточно много: PHP, Ruby, Java, C#, Python, Perl и другие.

По данным исследования, проводимого компанией W3Techs, на декабрь 2018 года почти 79% всех web-сайтов в качестве серверного языка используют язык PHP (сокращение от PHP: Hypertext Preprocessor - PHP: Препроцессор гипертекста). Результаты этого исследования обновляются и приводятся по адресу: https://w3techs.com/technologies/overview/programming_language/all.

Кроме того, на PHP написаны такие сайты, как Facebook, Wikipedia, Twitter, Vk и многие другие.

PHP был создан в 1994 году датским программистом Расмусом Лердорфом и изначально представлял собой набор скриптов на другом языке Perl. Позже этот набор скриптов был переписан в интерпретатор на языке Си. Целью возникновения PHP являлось создание удобного набора инструментов для упрощенной разработки web-сайтов и web-приложений.

1 Программное обеспечение для серверной разработки web-сайтов

Любое web-приложение строиться на основе клиент-серверных технологий. Это значит, что оно состоит из двух частей – серверной и клиентской. Серверная часть, как правило, располагается на удаленном компьютере и управляется специальной программой, web-сервером. Клиентская часть web-приложения – это браузер, установленный на компьютере клиента. Работа web-приложения сводится к диалогу между клиентским браузером и web-сервером.

Есть разные способы установки всего необходимого программного обеспечения. Мы можем устанавливать компоненты по отдельности, а можем использовать уже готовые сборки типа Denwer или EasyPHP. В подобных сборках компоненты уже имеют начальную настройку и уже готовы для создания web-сайтов.

Что подразумевает установка PHP? Во-первых, нам нужен интерпретатор PHP. Во-вторых, необходим web-сервер, с помощью которого мы сможем обращаться к ресурсам создаваемого нами сайта. В-третьих, поскольку мы будем использовать базы данных, то нам также надо будет установить какую-нибудь систему управления базами данных.

Для установки PHP достаточно перейти на сайт разработчиков http://php.net/. На странице загрузок можно найти различные дистрибутивы для операционной системы Linux. Если используется операционная система Windows, то необходимо загрузить один из пакетов со страницы http://windows.php.net/download/.

Для написания серверных сценариев достаточно использовать любой текстовый редактор. Однако желательно использовать специализированные интеллектуальные редакторы, поддерживающие язык PHP: Notepad++, Brackets, Sublime. Удобным для web-разработки будет использование интегрированной среды разработки на PHP – PhpStorm. Это интегрированная среда разработки на PHP с интеллектуальным редактором, которая глубоко понимает код, поддерживает PHP 5.3-7.2 для современных и классических проектов,

обеспечивает лучшее в индустрии автодополнение кода, рефакторинги, предотвращение ошибок налету и поддерживает смешивание языков.

Для установки web-сервера рекомендуется использование сервера Apache, который является самым распространенным (https://w3techs.com/technologies/overview/web_server/all). Загрузить пакет web-сервера Apache можно с сайта https://httpd.apache.org/download.cgi.

Для создания сайтов на PHP необходима также какая-нибудь СУБД. PHP поддерживает работу с множеством систем баз данных (MySQL, MSSQL, Oracle, Postgre, MongoDB и другие). Можно использовать любую СУБД, однако на сегодняшний день MySQL является наиболее популярной системой управления базами данных для работы с PHP, кроме того, она распространяется по лицензии GNU (свободное программное обеспечение). Для установки MySQL необходимо загрузить дистрибутив по адресу https://dev.mysql.com/downloads/mysql/ и выбрать нужную версию.

Как отмечалось ранее можно использовать готовые сборки для разработки web-сайтов. Одной из таких сборок под ОС Window является OpenServer – интегрированный пакет для web-разработчика, который по умолчанию включает в себя веб-сервер Apache, СУБД MySQL, клиент PhpMyAdmin для работы с БД и ряд других полезных инструментов. Панель управления сервером OSPanel является портативным программным комплексом и не требует установки. Сборку можно разместить на внешнем жёстком диске или флэш-накопителе, что позволит использовать OSPanel на любом компьютере. Одним из преимуществ этого пакета является то, что он легко позволяет «налету» изменять версии используемых Apache и PHP. Бесплатно скачать установщик OpenServer можно по адресу https://ospanel.io/download/.

Запустите загруженный файл и распакуйте его на диск в какую-либо папку. После распаковки в указанной папке будет создана папка OSPanel с вложенными папками и двумя exe файлами, как на рисунке 1:

Рисунок 1 – Папка с установленной панелью управления сервером

В зависимости от разрядности операционной системы необходимо запустить либо Open Server x64 exe, либо Open Server x86 exe. В системном трее появится иконка в виде флага – это иконка OpenServer. Цвет флага говорит о статусе программы: зеленый – работает, красный – остановлена, желтый – в переходном состоянии. Нажав правую кнопку мыши на иконке можно увидеть главное меню OpenServer (рисунок 2).

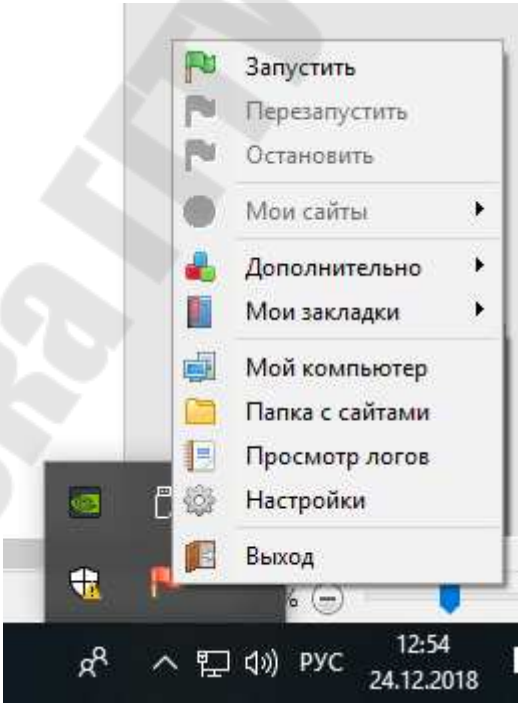


Рисунок 2 – Главное меню OpenServer

Если активизировать пункт меню «Настройки», раскроется окно настроек, в котором можно выбрать вкладку «Модули». Эта вкладка

показывает и позволяет изменять версии текущих модулей: самого PHP, сервера MySQL и других (рисунок 3).

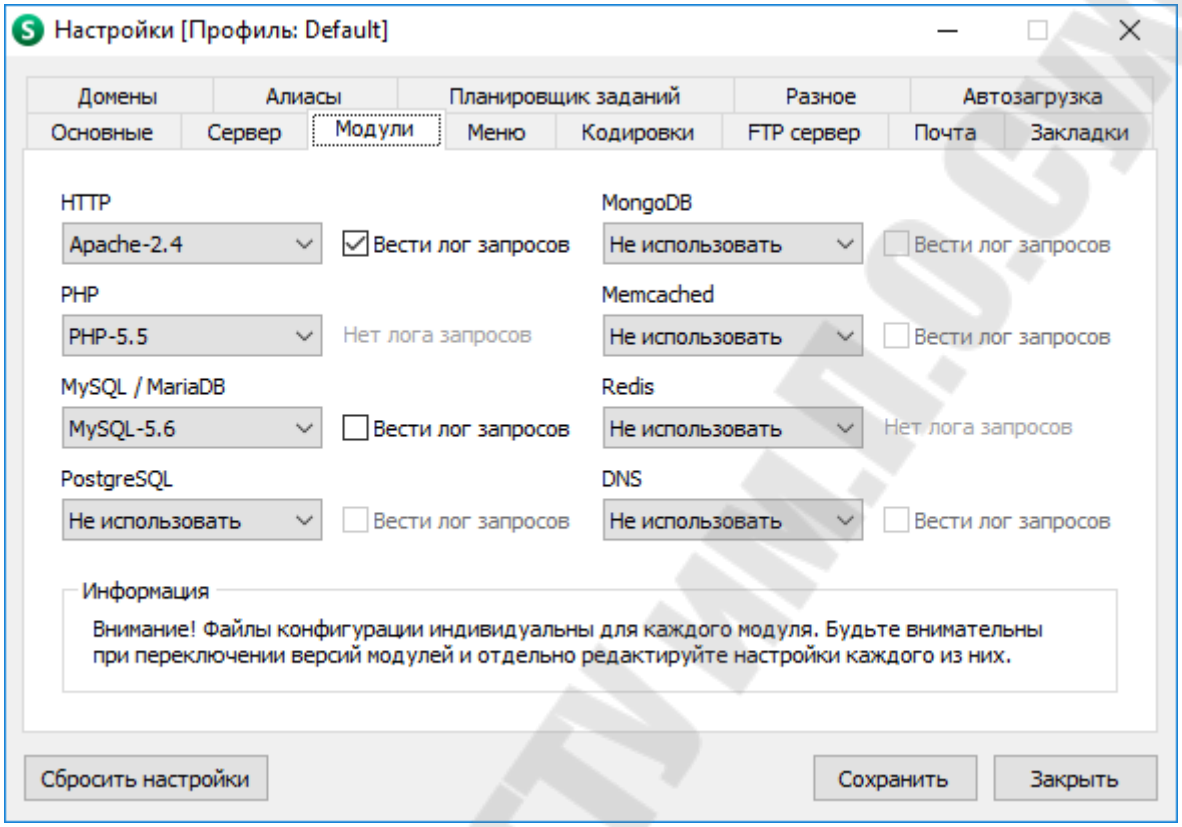


Рисунок 3 – Вкладка «Модули»

Когда возникнет необходимость изменить версию PHP и Apache, можно воспользуемся этой вкладкой. Очень полезной является вкладка Server, на которой отображены используемые программой порты, адреса и другие настройки (рисунок 4). Web-приложение является сетевым, имеющий специальный атрибут под названием сокет, позволяющий приложению передавать свои данные и получать присланные ему данные из сети. По сути, сокет представляет собой комбинацию сетевого адреса компьютера и «номера» конкретного приложения на этом компьютере, которому надо выходить в сеть. Такой номер приложения называется портом и представляет собой целое число в диапазоне 0-65535. Поэтому для выхода в сеть приложение должно иметь собственный номер порта. Для выхода в глобальную сеть по умолчанию используется порт 80 для браузера. У других компонент приложения – свои номера портов.

Если на компьютере какой-либо порт занят и OpenServer не запускается, перейдите на эту вкладку, измените конфликтый номер порта, нажмите кнопки Save, а затем Close, и перезапустите OpenServer Чаще всего, этих действий достаточно, чтобы решить проблему с запуском.

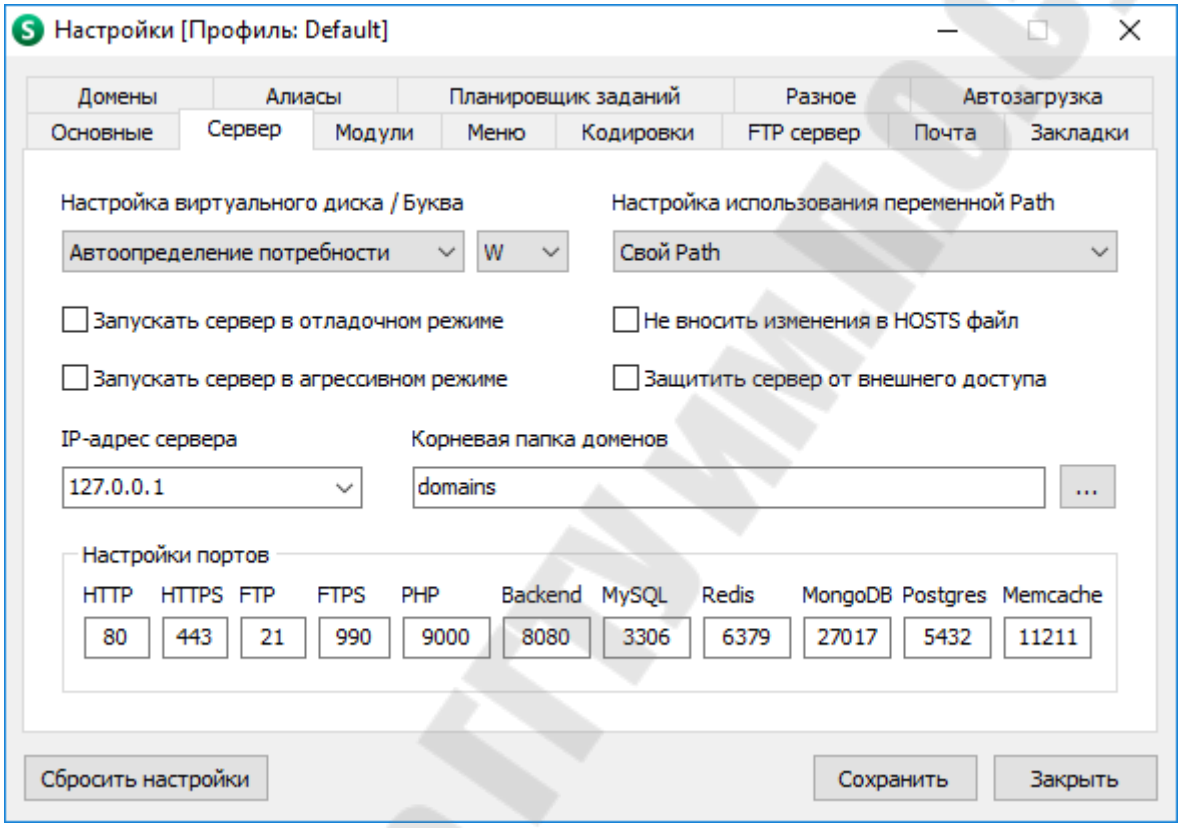


Рисунок 4 – Вкладка «Сервер»

Для создания локального сайта достаточно создать папку с названием сайта в каталоге «domains» и перезапустить OSPanel.

2 Основы PHP

2.1 Синтаксис языка

Программа или скрипт на PHP, как правило, находится в файле с расширением .php. Хотя разработчики могут также вставлять код php и в файлы с расширениями .html/.htm.

Когда пользователь обращается к скрипту в адресной строке браузера, набирая, например, http://localhost/Test/test.php, то web-сервер передает его интерпретатору PHP. Затем интерпретатор обрабатывает код и генерирует на его основе html-разметку. И затем сгенерированный html-код отправляется пользователю.

Сценарий PHP может содержать как разметку html, так и код на языке php. Для перехода от разметки html к коду php используются теги <?php и ?>, между которыми идет код php. Данные теги служат указанием интерпретатору, что их содержимое надо интерпретировать как код php, а не разметку html. Если у вас на странице расположен только php код (без html областей), то закрывающий тег желательно не ставить. Дело в том, что иногда за закрывающим тегом могут случайно добавляться невизуализируемые символы, типа пробелов или новых строк, которые в таком случае могут привести к нежелательному выводу на страницу.

Также можно использовать краткую версию тегов: <? и ?>. Для этого в файле php.ini надо изменить значение short_open_tag = Off на short_open_tag = On.

Программа на PHP – это набор команд (инструкций). Обработчику программы (парсеру) необходимо как-то отличать одну команду от другой. Для этого используются специальные символы – разделители. В PHP инструкции разделяются так же, как и в Си или Perl, – каждое выражение заканчивается точкой с запятой.

Закрывающий тег «?>» также подразумевает конец инструкции, поэтому перед ним точку с запятой не ставят. Например, два следующих фрагмента кода эквивалентны:

```php
<?php
echo "Hello, world!"; // точка с запятой
                      // в конце команды
                      // обязательна
?>

<?php
echo "Hello, world!" ?>
<!-- точка с запятой
    опускается из-за "?>" -->
```

2.2 Комментарии

Часто при написании программ возникает необходимость делать какие-либо комментарии к коду, которые никак не влияют на сам код, а только поясняют его. Это важно при создании больших программ и в случае, если несколько человек работают над одной программой. При наличии комментариев в программе в ее коде разобраться гораздо проще. Кроме того, если решать задачу по частям, недоделанные части решения также удобно комментировать, чтобы не забыть о них в дальнейшем. Во всех языках программирования предусмотрена возможность включать комментарии в код программы. PHP поддерживает несколько видов комментариев: в стиле Си, C++ и оболочки Unix. Символы // и # обозначают начало однострочных комментариев, /* и */ – соответственно начало и конец многострочных комментариев.

```php
<?php
echo "Меня зовут Вася";
  // Это однострочный комментарий
  // в стиле C++
echo "Фамилия моя Петров";
/* Это многострочный комментарий.
Здесь можно написать несколько строк.
При исполнении программы все, что
находится здесь (закомментировано),
будет игнорировано. */
echo "Я изучаю PHP";
  # Это комментарий в стиле
  # оболочки Unix
?>
```

2.3 Переменные, константы и операторы

Важным элементом каждого языка являются переменные, константы и операторы, применяемые к этим переменным и константам. Рассмотрим, как выделяются и обрабатываются эти элементы в PHP.

2.3.1 Переменные

Как и во многих языках программирования, в PHP есть переменные. Переменные хранят отдельные значения, которые можно использовать в выражениях на PHP. Для обозначения переменных используется знак доллара $. Например:

```
$my_var;
```

Имя переменной чувствительно к регистру, т.е. переменные $my_var и $My_var различны. Имена переменных соответствуют тем же правилам, что и остальные наименования в PHP: правильное имя переменной должно начинаться с буквы или символа подчеркивания с последующими в любом количестве буквами, цифрами или символами подчеркивания.

В PHP 3 переменные всегда присваивались по значению. То есть когда вы присваиваете выражение переменной, все значения оригинального выражения копируются в эту переменную. Это означает, к примеру, что после присвоения одной переменной значения другой, изменение одной из них не влияет на значение другой.

```
<?php
$first = ' Text ';// Присваиваем $first
                  // значение
                  // ' Text '
$second = $first; // Присваиваем $second
                  // значение
                  // переменной $first
$first = ' New text '; // Изменяем
                       // значение
                       // $first
                       // на ' New text '
echo "Переменная с именем first " .
     "равна $first <br>";
        // выводим значение $first
echo "Переменная с именем second " .
     "равна $second";
        // выводим значение $second
?>
```

Результат работы этого скрипта будет следующим:
Переменная с именем first равна New text
Переменная с именем second равна Text

В PHP 4 появился еще один способ присвоения значений переменным – присвоение по ссылке. Для того, чтобы присвоить значение переменной по ссылке, это значение должно иметь имя, т.е. оно должно быть представлено какой-либо переменной. Чтобы указать, что значение одной переменной присваивается другой переменной по ссылке, нужно перед именем первой переменной поставить знак амперсанд &.

Рассмотрим тот же пример, что и выше, только будем присваивать значение *переменной* first *переменной* second по ссылке:

```
<?php
$first = ' Text '; // Присваиваем $first
                   // значение ' Text '
$second = &$first;
/* Делаем ссылку на $first через $second.
   Теперь значения этих переменных
   будут всегда совпадать */
// Изменим значение $first
// на ' New text '
$first = ' New text ';
```

```
echo "Переменная с именем first " .
     "равна $first <br>";
// выведем значения обеих переменных
echo "Переменная с именем second " .
     "равна $second";
?>
```

Этот скрипт выведет следующее:

Переменная с именем first равна New text.

Переменная с именем second равна New text.

То есть вместе с переменной $first изменилась и переменная $second.

В PHP 5 появилась возможность использования динамических переменных, имя которой формируется «на лету»

```
<?php
// Создание динамических переменных
$client = 'user'; // Присваиваем $client
                  // значение 'user'
$$client = 'Nick';
// Создается новая переменная $user со значением Nick
?>
```

2.3.2 Константы

Для хранения постоянных величин, т.е. таких величин, значение которых не меняется в ходе выполнения скрипта, используются константы. Такими величинами могут быть математические константы, пароли, пути к файлам и т.п. Основное отличие константы от переменной состоит в том, что ей нельзя присвоить значение больше одного раза и ее значение нельзя аннулировать после ее объявления. Кроме того, у константы нет приставки в виде знака доллара и ее нельзя определить простым присваиванием значения. Для создания констант используется специальная функция define(). Ее синтаксис таков:

define("Имя_константы",
    "Значение_константы",

[Нечувствительность_к_регистру])

По умолчанию имена констант чувствительны к регистру. Для каждой константы это можно изменить, указав в качестве значения аргумента «Нечувствительность_к_регистру» значение True. Существует соглашение, по которому имена констант всегда пишутся в верхнем регистре.

Получить значение константы можно, указав ее имя. В отличие от переменных, не нужно предварять имя константы символом $. Кроме того, для получения значения константы можно использовать функцию constant() с именем константы в качестве параметра.

```
<?php
// определяем константу
// PASSWORD
define("PASSWORD","qwerty");
// определяем регистронезависимую
// константу PI со значением 3.14
define("PI","3.14", True);
// выведем значение константы PASSWORD,
// т.е. qwerty
echo (PASSWORD);
// тоже выведет qwerty
echo constant("PASSWORD");
echo (password);
/* выведет password и предупреждение,
   поскольку мы ввели регистрозависимую
   константу PASSWORD */
echo pi;
// выведет 3.14, поскольку константа PI
// регистронезависима по определению
?>
```

Кроме констант, объявляемых пользователем, о которых мы только что рассказали, в PHP существует ряд констант, определяемых самим интерпретатором. Например, константа __FILE__ хранит имя файла программы (и путь к нему), которая выполняется в данный момент, __FUNCTION__ содержит имя функции, __CLASS__ – имя класса, PHP_VERSION – версия интерпретатора PHP. Полный список предопределенных констант можно получить, прочитав руководство по PHP.

2.3.3 Операторы

Операторы позволяют выполнять различные действия с переменными, константами и выражениями. Выражение можно определить как все, что угодно, что имеет значение. Переменные и константы – это основные и наиболее простые формы выражений. Существует множество операций (и соответствующих им операторов), которые можно производить с выражениями (см. таблица 1).

Таблица 1 – Основные операторы PHP

Обозначение	Название	Пример
Арифметические операторы		
+	Сложение	$a + $b
-	Вычитание	$a - $b
*	Умножение	$a * $b
/	Деление	$a / $b
%	Остаток от деления	$a % $b
Строковые операторы		
.	*Конкатенация* (*сложение строк*)	$c = $a . $b (это строка, состоящая из $a и $b)
Операторы присваивания		
=	Присваивание	$a = ($b = 4) +5; ($a будет равна 9, $b будет равна 4)

Обозначение	Название	Пример
+=		$a += 5; (эквивалентно $a = $a + 5;)
.=		$b = "Привет "; $b .= "всем"; (эквивалентно $b = $b . "всем";) В результате: $b="Привет всем"
Логические операторы		
and	И	$a and $b
&&	И	$a && $b
or	Или	$a or $b
||	Или	$a || $b
xor	Исключающее или	$a xor $b
!	Инверсия (NOT)	! $a
Операторы сравнения		
==	Равенство	$a == $b
===	Эквивалентность	$a === $b
!=	Неравенство	$a != $b
<>	Неравенство	$a <> $b
!==	Неэквивалентность	$a !== $b
<	Меньше	$a < $b
>	Больше	$a > $b
<=	Меньше или равно	$a <= $b

Обозначение	Название	Пример
>=	Больше или равно	$a >= $b
Операторы инкремента и декремента		
++$a	Пре-*инкремент*	<? $a=4; echo "Должно быть 4:" .$a++; echo "Должно быть 6:" .++$a; ?>
$a++	Пост-*инкремент*	
--$a	Пре-*декремент*	
$a--	Пост-*декремент*	

2.4 Типы данных

PHP поддерживает восемь простых типов данных.

Четыре скалярных типа:

- boolean (логический);
- integer (целый);
- float (с плавающей точкой);
- string (строковый).

Два смешанных типа:

- array (массив);
- object (объект).

И два специальных типа:

- resource (ресурс);
- NULL.

В PHP не принято явное объявление типов переменных. Предпочтительнее, чтобы это делал сам интерпретатор во время выполнения программы в зависимости от контекста, в котором

используется переменная. Рассмотрим по порядку все перечисленные типы данных.

2.4.1 Тип boolean

Тип boolean (булев или логический тип) выражает истинность значения, то есть переменная этого типа может иметь только два значения – истина TRUE или ложь FALSE.

Чтобы определить булев тип, используют ключевое слово TRUE или FALSE. Оба регистронезависимы.

```
<?php
$test = True;
?>
```

Логические переменные используются в различных управляющих конструкциях (циклах, условиях и т.п., более подробно речь о них пойдет в одной из следующих лекций). Иметь логический тип, т.е. принимать только два значения, истину или ложь, могут также и некоторые операторы (например, оператор равенства). Они также используются в управляющих конструкциях для проверки каких-либо условий. Например, в условной конструкции проверяется истинность значения оператора или переменной и в зависимости от результата проверки выполняются те или иные действия. Здесь условие может быть истинно или ложно, что как раз и отражает переменная и оператор логического типа.

```
<?php
// Оператор '==' проверяет равенство
// и возвращает
// булево значение
if ($know == False) { // если $know
                      // имеет значение
                      // false
echo "Изучай PHP!";
}
if (!$know) { // то же самое, что
              // и выше, т.е. проверка
```

```
                // имеет ли $know значение
                // false
    echo "Изучай PHP!";
    }
    /* оператор == проверяет, совпадает ли
       значение переменной $know со строкой
       "Изучай PHP". Если совпадает, то
       возвращает true, иначе - false.
       Если возвращено true, то выполняется
       то, что внутри фигурных скобок */
    if ($know == "Изучай PHP")
    { echo "Начал изучать"; }
    ?>
```

2.4.2 Тип integer (целые)

Этот тип задает число из множества целых чисел Z = {..., -2, -1, 0, 1, 2, ...}. Целые могут быть указаны в десятичной, шестнадцатеричной или восьмеричной системе счисления, по желанию с предшествующим знаком «-» или «+».

Если вы используете восьмеричную систему счисления, вы должны предварить число 0 (нулем), для использования шестнадцатеричной системы нужно поставить перед числом 0x.

```
    <?php
    # десятичное число
    $a = 1234;
    # отрицательное число
    $a = -123;
    # восьмеричное число (эквивалентно
    # 83 в десятичной системе)
    $a = 0123;
    # шестнадцатеричное число (эквивалентно
    # 26 в десятичной системе)
    $a = 0x1A;
    ?>
```

Размер целого зависит от платформы, хотя, как правило, максимальное значение около двух миллиардов (это 32-битное знаковое). Беззнаковые целые PHP не поддерживает.

Если вы определите число, превышающее пределы целого типа, оно будет интерпретировано как число с плавающей точкой. Также если вы используете оператор, результатом работы которого будет число, превышающее пределы целого, вместо него будет возвращено число с плавающей точкой.

В PHP не существует оператора деления целых. Результатом 1/2 будет число с плавающей точкой 0.5. Вы можете привести значение к целому, что всегда округляет его в меньшую сторону, либо использовать функцию round(), округляющую значение по стандартным правилам. Для преобразования переменной к конкретному типу нужно перед переменной указать в скобках нужный тип. Например, для преобразования переменной $a=0.5 к целому типу необходимо написать (integer)(0.5) или (integer) $a или использовать сокращенную запись (int)(0.5). Возможность явного приведения типов по такому принципу существует для всех типов данных (конечно, не всегда значение одного типа можно перевести в другой тип). Мы не будем углубляться во все тонкости приведения типов, поскольку PHP делает это автоматически в зависимости от контекста.

2.4.3 Тип float (числа с плавающей точкой)

Числа с плавающей точкой (они же числа двойной точности или действительные числа) могут быть определены при помощи любого из следующих синтаксисов:

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

Размер числа с плавающей точкой зависит от платформы, хотя максимум, как правило, ~1.8e308 с точностью около 14 десятичных цифр.

2.4.4 Тип string (строки)

Строка – это набор символов. В PHP символ – это то же самое, что байт, это значит, что существует ровно 256 различных символов. Это также означает, что PHP не имеет встроенной поддержки Unicode. В PHP практически не существует ограничений на размер строк, поэтому нет абсолютно никаких причин беспокоиться об их длине.

Строка в PHP может быть определена тремя различными способами:

- с помощью одинарных кавычек;
- с помощью двойных кавычек;
- 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, и они имеют специальные значения, о которых мы расскажем немного позднее. Так вот, в отличие от двух других синтаксисов, переменные и управляющие последовательности

для специальных символов, встречающиеся в строках, заключенных в одинарные кавычки, не обрабатываются.

```
<?php
echo 'Также вы можете вставлять в строки
    символ новой строки таким образом,
    поскольку это нормально';

// Выведет: Чтобы вывести ' надо
// перед ней поставить \
echo 'Чтобы вывести \' надо перед ' .
     'ней поставить \\';
// Выведет: Вы хотите удалить C:\*.*?
echo 'Вы хотите удалить C:\\*.*?';
// Выведет: Это не вставит: \n новую строку
echo 'Это не вставит: \n новую строку';
// Выведет: Переменные $expand также
// $either не подставляются
echo 'Переменные $expand также $either' .
     'не подставляются';
?>
```

Двойные кавычки

Если строка заключена в двойные кавычки «"», PHP распознает большее количество управляющих последовательностей для специальных символов. Самым важным свойством строк в двойных кавычках является обработка переменных.

```
<?php
// Выведет: Переменные $expand также $either
подставляются
echo "Переменные $expand также $either подставляются в
строку";
?>
```

Heredoc

Другой способ определения строк – это использование heredoc-синтаксиса. 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 именования, 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 вам нет необходимости экранировать кавычки в heredoc, но вы по-прежнему можете использовать перечисленные выше управляющие последовательности. Переменные внутри heredoc тоже обрабатываются.

```
<?php
$str = <<<EOD
Пример строки, охватывающей несколько
строчек, с использованием
heredoc-синтаксиса
EOD;
// Здесь идентификатор – EOD. Ниже
// идентификатор EOD
$name = 'Вася';
echo <<<EOD
Меня зовут "$name".
EOD;
// это выведет: Меня зовут "Вася".
?>
```

2.4.5 Тип array (массив)

Массив в PHP представляет собой упорядоченную карту – тип, который преобразует значения в ключи. Этот тип оптимизирован в нескольких направлениях, поэтому вы можете использовать его как собственно массив, список (вектор), хеш-таблицу (являющуюся реализацией карты), стэк, очередь и т.д. Поскольку вы можете иметь в качестве значения другой массив PHP, можно также легко эмулировать деревья.

Определить массив можно с помощью конструкции array() или непосредственно задавая значения его элементам.

Определение при помощи array()

array ([key] => value,
[key1] => value1, ...)

Языковая конструкция array() принимает в качестве параметров пары ключ => значение, разделенные запятыми. Символ => устанавливает соответствие между значением и его ключом. Ключ может быть как целым числом, так и строкой, а значение может быть любого имеющегося в PHP типа. Числовой ключ массива часто называют индексом. Индексирование массива в PHP начинается с нуля. Значение элемента массива можно получить, указав после имени массива в квадратных скобках ключ искомого элемента. Если ключ массива представляет собой стандартную запись целого числа, то он рассматривается как число, в противном случае – как строка. Поэтому запись $a["1"] равносильна записи $a[1], так же как и $a["-1"] равносильно $a[-1].

```php
<?php
$books = array ("php" =>
                "PHP users guide",
                12 => true);
echo $books["php"]  ;
//выведет "PHP users guide"
echo $books[12];
?>
```

Если для элемента ключ не задан, то в качестве ключа берется максимальный числовой ключ, увеличенный на единицу. Если указать ключ, которому уже было присвоено какое-то значение, то оно будет перезаписано. Начиная с PHP 4.3.0, если максимальный ключ – отрицательное число, то следующим ключом массива будет ноль (0).

```php
<?php
// массивы $arr и $arr1 эквивалентны
$arr = array(5 => 43, 32, 56, "b" => 12);
$arr1 = array(5 => 43, 6 => 32,
              7 => 56, "b" => 12);
?>
```

Если использовать в качестве ключа TRUE или FALSE, то его значение переводится соответственно в единицу и ноль типа integer. Если использовать NULL, то вместо ключа получим пустую строку. Можно использовать и саму пустую строку в качестве ключа, при этом ее надо брать в кавычки. Так что это не то же самое, что использование пустых квадратных скобок. Нельзя использовать в качестве ключа массивы и объекты.

Создать массив можно, просто записывая в него значения. Как мы уже говорили, значение элемента массива можно получить с помощью квадратных скобок, внутри которых нужно указать его ключ, например, $book["php"]. Если указать новый ключ и новое значение, например, $book["new_key"]="new_value", то в массив добавится новый элемент. Если мы не укажем ключ, а только присвоим значение $book[]="new_value", то новый элемент массива будет иметь числовой ключ, на единицу больший максимального существующего. Если массив, в который мы добавляем значения, еще не существует, то он будет создан.

```
<?
$books["key"]= value; // добавили в массив
                      // $books значение
                      // value с ключом key
$books[] = value1; /* добавили в массив
                      значение value1 с
                      ключом 13, поскольку
                      максимальный ключ у
                      нас был 12 */
?>
```

Для того чтобы изменить конкретный элемент массива, нужно просто присвоить ему с его ключом новое значение. Изменить ключ элемента нельзя, можно только удалить элемент (пару ключ/значение) и добавить новую. Чтобы удалить элемент массива, нужно использовать функцию unset().

```
<?php
$books = array ("php" =>
                "PHP users guide",
```

```
                    12 => true);

    $books[] =  "Book about Perl"; // добавили элемент
                        // с ключом (индексом)
                        // 13 это эквивалентно
                        // $books[13] =
                        // "Book about Perl";
    $books["lisp"] = 123456; /* Это добавляет к массиву новый
                элемент с ключом "lisp" и
                значением 123456 */
    unset($books[12]); // Это удаляет элемент
                       // с ключом 12 из массива
    unset ($books); // удаляет массив полностью
    ?>
```

Заметим, что, когда используются пустые квадратные скобки, максимальный числовой ключ ищется среди ключей, существующих в массиве с момента последнего переиндексирования. Переиндексировать массив можно с помощью функции array_values().

```
    <?php
    $arr =
     array ("a","b","c"); /* Создаем массив со значениями
"a", "b" и "c".
                             Поскольку ключи не указаны, они
будут 0,1,2
                             соответственно */
    print_r($arr); // выводим массив (и ключи,
                   // и значения)
    unset($arr[0]);
    unset($arr[1]);
    unset($arr[2]);
              // удаляем из него все значения
    print_r($arr);  // выводим массив (и ключи, и значения)
    $arr[] = "aa";  // добавляем новый элемент
                    // в массив.
                    // Его индексом (ключом)
                    // будет 3, а не 0
    print_r($arr);

    $arr =
     array_values($arr); // переиндексируем
                         // массив
    $arr[] = "bb"; // ключом этого элемента
                   // будет 1
    print_r($arr);
    ?>
```

Результатом работы этого скрипта будет:
Array ([0] => a [1] => b [2] => c)
Array ( )
Array ([3] => aa)
Array ([0] => aa [1] => bb)

2.4.6 Тип object (объекты)

Объекты – тип данных, пришедший из объектно-ориентированного программирования (ООП). Согласно принципам ООП, класс – это набор объектов, обладающих определенными свойствами и методами работы с ним, а объект соответственно – экземпляр класса. Например, программисты – это класс людей, которые пишут программы, изучают компьютерную литературу и, кроме того, как все люди, имеют имя и фамилию. Теперь, если взять одного конкретного программиста, Васю Иванова, то можно сказать, что он является объектом класса программистов, обладает теми же свойствами, что и другие программисты, тоже имеет имя, пишет программы и т.п. В PHP для доступа к методам объекта используется оператор ->. Для инициализации объекта используется выражение new, создающее в переменной экземпляр объекта.

```
<?php
//создаем класс людей
class Person
{
// метод, который обучает человека PHP
    function know_php()
        {
            echo "Теперь я знаю PHP";
        }
}
$bob = new Person; // создаем объект
                   // класса человек
$bob -> know_php(); // обучаем его PHP
?>
```

2.4.7 Тип resource (ресурсы)

.Ресурс – это специальная переменная, содержащая ссылку на внешний ресурс (например, соединение с базой данных). Ресурсы создаются и используются специальными функциями (например, mysql_connect(), pdf_new() и т.п.).

2.4.8 Тип Null

Специальное значение NULL говорит о том, что переменная не имеет значения.

Переменная считается NULL, если:

- ей была присвоена константа NULL ($var = NULL);
- ей еще не было присвоено какое-либо значение;
- она была удалена с помощью unset().

Существует только одно значение типа NULL – регистронезависимое ключевое слово NULL.

2.5 Операторы условий и циклов

2.5.1 Условные операторы

Оператор if

Это один из самых важных операторов многих языков, включая PHP. Он позволяет выполнять фрагменты кода в зависимости от условия. Структуру оператора if можно представить следующим образом:

if (выражение) блок_выполнения

Здесь выражение есть любое правильное PHP-выражение (т.е. все, что имеет значение). В процессе обработки скрипта выражение преобразуется к логическому типу. Если в результате преобразования

значение выражения истинно (True), то выполняется блок_выполнения. В противном случае блок_выполнения игнорируется. Если блок_выполнения содержит несколько команд, то он должен быть заключен в фигурные скобки { }.

Правила преобразования выражения к логическому типу:

1. В FALSE преобразуются следующие значения:
   - логическое False
   - целый ноль (0)
   - действительный ноль (0.0)
   - пустая строка и строка "0"
   - массив без элементов
   - объект без переменных (подробно об объектах будет рассказано в одной из следующих лекций)
   - специальный тип NULL
2. Все остальные значения преобразуются в TRUE.

```
<?
$names = array("Иван","Петр","Семен");
if ($names[0]=="Иван") {
    echo "Привет, Ваня!";
    $num = 1;
    $account = 2000;
}
if ($num) echo "Иван первый в списке!";
$bax = 30;
if ($account > 100*$bax+3)
    echo "Эта строчка не появится
    на экране, так как условие не выполнено";
?>
```

Оператор else

Мы рассмотрели только одну, основную часть оператора if. Существует несколько расширений этого оператора. Оператор else расширяет if на случай, если проверяемое в if выражение является неверным, и позволяет выполнить какие-либо действия при таких условиях.

Структуру оператора if, расширенного с помощью оператора else, можно представить следующим образом:

if (выражение) блок_выполнения
else блок_выполнения1

Эту конструкцию if...else можно интерпретировать примерно так: если выполнено условие (т.е. выражение=true), то выполняем действия из блока_выполнения, иначе – действия из блока_выполнения1. Использовать оператор else не обязательно.

Посмотрим, как можно изменить предыдущий пример, учитывая необходимость совершения действий и в случае невыполнения условия.

```
<?
$names = array("Иван","Петр","Семен");
if ($names[0]=="Иван") {
    echo "Привет, Ваня!";
    $num = 1;
    $account = 2000;
} else {
    echo "Привет, $names[0].
    А мы ждали Ваню :(";
}
if ($num) echo "Иван первый в списке!";
else echo "Иван НЕ первый в списке?!";
$bax = 30;
if ($account > 100*$bax+3)
    echo "Эта строка не появится на экране,
    так как условие не выполнено";
    else echo "Зато появится эта строка!";
?>
```

Оператор elseif

Еще один способ расширения условного оператора if – использование оператора elseif. elseif – это комбинация else и if. Как и else, он расширяет if для выполнения различных действий в том случае, если условие, проверяемое в if, неверно. Но в отличие от else, альтернативные действия будут выполнены, только если elseif-

условие является верным. Структуру оператора if, расширенного с помощью операторов else и elseif, можно представить следующим образом:

if (выражение) блок_выполнения
elseif(выражение1) блок_выполнения1
...
else блок_выполненияN

Операторов elseif может быть сразу несколько в одном if-блоке. Elseif-утверждение будет выполнено, только если предшествующее if-условие является False, все предшествующие elseif-условия являются False, а данное elseif-условие – True.

```
<?
$names = array("Иван","Петр","Семен");
if ($names[1]=="Иван") {
    // если первое имя в массиве Иван
    echo "Привет, Ваня!";
}elseif ($names[1] == "Петр"){
    // если первое имя
    // не Иван, а Петр
    echo "Привет, Петя!";
}elseif ($names[0] == "Семен"){
    // если первое имя не
    // Иван, не Петр, а Семен
    echo "Привет, Сеня!";
}else {
    // если первое имя не Иван,
    // не Петр и не Семен
    echo "Привет, $names[0]. А ты кто такой?";
}
?>
```

Альтернативный синтаксис

PHP предлагает альтернативный синтаксис для некоторых своих управляющих структур, а именно для if, while, for, foreach и switch. В каждом случае открывающую скобку нужно заменить на двоеточие (:), а закрывающую – на endif;, endwhile; и т.д. соответственно.

Например, синтаксис оператора if можно записать таким образом:

if(выражение): блок_выполнения endif;

Смысл остается тем же: если условие, записанное в круглых скобках оператора if, оказалось истиной, будет выполняться весь код, от двоеточия «:» до команды endif;. Использование такого синтаксиса полезно при встраивании php в html-код.

```
<?php
$names = array("Иван","Петр","Семен");
if ($names[0]=="Иван"):
?>
Привет, Ваня!
<?php
endif; ?>
```

Если используются конструкции else и elseif, то также можно задействовать альтернативный синтаксис:

```
<?php
$a=1;
if ($a == 5):
    print "a равно 5";
    print "...";
elseif ($a == 6):
    print "a равно 6";
    print "!!!";
else:
    print "a не равно ни 5, ни 6";
endif;
?>
```

Оператор switch

Еще одна конструкция, позволяющая проверять условие и выполнять в зависимости от этого различные действия, – это switch. На русский язык название данного оператора можно перевести как «переключатель». И смысл у него именно такой. В зависимости от

того, какое значение имеет переменная, он переключается между различными блоками действия. switch очень похож на оператор if...elseif...else или набор операторов if. Структуру switch можно записать следующим образом:

```
switch (выражение или переменная){
    case значение1:
       блок_действий1
    break;
    case значение2:
       блок_действий2
    break;
    ...
    default:
       блок_действий_по_умолчанию
}
```

В отличие от if, здесь значение выражения не приводится к логическому типу, а просто сравнивается со значениями, перечисленными после ключевых слов case (значение1, значение2 и т.д.). Если значение выражения совпало с каким-то вариантом, то выполняется соответствующий блок_действий – от двоеточия после совпавшего значения до конца switch или до первого оператора break, если таковой найдется. Если значение выражения не совпало ни с одним из вариантов, то выполняются действия по умолчанию (блок_действий_по_умолчанию), находящиеся после ключевого слова default. Выражение в switch вычисляется только один раз, а в операторе elseif – каждый раз, поэтому, если выражение достаточно сложное, то switch работает быстрее.

```php
<?
$names = array("Иван","Петр","Семен");
switch ($names[0]){
case "Иван":
    echo "Привет, Ваня!";
break;
```

```
    case "Петр":
        echo "Привет, Петя!";
    break;
    case "Семен":
        echo "Привет, Сеня!";
    break;
    default:
        echo "Привет, $names[0].
        А как Вас зовут?";
    }
    ?>
```

Если в этом примере опустить оператор break, например, в case "Петр":, то, если переменная окажется равной строке "Петр", после вывода на экран сообщения "Привет, Петя!" программа пойдет дальше и выведет также сообщение "Привет, Сеня!" и только потом, встретив break, продолжит свое выполнение за пределами switch.

Для конструкции switch, как и для if, возможен альтернативный синтаксис, где открывающая switch фигурная скобка заменяется двоеточием, а закрывающая – endswitch; соответственно.

2.5.2 Циклы

В PHP существует несколько конструкций, позволяющих выполнять повторяющиеся действия в зависимости от условия. Это циклы while, do..while, foreach и for. Рассмотрим их более подробно.

Цикл while

Структура:
while (выражение) { блок_выполнения }
либо
while (выражение): блок_выполнения endwhile;

Цикл while предписывает PHP выполнять команды блока_выполнения до тех пор, пока выражение вычисляется как True (здесь, как и в if, происходит приведение выражения к логическому

типу). Значение выражения проверяется каждый раз в начале цикла, так что, даже если его значение изменилось в процессе выполнения блока_выполнения, цикл не будет остановлен до конца итерации (т.е. пока все команды блока_выполнения не будут исполнены).

```
<?
//эта программа напечатает все четные цифры
    $i = 1;
    while ($i < 10) {
        if ($i % 2 == 0) print $i;
    // печатаем цифру, если она четная
        $i++;
    // и увеличиваем $i на единицу
    }
?>
```

Цикл do... while

Циклы do..while очень похожи на циклы while, с той лишь разницей, что истинность выражения проверяется в конце цикла, а не в начале. Благодаря этому блок_выполнения цикла do...while гарантированно выполняется хотя бы один раз.

Структура:

do {блок_выполнения} while (выражение);

```
<?
// эта программа напечатает число 12, несмотря на то
// что условие цикла не выполнено
$i = 12;
    do {
        if ($i % 2 == 0) print $i;
    // если число четное, то печатаем его
        $i++;
    // увеличиваем число на единицу
    } while ($i<10)
?>
```

Цикл for

Это самые сложные циклы в PHP. Они напоминают соответствующие циклы C.

Структура:

for (выражение1; выражение2; выражение3) {блок_выполнения}

либо

for (выражение1; выражение2; выражение3): блок_выполнения endfor;

Здесь, как мы видим, условие состоит сразу из трех выражений. Первое выражение выражение1 вычисляется безусловно один раз в начале цикла. В начале каждой итерации вычисляется выражение2. Если оно является True, то цикл продолжается и выполняются все команды блока_выполнения. Если выражение2 вычисляется как False, то исполнение цикла останавливается. В конце каждой итерации (т.е. после выполнения всех команд блока_выполнения) вычисляется выражение3.

Каждое из выражений 1, 2, 3 может быть пустым. Если выражение2 является пустым, то это значит, что цикл должен выполняться неопределенное время (в этом случае PHP считает это выражение всегда истинным). Это не так бесполезно, как кажется, ведь цикл можно останавливать, используя оператор break.

Например, все четные цифры можно вывести с использованием цикла for таким образом:

```
<?php
for ($i=0; $i<10; $i++){
    if ($i % 2 == 0) print $i;
    // печатаем четные числа
}
?>
```

Если опустить второе выражение (условие $i<10), то такую же задачу можно решить, останавливая цикл оператором break.

```
<?php
for ($i=0; ; $i++){
```

```
        if ($i>=10) break;
          // если $i больше или равно 10,
          // то прекращаем работу цикла
        if ($i % 2 == 0) print $i;
          // если число четное,
          // то печатаем его
    }
    ?>
```

Можно опустить все три выражения. В этом случае просто не будет задано начальное значение счетчика $i и оно не будет изменяться каждый раз в конце цикла. Все эти действия можно записать в виде отдельных команд либо в блоке_выполнения, либо перед циклом:

```
    <?php
    $i=2; // задаем начальное значение счетчика
    for ( ; ; ){
        if ($i>=10) break;
          // если $i больше или равно 10,
          // то прекращаем работу цикла
        if ($i % 2 == 0) print $i;
          // если число четное,
          // то печатаем его
    $i++; // увеличиваем счетчик на единицу
    }
    ?>
```

В третье выражение конструкции for можно записывать через запятую сразу несколько простейших команд. Например, если мы хотим просто вывести все цифры, то программу можно записать совсем просто:

```
    <?php
    for ($i=0; $i<10; print $i, $i++)
    /* Если блок_выполнения не содержит команд
    или содержит только одну команду,
    фигурные скобки, в которые он заключен,
    можно опускать*/
    ?>
```

Цикл foreach

Еще одна полезная конструкция. Она появилась в PHP4 и предназначена исключительно для работы с массивами.

Синтаксис:

foreach ($array as $value) {блок_выполнения}
либо
foreach ($array as $key => $value)
  {блок_выполнения}

В первом случае формируется цикл по всем элементам массива, заданного переменной $array. На каждом шаге цикла значение текущего элемента массива записывается в переменную $value, и внутренний счетчик массива передвигается на единицу (так что на следующем шаге будет виден следующий элемент массива). Внутри блока_выполнения значение текущего элемента массива может быть получено с помощью переменной $value. Выполнение блока_выполнения происходит столько раз, сколько элементов в массиве $array.

Вторая форма записи в дополнение к перечисленному выше на каждом шаге цикла записывает ключ текущего элемента массива в переменную $key, которую тоже можно использовать в блоке_выполнения.

Когда foreach начинает исполнение, внутренний указатель массива автоматически устанавливается на первый элемент.

```
<?php
$names = array("Иван","Петр","Семен");
foreach ($names as $val) {
    echo "Привет, $val <br>";
    // выведет всем приветствие
}
foreach ($names as $k => $val) {
    // кроме приветствия,
    // выведем номера в списке, т.е. ключи
    echo "Привет, $val !
      Ты в списке под номером $k <br>";
}
```

```
?>
```

2.5.3 Операторы передачи управления

Иногда требуется немедленно завершить работу цикла либо отдельной его итерации. Для этого используют операторы break и continue.

Break

Оператор break заканчивает выполнение текущего цикла, будь то for, foreach, while, do..while или switch. break может использоваться с числовым аргументом, который говорит, работу скольких управляющих структур, содержащих его, нужно завершить.

```
<?php
$i=1;
while ($i) {
    $n = rand(1,10);
    // генерируем произвольное число
    // от 1 до 10
    echo "$i:$n ";
    // выводим номер итерации и
    // сгенерированное число
    if ($n==5) break;
/* Если было сгенерировано число 5,
то прекращаем работу цикла. В этом случае
все, что находится после этой строчки
внутри цикла, не будет выполнено */
    echo "Цикл работает <br>";
    $i++;
}
echo "<br>Число итераций цикла $i ";
?>
```

Результатом работы этого скрипта будет примерно следующее:

1:7 Цикл работает

2:2 Цикл работает

3:5

Число итераций цикла 3

Если после оператора break указать число, то прервется именно такое количество содержащих этот оператор циклов. В приведенном выше примере это неактуально, поскольку вложенных циклов нет. Немного изменим наш скрипт:

```
<?php
$i=1;
while ($i) {
  $n = rand(1,10);
  // генерируем произвольное число
  // от 1 до 10
  switch ($n){
    case 5:
      echo "<font color=blue>
        Выход из switch (n=$n)</font>";
    break 1;
    // прекращаем работу switch
    // (первого содержащего break цикла)
    case 10:
      echo "<font color=red>
        Выход из switch и
        while (n=$n)</font>";
    break 2;
    // прекращаем работу switch и while
    // (двух содержащих break циклов)
    default:
      echo "switch работает (n=$n), ";
}
echo " while работает - шаг $i <br>";
$i++;
}
echo "<br>Число итераций цикла $i ";
?>
```

Оператор continue

Иногда нужно не полностью прекратить работу цикла, а только начать его новую итерацию. Оператор continue позволяет пропустить дальнейшие инструкции из блока_выполнения любого цикла и продолжить выполнение с нового круга. continue можно использовать с числовым аргументом, который указывает, сколько содержащих его управляющих конструкций должны завершить работу. Заменим в

примере предыдущего параграфа оператор break на continue. Кроме того, ограничим число шагов цикла тремя.

```
<?php
$i=1;
while ($i<=4) {
    $n = rand(1,10);
    // генерируем произвольное число
    // от 1 до 10
    echo "$i:$n ";
    // выводим номер итерации и
    // сгенерированное число
    if ($n==5) {
        echo "Новая итерация <br>";
        continue;
/* Если было сгенерировано число 5,
то начинаем новую итерацию цикла,
$i не увеличивается */
    }
    echo "Цикл работает <br>";
    $i++;
}
--$i;
echo "<br>Число итераций цикла $i ";
?>
```

Результатом работы этого скрипта будет

1:10 Цикл работает
2:5 Новая итерация
2:1 Цикл работает
3:1 Цикл работает
Число итераций цикла 4

Заметим, что после выполнения оператора continue работа цикла не заканчивается. В примере счетчик цикла не меняется в случае получения числа 5, поскольку он находится после оператора continue. Фактически с помощью continue мы пытаемся избежать ситуации, когда будет сгенерировано число 5. Поэтому можно было просто написать, заменив оператор continue на проверку истинности выражения:

```php
<?php
$i=1;
while ($i<4) {
    $n = rand(1,10);
    // генерируем произвольное число
    // от 1 до 10
    if ($n!==5) {
        echo "$i:$n <br>";
    // выводим номер итерации
    // и сгенерированное число
    $i++;
    }
}
?>
```

В PHP существует одна особенность использования оператора continue – в конструкциях switch он работает так же, как и break. Если switch находится внутри цикла и нужно начать новую итерацию цикла, следует использовать continue 2.

2.6 Операторы включения

Оператор include

Оператор include позволяет включать код, содержащийся в указанном файле, и выполнять его столько раз, сколько программа встречает этот оператор. Включение может производиться любым из перечисленных способов:

include 'имя_файла';
include $file_name;
include ("имя_файла");

Допустим, что существует следующий файл params.inc.

```php
<?php
$user = "Вася";
$today = date("d.m.y");
/* функция date() возвращает дату
и время (здесь – дату в формате
```

```
    день.месяц.год) */
    ?>
```

Для включения содержимого этого файла используется оператор include.

```
    <?php
    include ("params.inc");
    /* переменные $user и $today заданы в файле
    params.inc. Здесь мы тоже можем ими
    пользоваться благодаря команде
    include("params.inc") */

    echo "Привет, $user!<br>";
        // выведет "Привет, Вася!"
    echo "Сегодня $today";
        // выведет, например, "Сегодня 7.07.05"
    ?>
```

Заметим, что использование оператора include эквивалентно простой вставке содержательной части файла params.inc в код программы include.php. Может быть, тогда можно было в params.inc записать простой текст без всяких тегов, указывающих на то, что это php-код? Нельзя! Дело в том, что в момент вставки файла происходит переключение из режима обработки PHP в режим HTML. Поэтому код внутри включаемого файла, который нужно обработать как PHP-скрипт, должен быть заключен в соответствующие теги.

Поиск файла для вставки происходит по следующим правилам.

1. Сначала ведется поиск файла в include_path относительно текущей рабочей директории.
2. Если файл не найден, то поиск производится в include_path относительно директории текущего скрипта.
3. Параметр include_path, определяемый в файле настроек PHP, задает имена директорий, в которых нужно искать включаемые файлы.

Если файл включен с помощью include, то содержащийся в нем код наследует область видимости переменных строки, где появился include. Любые переменные вызванного файла будут доступны в

вызывающем файле с этой строки и далее. Соответственно, если include появляется внутри функции вызывающего файла, то код, содержащийся в вызываемом файле, будет вести себя так, как будто он был определен внутри функции. Таким образом, он унаследует область видимости этой функции. Хотя мы и не знакомились еще с понятием функции, все же приводим здесь эти сведения в расчете на интуитивное его понимание.

Кроме локальных файлов, с помощью include можно включать и внешние файлы, указывая их url-адреса. Данная возможность контролируется директивой url_fopen_wrappers в файле настроек PHP и по умолчанию, как правило, включена. Но в версиях PHP для Windows до PHP 4.3.0 эта возможность не поддерживается совсем, вне зависимости от url_fopen_wrappers.

Оператор include – это специальная языковая конструкция, поэтому при использовании внутри условных блоков ее нужно заключать в фигурные скобки.

```
<?php
/*  Это неверная запись. Получим ошибку.
    Мы же вставляем не одну команду,
    а несколько, они только записаны
    в другом файле */
if ($condition) include("first.php");
else include("second.php");
// А вот так правильно.
if ($condition){ include("first.php"); }
else { include("second.php"); }
?>
```

При использовании include возможны два вида ошибок – ошибка вставки (например, нельзя найти указанный файл, неверно написана сама команда вставки и т.п.) или ошибка исполнения (если ошибка содержится во вставляемом файле). В любом случае при ошибке в команде include исполнение скрипта не завершается.

Оператор require

Основное отличие require и include заключается в том, как они реагируют на возникновение ошибки. Как уже говорилось, include выдает предупреждение, и работа скрипта продолжается. Ошибка в require вызывает фатальную ошибку работы скрипта и прекращает его выполнение.

Условные операторы на require() не влияют. Хотя, если строка, в которой появляется этот оператор, не исполняется, то ни одна строка кода из вставляемого файла тоже не исполняется. Циклы также не влияют на require(). Хотя код, содержащийся во вставляемом файле, является объектом цикла, но вставка сама по себе происходит только однажды.

В реализациях PHP до версии 4.0.2 использование require() означало, что интерпретатор обязательно попытается прочесть вставляемый файл.

Оператор require, как и include, при использовании внутри условных блоков нужно заключать в фигурные скобки.

Использование операторов include и require имеют недостатки. Так, мы можем в разных местах кода неумышленно подключить один и тот же файл, что при выполнении кода вызовет ошибки. Чтобы исключить повторное подключение файла, вместо инструкции include надо применять инструкцию include_once, а вместо require – require_once.

2.7 Функции

Функции представляют собой набор инструкций, выполняющих определенное действие.

Функция может быть определена с помощью следующего синтаксиса:

```
function Имя_функции (параметр1, параметр2, ... параметрN){
  Блок_действий
  return "значение, возвращаемое функцией";
}
```

Для выполнения набора инструкций, описанных в теле функции, функцию необходимо вызвать, указав имя функции и в круглых скобках список значений ее параметров, если таковые имеются:

```
<?php
Имя_функции ("значение_для_параметра1",
  "значение_для_параметра2",...);
// пример вызова функции – вызов функции
// вычисления факториала приведен выше,
// там для вычисления факториала числа 3
// мы писали: fact(3);
// где fact – имя вызываемой функции,
// а 3 – значение ее параметра с именем $n
?>
```

В ранних версиях PHP функцию можно было вызвать только после ее определения, т.е. в любой строке программы ниже блока function f_name(){...}. В современных версиях PHP такого требования нет. Все дело в том, как интерпретатор обрабатывает получаемый код. Единственное исключение составляют функции, определяемые условно (внутри условных операторов или других функций). Когда функция определяется таким образом, ее определение должно предшествовать ее вызову.

```
<?php
$make = true;
/* здесь нельзя вызвать Make_event();
потому что она еще не существует, но можно
вызвать Save_info() */

Save_info("Вася","Иванов",
   "Я выбрал курс по PHP");

if ($make){
// определение функции Make_event()
  function Make_event(){
    echo "<p>Хочу изучать Python<br>";
  }
}
// теперь можно вызывать Make_event()
Make_event();
// определение функции Save_info
```

```
function Save_info($first, $last, $message){
  echo "<br>$message<br>";
  echo "Имя: ". $first . " ". $last . "<br>";
}
Save_info("Федя","Федоров",
          "А я выбрал Lisp");
// Save_info можно вызывать и здесь
?>
```

Если функция однажды определена в программе, то переопределить или удалить ее позже нельзя. Несмотря на то, что имена функций нечувствительны к регистру, лучше вызывать функцию по тому же имени, каким она была задана в определении.

2.7.1 Аргументы функций

У каждой функции может быть, как мы уже говорили, список аргументов. С помощью этих аргументов в функцию передается различная информация. Каждый аргумент представляет собой переменную или константу.

С помощью аргументов данные в функцию можно передавать тремя различными способами. Это передача аргументов по значению (используется по умолчанию), по ссылке и задание значения аргументов по умолчанию. Рассмотрим эти способы подробнее.

Когда аргумент передается в функцию по значению, изменение значения аргумента внутри функции не влияет на его значение вне функции. Чтобы позволить функции изменять ее аргументы, их нужно передавать по ссылке. Для этого в определении функции перед именем аргумента следует написать знак амперсанд "&".

```
<?php
// напишем функцию, которая бы добавляла к строке слово
checked
function add_label(&$data_str){
  $data_str .= "checked";
}
$str = "<input type=radio name=article "; // пусть
имеется такая строка
echo $str .="><br>";
```

```
                        // выведет элемент формы -
                              // не отмеченную радио кнопку
    add_label($str);  // вызовем функцию
    echo $str ."><br>";
        // это выведет уже отмеченную
        // радио кнопку
    ?>
```

В функции можно определять значения аргументов, используемые по умолчанию. Само значение по умолчанию должно быть константным выражением, а не переменной и не представителем класса или вызовом другой функции.

Допустим у нас есть функция, создающая информационное сообщение, подпись к которому меняется в зависимости от значения переданного ей параметра. Если значение параметра не задано, то используется подпись "Оргкомитет.".

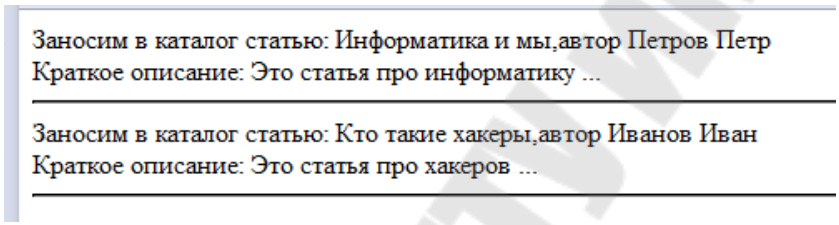
```
    <?php
    function Message($sign="Оргкомитет."){
    // здесь параметр sign имеет по умолчанию значение
"Оргкомитет"
      echo "Следующее собрание состоится завтра.<br>";
      echo $sign . "<br>";
    }
    Message();
        // вызываем функцию без параметра.
        // В этом случае подпись - это Оргкомитет
    Message("С уважением, Вася.");
        // В этом случае подпись
        // будет "С уважением, Вася."
    ?>
```

Если у функции несколько параметров, то те аргументы, для которых задаются значения по умолчанию, должны быть записаны после всех остальных аргументов в определении функции. В противном случае появится ошибка, если эти аргументы будут опущены при вызове функции.

Например, мы хотим внести описание статьи в каталог. Пользователь должен ввести такие характеристики статьи, как ее название, автор и краткое описание. Если пользователь не вводит имя автора статьи, считаем, что это Иванов Иван.

```php
<?php
function Add_article($title, $description,
    $author="Иванов Иван"){
  echo "Заносим в каталог статью: $title,";
  echo "автор $author";
  echo "<br>Краткое описание: ";
  echo "$description <hr>";
}
Add_article("Информатика и мы", "Это статья про
информатику ...",
    "Петров Петр");
Add_article("Кто такие хакеры", "Это статья про хакеров
...");
?>
```

В результате работы скрипта получим следующее:



Если написать вот так:

```php
<?php
function Add_article($author="Иванов Иван",
    $title, $description){
// ...действия как в предыдущем примере
}
Add_article("Кто такие хакеры",
    "Это статья про хакеров...");
?>
```

Будет выдано сообщение об ошибке.

2.7.2 Списки аргументов переменной длины

Начингая с PHP4 можно создавать функции с переменным числом аргументов. То есть мы создаем функцию, не зная заранее, со

сколькими аргументами ее вызовут. Для написания такой функции никакого специального синтаксиса не требуется. Все делается с помощью встроенных функций func_num_args(), func_get_arg(), func_get_args().

Функция func_num_args() возвращает число аргументов, переданных в текущую функцию. Эта функция может использоваться только внутри определения пользовательской функции. Если она появится вне функции, то интерпретатор выдаст предупреждение.

```
<?php
function DataCheck(){
  $n = func_num_args();
  echo "Число аргументов функции $n";
}
DataCheck();
   // выведет строку
   // "Число аргументов функции 0"
DataCheck(1,2,3);
   // выведет строку
   // "Число аргументов функции 3"
?>
```

Функция func_get_arg (целое номер_аргумента) возвращает аргумент из списка переданных в функцию аргументов, порядковый номер которого задан параметром номер_аргумента. Аргументы функции считаются, начиная с нуля. Как и func_num_args(), эта функция может использоваться только внутри определения какой-либо функции.

Номер_аргумента не может превышать число аргументов, переданных в функцию. Иначе будет сгенерировано предупреждение, и функция func_get_arg() возвратит False.

Создадим функцию для проверки типа данных ее аргументов. Считаем, что проверка прошла успешно, если первый аргумент функции – целое число, второй – строка.

```
<?
function DataCheck(){
  $check =true;
  $n = func_num_args();
```

```
        // число аргументов,
        // переданных в функцию
    /* проверяем, является ли первый
    переданный аргумент целым числом */
    if ($n>=1) if (!is_int(func_get_arg(0)))
        $check = false;
    /* проверяем, является ли второй
    переданный аргумент строкой */
    if ($n>=2)
       if (!is_string(func_get_arg(1)))
          $check = false;
return $check;
}

if (DataCheck(123,"text"))
   echo "Проверка прошла успешно<br>";
else echo "Данные не удовлетворяют
   условиям<br>";
if (DataCheck(324))
   echo "Проверка прошла успешно<br>";
else echo "Данные не удовлетворяют условиям<br>";
?>
```

Функция func_get_args() возвращает массив, состоящий из списка аргументов, переданных функции. Каждый элемент массива соответствует аргументу, переданному функции. Если функция используется вне определения пользовательской функции, то генерируется предупреждение.

Перепишем предыдущий пример, используя эту функцию. Будем проверять, является ли целым числом каждый четный аргумент, передаваемый функции:

```
<?
function DataCheck(){
  $check =true;
  $n = func_num_args();
  // число аргументов,
  // переданных в функцию

  $args = func_get_args();
  // массив аргументов функции
  for ($i=0;$i<$n;$i++){
    $v = $args[$i];
    if ($i % 2 == 0){
      if (!is_int($v)) $check = false;
```

```
            // проверяем,
            // является ли четный аргумент целым
        }
    }
return $check;
}
if (DataCheck(array("text", 324)))
   echo "Проверка прошла успешно<br>";
else echo "Данные не удовлетворяют
   условиям<br>";
?>
```

Как видим, комбинации функций func_num_args(), func_get_arg() и func_get_args() используется для того, чтобы функции могли иметь переменный список аргументов.

2.7.3 Использование переменных внутри функции

Глобальные переменные

Переменные, определенные внутри функции, имеют локальную область видимости. Чтобы использовать внутри функции переменные, заданные вне ее, эти переменные нужно объявить как глобальные. Для этого в теле функции следует перечислить их имена после ключевого слова global:

```
<?
global $var1, $var2;
$a=1;
function Test_g(){
global $a;
  $a = $a*2;
  echo 'в результате работы функции $a=',$a;
}
echo 'вне функции $a=',$a,', ';
Test_g();
echo "<br>";
echo 'вне функции $a=',$a,', ';
Test_g();
?>
```

Когда переменная объявляется как глобальная, фактически создается ссылка на глобальную переменную. Поэтому такая запись эквивалентна следующей (массив $GLOBALS содержит все переменные, глобальные относительно текущей области видимости):

```
$var1 = &$GLOBALS["var1"];
$var2 = &$GLOBALS["var2"];
```

Это значит, например, что удаление переменной $var1 не удаляет глобальной переменной $GLOBALS["var1"].

Статические переменные

Чтобы использовать переменные только внутри функции, при этом сохраняя их значения и после выхода из функции, нужно объявить эти переменные как статические. Статические переменные видны только внутри функции и не теряют своего значения, если выполнение программы выходит за пределы функции. Объявление таких переменных производится с помощью ключевого слова static:

```
static $var1, $var2;
```

Статической переменной может быть присвоено любое значение, но не ссылка.

```
<?
function Test_s(){
static $a = 1;
// нельзя присваивать выражение или ссылку
  $a = $a*2;
  echo $a;
}
Test_s(); // выведет 2
echo $a;  // ничего не выведет, так как
          // $a доступна только
          // внутри функции
Test_s(); // внутри функции $a=2, поэтому
          // результатом работы функции
          // будет число 4
?>
```

2.7.4 Возвращаемые значения

Все функции, приведенные выше в качестве примеров, выполняли какие-либо действия. Кроме подобных действий, любая функция может возвращать как результат своей работы какое-нибудь значение. Это делается с помощью оператора return. Возвращаемое значение может быть любого типа, включая списки и объекты. Когда интерпретатор встречает команду return в теле функции, он немедленно прекращает ее исполнение и переходит на ту строку, из которой была вызвана функция.

Например, составим функцию, которая возвращает возраст человека. Если человек не умер, то возраст считается относительно текущего года.

```
<?php
/* если второй параметр вычисляется
как true, то он рассматривается как
дата смерти, */

function Age($birth, $is_dead){
  if ($is_dead) return $is_dead-$birth;
  else return date("Y")-$birth;
}
echo Age(1971, false); // для 2014 года выведет 43
echo Age(1971, 2001); // выведет 30
?>
```

В этом примере можно было и не использовать функцию return, а просто заменить ее функцией вывода echo. Однако если мы все же делаем так, что функция возвращает какое-то значение (в данном случае возраст человека), то в программе мы можем присвоить любой переменной значение этой функции:

$an_age = Age(1981, 2004);

В результате работы функции может быть возвращено только одно значение. Несколько значений можно получить, если возвращать список значений (одномерный массив). Допустим, мы хотим получить полный возраст человека с точностью до дня.

```
<?php
function Full_age($b_day, $b_month, $b_year)
{
        $y = date("Y");
        $m = intval(date("m"));
        $d = intval(date("d"));
        $b_month = intval($b_month);
        $b_day = intval($b_day);
        $b_year = intval($b_year);

        $day = ($b_day > $d ? 30 - $b_day + $d : $d - $b_day);
        $tmpMonth = ($b_day > $d ?  -1 : 0);
        $month = ($b_month > $m + $tmpMonth ? 12 - $b_month +
          $tmpMonth + $m : $m+$tmpMonth - $b_month);
        $tmpYear  = ($b_month >  $m + $tmpMonth ?  -1 : 0);
        if ($b_year > $y + $tmpYear)
        {
                $year = 0; $month = 0; $day = 0;
        }
        else
        {
                $year = $y + $tmpYear - $b_year;
        }
        return array ($day,$month,$year);
}
$age = Full_age("29","06","1986");
echo "Вам $age[2] лет, $age[1] месяцев и $age[0] дней";
?>
```

Когда функция возвращает несколько значений для их обработки в программе, удобно использовать языковую конструкцию list (), которая позволяет одним действием присвоить значения сразу нескольким переменным. Например, в предыдущем примере, оставив без изменения функцию, обработать возвращаемые ей значения можно было так:

```
<?
// задание функции Full_age()
list($day,$month,$year) = Full_age("07",
```

```
        "08","1974");
echo "Вам $year лет, $month месяцев и
    $day дней";
?>
```

2.7.5 Переменные функции

PHP поддерживает концепцию переменных функций. Это значит, что если имя переменной заканчивается круглыми скобками, то PHP ищет функцию с таким же именем и пытается ее выполнить.

```
<?
/* создадим две простые функции:
Add_sign - добавляет подпись к строке и
Show_text - выводит строку текста */

function Add_sign($string,
   $sign="С уважением, Петр"){
   echo $string ." ".$sign;
}
function Show_text(){
  echo "Отправить сообщение по почте<br>";
}
$func = "Show_text";
   // создаем переменную со значением,
   // равным имени функции Show_text
$func();
   // это вызовет функцию Show_text
$func = "Add_sign";
   // создаем переменную со значением,
   // равным имени функции Add_sign
$func("Привет всем <br>");
   // это вызовет функцию
   // Add_sign с параметром "Привет всем"
?>
```

В этом примере функция Show_text просто выводит строку текста. Казалось бы, зачем для этого создавать отдельную функцию, если существует специальная функция echo(). Дело в том, что такие функции, как echo(), print(), unset(), include() и т.п. нельзя использовать в качестве переменных функций. То есть если мы напишем:

```php
<?
$func = "echo";
$func("TEXT");
?>
```

то интерпретатор выведет ошибку:

Fatal error: Call to undefined function:
echo() in
c:\users\tasks\func\var_f.php on line 2

Поэтому для того, чтобы использовать любую из перечисленных выше функций как переменную функцию, нужно создать собственную функцию, что мы и сделали в предыдущем примере.

2.8 Массивы

Массивы предназначены для хранения наборов данных или элементов. Каждый элемент в массиве имеет свой уникальный ключ и значение.

Массив можно создать двумя способами:

1. С помощью конструкции array
   $array_name = array("key1"=>"value1",
   "key2"=>"value2");
2. Непосредственно задавая значения элементам массива
   $array_name["key1"] = value1;

Например, нам нужно хранить список документов, которые будут удалены из базы данных. Естественно хранить его будем в виде массива, ключом в котором будет идентификатор документа (его уникальный номер), а значением – название документа. Этот массив можно создать таким образом:

```php
<?
$del_items = array("10"=>"Наука и жизнь",
   "12"=>"Информатика");
$del_items["13"] = "Программирование на PHP";
    // добавляем элемент в массив
```

```
?>
```

Операции с массивами

Массив – это тип данных, с данными этого типа должны быть определены операции. Какие же операции можно производить с массивами?

Массивы можно складывать и сравнивать.

Складывают массивы с помощью стандартного оператора «+». Вообще говоря, эту операцию по отношению к массивам точнее назвать объединением. Если у нас есть два массива, $a и $b, то результатом их сложения (объединения) будет массив $c, состоящий из элементов $a, к которым справа дописаны элементы массива $b. Причем, если встречаются совпадающие ключи, то в результирующий массив включается элемент из первого массива, т.е. из $a. Таким образом, если складываются массивы в языке PHP, от перемены мест слагаемых сумма меняется.

```
<?
$a = array("и"=>"Информатика",
           "м"=>"Математика");
$b = array("и"=>"История","м"=>"Биология",
           "ф"=>"Физика");
$c = $a + $b;
$d = $b + $a;
print_r($c);
/* получим: Array([и]=>Информатика
   [м]=>Математика [ф]=>Физика) */
print_r($d);
/* получим: Array([и]=>История
   [м]=>Биология [ф]=>Физика) */
?>
```

Сравнивать массивы можно, проверяя их равенство или неравенство либо эквивалентность или неэквивалентность. Равенство массивов – это когда совпадают все пары ключ/значение элементов массивов. Эквивалентность – когда кроме равенства значений и ключей элементов требуется еще, чтобы элементы в обоих массивах были записаны в одном и том же порядке. Равенство значений в PHP обозначается символом «==», а эквивалентность – символом «===».

```
<?
$a = array("и"=>"Информатика",
           "м"=>"Математика");
$b = array("м"=>"Математика",
           "и"=>"Информатика");
if ($a == $b) echo "Массивы равны и";
  else echo "Массивы НЕ равны и ";
if ($a === $b) echo " эквивалентны";
  else echo " НЕ эквивалентны";
// получим echo "Массивы равны и
//               НЕ эквивалентны"
?>
```

Далее рассмотрим еще одну важную операцию с массивом – подсчет количества его элементов. Для ее реализации в PHP есть специальная функция - count.

Функция count

Функция count() вычисляет количество элементов массива. На самом деле эта функция вычисляет число элементов в переменной вообще. Если применить ее к любой другой переменной, она возвратит 1. Исключение составляет переменная типа NULL – count(NULL) есть 0. Кроме того, применяя эту функцию к многомерному массиву, чтобы получить число его элементов, нужно использовать дополнительный параметр COUNT_RECURSIVE.

```
<?
$del_items = array("langs" => array("10"=>"Python",
"12"=>"Lisp"),
        "other"=>"Информатика");
echo count($del_items) . "<br>";
    // выведет 2
echo count($del_items,COUNT_RECURSIVE);
    // выведет 4
?>
```

Функция in_array

in_array("искомое значение","массив",["ограничение на тип"]);

позволяет установить, содержится ли в заданном массиве искомое значение. Если третий аргумент задан как true, то в массиве

нужно найти элемент, совпадающий с искомым не только по значению, но и по типу. Если искомое значение – строка, то сравнение чувствительно к регистру.

Например, имеется массив не изученных нами языков программирования. Мы хотим узнать, содержится ли в этом массиве язык PHP. Напишем следующую программу:

```
<?php
$langs = array("Lisp","Python","Java",
                "PHP","Perl");
if (in_array("PHP",$langs,true))
    echo "Надо бы изучить PHP<br>";
// выведет сообщение "Надо бы изучить PHP"
if (in_array("php",$langs))
    echo "Надо бы изучить php<br>";
// ничего не выведет, поскольку в массиве
// есть строка "PHP", а не "php"
?>
```

В качестве искомого значения этой функции может выступать и массив:

```
<?php
$langs =
array("Lisp","Python",array("PHP","Java"),"Perl");
if (in_array(array("PHP","Java"),$langs))
  echo "Надо бы изучить PHP и Java<br>";
?>
```

Функция array_search

Функция array_search - функция для поиска значения в массиве.

В отличие от in_array в результате работы array_search возвращает значение ключа, если элемент найден, и ложь – в противном случае. А вот синтаксис у этих функций одинаковый:

array_search("искомое значение","массив",["ограничение на тип"]);

Сравнение строк чувствительно к регистру, а если указан опциональный аргумент, то сравниваются еще и типы значений.

Пусть у нас есть массив языков программирования, которые мы знаем. Причем ключом каждого элемента является номер, указывающий, каким по счету был изучен этот язык.

```
<?php
$langs = array("","Lisp","Python","Java","PHP","Perl");
if (!array_search("PHP",$langs))
    echo "Надо бы изучить PHP<br>";
else {
  $k = array_search("PHP",$langs);
  echo "PHP я изучил $k-м";
     }
?>
```

Очевидно, что эта функция более функциональна, чем in_array, поскольку мы не только получаем информацию о том, что искомый элемент в массиве есть, но и узнаем, где именно в массиве он находится. А что будет, если искомых элементов в массиве несколько? В таком случае функция array_search() вернет ключ первого из найденных элементов. Чтобы получить ключи всех элементов, нужно воспользоваться функцией array_keys().

Функция array_keys

Функция array_keys() выбирает все ключи массива. Но у нее имеется дополнительный аргумент, с помощью которого можно получить список ключей элементов с конкретным значением. Синтаксис этой функции таков:

array_keys ("массив",["значение для поиска"])

Функция array_keys() возвращает как строковые, так и числовые ключи массива, организуя все значения в виде нового массива с числовыми индексами.

В предыдущем примере мы записали массив языков, которые изучили. Список был длинным, и некоторые языки были записаны

несколько раз. У нас возникло подозрение, что один из таких языков – Lisp. Давайте это проверим:

```
<?php
$langs =
array("Lisp","Python","Java","PHP",
      "Perl","Lisp");
$lisp_keys = array_keys($langs,"Lisp");
echo "Lisp входит в массив ".
          count($lisp_keys) ." раза:<br>";
foreach ($lisp_keys as $val){
    echo "под номером $val <br>";
}
?>
```

Функция array_keys(), как и две предыдущие, зависит от регистра, т.е. элементов LISP в массиве она не обнаружит.

Если есть функция для получения всех ключей массива, то можно предположить, что существует и функция для получения всех значений массива. Действительно, она существует. Это функция array_values(массив). Все значения переданного ей массива записываются в новый массив, проиндексированный целыми числами, т.е. все ключи массива теряются, остаются только значения. Но вернемся к нашему примеру.

Итак, мы выяснили, что язык Lisp случайно упомянут в нашем массиве дважды. Поскольку изучить один язык дважды нельзя («учил, но забыл» не считается), то нужно как-то избавиться от повторяющихся языков. Сделать это довольно просто с помощью функции array_unique().

Функция array_unique

Функция array_unique(массив) возвращает новый массив в котором повторяющиеся элементы фигурируют в одном экземпляре. Таким образом, вместо нескольких одинаковых значений и их ключей мы имеем одно значение. Какой у него будет ключ? Как из нескольких ключей одинаковых элементов выбирается тот, который будет сохранен в новом массиве? Происходит следующее. Все элементы массива преобразуются в строки и сортируются. Затем

обработчик запоминает первый ключ для каждого значения, а остальные ключи игнорирует.

Попробуем избавиться от повторяющихся языков в списке изученных.

```
<?php
$langs =
array("Lisp","Java","Python","Java",
      "PHP","Perl","Lisp");
print_r(array_unique($langs));
?>
```

Функция sort

Функция sort имеет следующий синтаксис

sort (массив [, флаги])

и сортирует массив, т.е. упорядочивает его значения по возрастанию. Эта функция удаляет все существовавшие в массиве ключи, заменяя их числовыми индексами, соответствующими новому порядку элементов. В случае успешного завершения работы она возвращает true, иначе – false.

Пусть у нас есть два массива: цены товаров – их названия и, наоборот, названия товаров – их цены. Упорядочим эти массивы по возрастанию:

```
<?
$items = array(10 => "хлеб", 20 => "молоко",
  30 => "бутерброд");
sort($items);
    // строки сортируются в алфавитном
    // порядке, ключи теряются
print_r($items);

$rev_items = array("хлеб" => 10,
    "бутерброд" => 30, "молоко" => 20);
sort($rev_items);
    // числа сортируются по возрастанию,
    // ключи теряются
print_r($rev_items);
?>
```

В качестве дополнительного аргумента может использоваться одна из следующих констант:

- SORT_REGULAR – сравнивать элементы массива обычным образом;
- SORT_NUMERIC – сравнивать элементы массива как числа;
- SORT_STRING – сравнивать элементы массива как строки.

Функции asort, rsort, arsort

Если требуется сохранять индексы элементов массива после сортировки, то нужно использовать функцию asort (массив [, флаги]). Если необходимо отсортировать массив в обратном порядке, т.е. от наибольшего значения к наименьшему, то можно задействовать функцию rsort (массив [, флаги]). А если при этом нужно еще и сохранить значения ключей, то следует использовать функцию arsort(массив [, флаги]). Как вы, наверное, заметили синтаксис у этих функций абсолютно такой же, как у функции sort. Соответственно и значения флагов могут быть такими же, как у sort: SORT_REGULAR, SORT_NUMERIC, SORT_STRING.

```php
<?php
$books = array("Пушкин"=>"Руслан и Людмила",
    "Толстой"=>"Война и мир",
    "Лермонтов"=>"Герой нашего времени");
asort($books);
    // сортируем массив,
    // сохраняя значения ключей
print_r($books);
echo "<br>";
rsort($books);
    // сортируем массив в обратном порядке,
    // ключи будут заменены
print_r($books);
?>
```

В результате работы этого скрипта получим:

```
Array ( [Толстой] => Война и мир [Лермонтов] => Герой нашего времени [Пушкин] => Руслан и Людмила )
Array ( [0] => Руслан и Людмила [1] => Герой нашего времени [2] => Война и мир )
```

Функции ksort(), krsort()

Очевидно, что может возникнуть необходимость в сортировке массива по значениям ключей. Например, если у нас есть массив данных о книгах, как в приведенном выше примере, то вполне вероятно, что мы захотим отсортировать книги по именам авторов. Для этого в PHP также не нужно писать много строк кода – можно просто воспользоваться функцией ksort() для сортировки по возрастанию (прямой порядок сортировки) или krsort() – для сортировки по убыванию (обратный порядок сортировки). Синтаксис этих функций опять же аналогичен синтаксису функции sort().

```
<?php
$books = array("Пушкин"=>"Руслан и Людмила",
     "Толстой"=>"Война и мир",
     "Лермонтов"=>"Герой нашего времени");
ksort($books);
    // сортируем массив,
    // сохраняя значения ключей
print_r($books);
?>
```

Функции usort(), uksort()

Кроме двух простых способов сортировки значений массива (по убыванию или по возрастанию) PHP предлагает пользователю возможность самому задавать критерии для сортировки данных. Критерий задается с помощью функции, имя которой указывается в качестве аргумента для специальных функций сортировки usort() или uksort(). По названиям этих функций можно догадаться, что usort() сортирует значения элементов массива, а uksort() – значения ключей массива с помощью определенной пользователем функции. Обе функции возвращают true, если сортировка прошла успешно, и false – в противном случае. Их синтаксис выглядит следующим образом:

usort (массив , сортирующая функция)
uksort (массив , сортирующая функция)

Конечно же, нельзя сортировать массив с помощью любой пользовательской функции. Эта функция должна удовлетворять определенным критериям, позволяющим сравнивать элементы массива. Как должна быть устроена сортирующая функция?

Во-первых, она должна иметь два аргумента. В них интерпретатор будет передавать пары значений элементов для функции usort() или ключей массива для функции uksort().

Во-вторых, сортирующая функция должна возвращать:

- целое число, меньшее нуля, если первый аргумент меньше второго;
- число, равное нулю, если два аргумента равны;
- число большее нуля, если первый аргумент больше второго.

Как и для других функций сортировки, для функции usort() существует аналог, не изменяющий значения ключей, – функция uasort().

Допустим, у нас есть массив, содержащий такие сведения о литературных произведениях, как название, автор и год создания. Мы хотим упорядочить книги по дате создания.

```
<?php
// массив выглядит таким образом:
$books = array("Герой нашего времени" =>
              array ("Лермонтов", 1840),
  "Руслан и Людмила" => array("Пушкин",1820),
  "Война и мир" => array ("Толстой",1863),
  "Идиот" => array("Достоевский",1868));
/* можно, конечно переписать этот массив
по-другому, сделав год издания, например,
индексом, но гораздо удобнее написать свою
функцию для сортировки */

uasort($books,"cmp");
// сортируем массив с помощью функции cmp

foreach ($books as $key => $book) {
   echo "$book[0]: \"$key\"<br>";
}
function cmp($a,$b){
// функция, определяющая способ сортировки
   if ($a[1] < $b[1]) return -1;
```

```
        elseif ($a[1]==$b[1]) return 0;
        else return 1;
    }
    ?>
```

Функция array_walk

Функция array_walk(массив, функция [, данные]) применяет созданную пользователем функцию ко всем элементам массива массив и возвращает true в случае успешного выполнения операции и false – в противном случае.

Пользовательская функция, как правило, имеет два аргумента, в которые поочередно передаются значение и ключ каждого элемента массива. Но если при вызове функции array_walk() указан третий аргумент, то он будет рассмотрен как значение третьего аргумента пользовательской функции, смысл которого определяет сам пользователь. Если функция пользователя требует больше аргументов, чем в нее передано, то при каждом вызове array_walk() будет выдаваться предупреждение.

Если необходимо работать с реальными значениями массива, а не с их копиями, следует передавать аргумент в функцию по ссылке. Однако нужно иметь в виду, что нельзя добавлять или удалять элементы массива и производить действия, изменяющие сам массив, поскольку в этом случае результат работы array_walk() считается неопределенным.

```
    <?php
    $books1 = array(
      "А.С. Пушкин"=>"Руслан и Людмила",
      "Л.Н. Толстой"=>"Война и мир",
      "М.Ю. Лермонтов"=>"Герой нашего времени");
    // создаем функцию, которую хотим применить к элементам
массива

    function try_walk($val,$key,$data){
       echo "$data \"$val\" написал $key<br>";
    }
    // применяем ко всем элементам массива
    // $books1 функцию try_walk
    array_walk($books1,"try_walk","Роман");
    ?>
```

Функция array_slice

Поскольку массив – это набор элементов, вполне вероятно, потребуется выделить из него какой-нибудь поднабор. В PHP для этих целей есть функция array_slice.

array_slice (массив, номер_элемента [, длина])

Эта функция выделяет подмассив длины длина в массиве массив, начиная с элемента, номер которого задан параметром номер_элемента. Положительный номер_элемента указывает на порядковый номер элемента относительно начала массива, отрицательный – на номер элемента с конца массива.

```
<?php
$arr = array(1,2,3,4,5);
$sub_arr = array_slice($arr,2);
print_r($sub_arr);
/*
выведет Array ( [0] => 3 [1] =>4 [2] => 5 ),
т.е. подмассив, состоящий из элементов
3, 4, 5 */
$sub_arr = array_slice($arr,-2);
print_r($sub_arr);
   // выведет Array ( [0] => 4 [1] => 5 ),
   // т.е. подмассив, из элементов 4, 5
?>
```

Если задать параметр длина при использовании array_slice, то будет выделен подмассив, имеющий ровно столько элементов, сколько задано этим параметром. Длину можно указывать и отрицательную. В этом случае интерпретатор удалит с конца массива число элементов, равное модулю параметра длина.

```
<?php
$arr = array(1,2,3,4,5);
$sub_arr = array_slice($arr, 2, 2);
print_r($sub_arr);
    // содержит массив из элементов 3, 4
$sub = array_slice($arr,-3, 2);
    // тоже содержит массив из элементов 3, 4
$sub1 = array_slice($arr,0, -1);
```

```
        // содержит массив из
        // элементов 1, 2, 3, 4
    $sub2 = array_slice($arr,-4, -2);
        // содержит массив из элементов 2, 3
    ?>
```

Функция array_chunk

Есть еще одна функция, похожая на array_slice() – это array_chunk(). Она разбивает массив на несколько подмассивов заданной длины. Синтаксис ее такой:

array_chunk (массив, размер [, сохранять_ключи])

В результате работы array_chunk() возвращает многомерный массив, элементы которого представляют собой полученные подмассивы. Если задать параметр сохранять ключи как true, то при разбиении будут сохранены ключи исходного массива. В противном случае ключи элементов заменяются числовыми индексами, которые начинаются с нуля.

Допустим у нас есть список приглашенных, оформленный в виде массива их фамилий. У нас имеются столики на три персоны. Поэтому нужно распределить всех приглашенных по трое.

```
    <?php
    $persons = array("Иванов", "Петров",
        "Сидорова","Зайцева", "Волкова");
    $triples = array_chunk($persons,3);
        // делим массив на подмассивы
        // по три элемента
    foreach ($triples as $k => $table){
        // выводим полученные тройки
      echo "За столиком номер $k сидят: <ul>";
      foreach ($table as $pers)
         echo "<li>$pers";
      echo "</ul>";
    }
    ?>
```

Функция array_sum()

В этом разделе мы познакомимся с функцией, вычисляющей сумму всех элементов массива. Сама задача вычисления суммы значений массива предельно проста. Но зачем писать лишний раз один и тот же код, если можно воспользоваться специально созданной и всегда доступной функцией. Функция эта называется, как можно догадаться, array_sum(). И в качестве параметра ей передается только имя массива, сумму значений элементов которого нужно вычислить.

В качестве примера использования этой функции приведем решение более сложной задачи, чем просто вычисление суммы элементов. Этот пример также иллюстрирует применение функции array_slice(), которую мы обсуждали чуть раньше.

Пусть дан массив натуральных чисел. Нужно найти в нем такое число, что сумма элементов справа от него равна сумме элементов слева от него.

```
<?php
$arr = array(2,1,3,4,5,6,4); //массив задается функцией array

// перебираем каждый элемент массива $arr.
// Внутри цикла текущий ключ массива содержится в переменной
$k,
// текущее значение - в переменной $val
foreach ($arr as $k => $val){
   $p = $k + 1;
// array_slice выделяет подмассив длины length в массиве
array,
// начиная с элемента offset.
   $out_next = array_slice($arr,$p);
   // получаем массив элементов, идущих после текущего
   $out_prev = array_slice($arr,0,$k);
   // получаем массив элементов, идущих перед текущим

// функция mixed array_sum (array array)
// подсчитывает сумму элементов массива array
   $next_sum = array_sum($out_next);
   $prev_sum = array_sum($out_prev);
// если сумма элементов до текущего равна
// сумме элементов после, то выводим
// значение текущего элемента
   if ($next_sum==$prev_sum)
       echo "value:$val";
```

```
// можно посмотреть, что представляют собой
// рассмотренные массивы на каждом шаге
// print_r($out_next); echo "<br>";
// print_r($out_prev);
// echo "$next_sum, $prev_sum<br>";
   echo "<hr>";
}
?>
```

2.9 Строки

Как отмечалось ранее существует три способа задания строк: с помощью одинарных кавычек, двойных кавычек и с помощью heredoc –синтаксиса.

```
<?php
echo 'В такой строке НЕ обрабатываются
      переменные и большинство
      последовательностей';
echo "Здесь переменные и последовательности
      обрабатываются";
echo <<<EOT
Здесь тоже обрабатываются как переменные,
так и управляющие последовательности.
И кроме того, можно вводить символы кавычек
без их экранирования обратным слэшем.
EOT;
?>
```

Уже не раз мы использовали функцию echo . На самом деле, echo – не функция, а языковая конструкция, поэтому использовать при ее вызове круглые скобки не обязательно.

Echo позволяет выводить на экран строки, переданные ей в качестве параметров. Параметров у echo может быть сколько угодно. Их разделяют запятыми или объединяют с помощью оператора конкатенации и никогда не заключают в круглые скобки.

```
<?
echo "Пришел ", "увидел ", "победил ";
// выведет строку "Пришел увидел победил"
// многие предпочитают передавать несколько
// параметров в echo с помощью конкатенации
```

```
echo "Пришел " . "увидел " . "победил ";
// тоже выведет строку
// "Пришел увидел победил"
echo ("Пришел ", "увидел ", "победил ");
// выдаст ошибку: unexpected ','
?>
```

Существует сокращенный синтаксис для команды echo :

<?=строка_для_вывода?>

Здесь параметр строка_для_вывода содержит строку, заданную любым из известных способов, которая должна быть выведена на экран.

Кроме языковой конструкции echo существует ряд функций для вывода строк. Это в первую очередь функция print и ее разновидности printf, sprintf и т.п.

Функция print позволяет выводить на экран только одну строку и, как и echo , не может быть вызвана с помощью переменных функций, поскольку является языковой конструкцией.

Функция print_r не относится к строковым функциям, как можно было бы подумать. Она отображает информацию о переменной в форме, понятной пользователю. Кроме этой функции можно использовать var_dump(), которая более информативна.

2.9.1 Поиск элемента в строке

Для того чтобы определить, входит ли данная подстрока в состав строки, используется функция strpos() . Синтаксис strpos() такой:

strpos (исходная строка,строка для поиска [,с какого символа искать])

Она возвращает позицию появления искомой строки в исходной строке или возвращает логическое false, если вхождение не найдено.

Дополнительный аргумент позволяет задавать символ, начиная с которого будет производиться поиск. Кроме логического false эта функция может возвращать и другие значения, которые приводятся к false (например, 0 или ""). Поэтому для того, чтобы проверить, найдена ли искомая строка, рекомендуют использовать оператор эквивалентности " === ".

```
<?
$str = "Идея наносить данные на перфокарты и затем считывать и
обрабатывать их
автоматически принадлежала Джону Биллингсу,
а ее техническое решение осуществил Герман Холлерит.
Перфокарта Холлерита оказалась настолько удачной, что без
малейших изменений
просуществовала до наших дней.";
$pos = strpos($str,"Холлерит");
if ($pos !== false)
echo "Искомая строка   встречена в позиции номер $pos ";
else echo "Искомая строка не найдена";
?>
```

Если значение параметра строка_для_поиска не является строкой, то оно преобразуется к целому типу и рассматривается как ASCII-код символа. Чтобы получить ASCII-код любого символа в PHP, можно воспользоваться функцией ord("символ").

Например, если мы напишем $pos = strpos($str,228); то интерпретатор будет считать, что мы ищем символ " д ". Если добавить эту строчку в приведенный выше пример и вывести результат, то получим сообщение, что искомая строка найдена в позиции 1.

Функция, обратная по смыслу ord, – это chr (код символа) . Она по ASCII-коду выводит символ, соответствующий этому коду.

С помощью функции strpos можно найти номер только первого появления строки в исходной строке.

Естественно, есть функции, которые позволяют вычислить номер последнего появления строки в исходной строке. Это функция strrpos() . Ее синтаксис таков:

strrpos (исходная строка, символ для поиска)

В отличие от strpos() эта функция позволяет найти позицию последнего появления в строке указанного символа.

Бывают ситуации, когда знать позицию, где находится искомая строка, необязательно, а нужно просто получить все символы, которые расположены после вхождения этой строки. Можно, конечно, воспользоваться и приведенными выше функциями strpos() и strrpos(), но можно сделать и проще – выделить подстроку с помощью предназначенных именно для этого функций.

2.9.2 Выделение подстроки

Функция strstr

Функция strstr (исходная строка, строка для поиска) находит первое появление искомой строки и возвращает подстроку, начиная с этой искомой строки до конца исходной строки.

Если строка для поиска не найдена, то функция вернет false. Если строка для поиска не принадлежит строковому типу данных, то она переводится в целое число и рассматривается как код символа. Кроме того, эта функция чувствительна к регистру, т.е. если мы будем параллельно искать вхождения слов "Идея" и "идея", то результаты будут разными. Вместо strstr() можно использовать абсолютно идентичную ей функцию strchr().

В качестве примера выделим из строки, содержащей название и автора исследования, подстроку, начинающуюся со слова "Название":

```
<?php
$str = "Автор: Иванов Иван (<a
href=mailto:van@mail.ru>написать письмо</a>),
   Название: 'Исследование языков
             программирования' ";
echo "<b>Исходная строка: </b>",$str;
if (!strstr($str, "Название"))
   echo "Строка не найдена<br>";
else echo "<p><strong>Полученная подстрока: </strong>",
   strstr($str, "Название");
```

```
?>
```

В результате получим:

Исходная строка: Автор: Иванов Иван (написать письмо), Название: 'Исследование языков программирования'

Полученная подстрока: Название: 'Исследование языков программирования'

Для реализации регистронезависимого поиска подстроки существует соответствующий аналог этой функции – функция stristr (исходная строка, искомая строка). Действует и используется она точно так же, как и strstr(), за исключением того, что регистр, в котором записаны символы искомой строки, не играет роли при поиске.

Очевидно, что функция strstr() не слишком часто используется – на практике редко бывает нужно получить подстроку, начинающуюся с определенного слова или строки. Но в некоторых случаях и она может пригодиться. Кроме того, в PHP есть и более удобные функции для поиска вхождений. Наиболее мощные из них, конечно, связаны с регулярными выражениями.

Функция substr

Иногда мы не знаем, с каких символов начинается искомая строка, но знаем, например, что начинается она с пятого символа и заканчивается за два символа до конца исходной строки. Как выделить подстроку по такому описанию? Очень просто, с помощью функции substr() . Ее синтаксис можно записать следующим образом:

substr (исходная строка,
позиция начального символа [, длина])

Эта функция возвращает часть строки длиной, заданной параметром длина, начиная с символа, указанного параметром позиция начального символа. Позиция, с которой начинается выделяемая подстрока, может быть как положительным целым числом, так и отрицательным. В последнем случае отсчет элементов производится с конца строки. Если параметр длина опущен, то

substr() возвращает подстроку от указанного символа и до конца исходной строки. Длина выделяемой подстроки тоже может быть задана отрицательным числом. Это означает, что указанное число символов отбрасывается с конца строки.

Допустим, у нас есть фраза, выделенная жирным шрифтом с помощью тега <b> языка HTML. Мы хотим получить эту же фразу, но в обычном стиле. Напишем такую программу:

```
<?php
$word = "<b>Hello, world!</b>";
echo $word , "<br>";
$pure_str = substr($word, 3, -4);
/* выделяем подстроку,
   начиная с 3-го символа,
   не включая 4 символа с конца строки */
echo $pure_str;
?>
```

В результате работы этого скрипта получим:

Hello, world!

Hello, world!

Функции strip_tags

На самом деле решить задачу, приведенную выше, можно гораздо проще, с помощью функции strip_tags :

strip_tags (строка [, допустимые теги])

Эта функция возвращает строку, из которой удалены все html и php-теги. С помощью дополнительного аргумента можно задать теги, которые не будут удалены из строки. Список из нескольких тегов вводится без каких-либо знаков разделителей. Функция выдает предупреждение, если встречает неправильные или неполные теги.

```
<?php
$string = "<b>Bold text</b>
           <i>Italic text</i>";
$str = strip_tags($string);
```

```
    // удаляем все теги из строки
$str1 = strip_tags($string, '<b>');
    // удаляем все теги кроме тега <b>
$str2 = strip_tags($string, '<i>');
    // удаляем все теги кроме тегов <i>
echo $str,"<br>",$str1,"<br>", $str2;
?>
```

В результате получим:
Bold text Italic text
Bold text Italic text
Bold text *Italic text*

Функция strlen

Длина строки определяется количеством символов, входящих в эту строку. Вычислить длину строки можно с помощью функции strlen (строка). Например, длина строки "Разработка информационной модели" вычисляется с помощью команды: strlen ("Разработка информационной модели"); и равна 32 символам.

2.9.3 Замена вхождения подстроки

Функция str_replace

Для замены вхождения подстроки можно использовать функцию str_replace() . Это простая и удобная функция, позволяющая решать множество задач, не требующих особых тонкостей при выборе заменяемой подстроки. Для того чтобы производить замены с более сложными условиями, используют механизм регулярных выражений и соответствующие функции ereg_replace() и preg_replace(). Синтаксис функции str_replace() такой:

str_replace(искомое значение,
  значение для замены, объект)

Функция str_replace() ищет в рассматриваемом объекте значение и заменяет его значением, предназначенным для замены. Почему мы

говорим здесь не про строки для поиска и замены и исходную строку, а про значения и объект, в котором происходит замена? Дело в том, что начиная с PHP 4.0.5 любой аргумент этой функции может быть массивом.

Если объект, в котором производится поиск и замена, является массивом, то эти действия выполняются для каждого элемента массива и в результате возвращается новый массив.

```
<?php
$greeting = array("Привет", "Привет всем!",
   "Привет, дорогая!"); // объект
$new_greet = str_replace("Привет",
   "Доброе утро", $greeting);
   // делаем замену
print_r($new_greet);
/* получим: Array ([0]=>Доброе утро
   [1]=>Доброе утро всем!
   [2]=>Доброе утро, дорогая!) */
?>
```

Если искомое значение и значение для замены – массивы, то берется по одному значению из каждого массива и производится их поиск и замена в объекте. Если значений для замены меньше, чем значений для поиска, то в качестве новых значений используется пустая строка.

```
<?php
$greeting = array("Привет", "Привет всем!",
   "Привет, дорогая!","Здравствуйте",
   "Здравствуйте, товарищи", "Hi");
   // объект
$search = array ("Привет",
   "Здравствуйте", "Hi");
   // значения, которые будем заменять
$replace = array ("Доброе утро",
   "День добрый");
   // значения, которыми будем заменять
$new_greet = str_replace($search, $replace,
                         $greeting);
   // делаем замену
print_r($new_greet);
   //выводим полученный массив
?>
```

В результате получим такой массив:

Array (
[0] => Доброе утро
[1] => Доброе утро всем!
[2] => Доброе утро, дорогая!
[3] => День добрый
[4] => День добрый, товарищи
[5] =>
)

Если значения для поиска – массив, а значение для замены – строка, то эта строка будет использована для замены всех найденных значений.

```php
<?php
$greeting = array("Привет", "Привет всем!",
   "Привет, дорогая!", "Здравствуйте",
   "Здравствуйте, товарищи");
   // объект
$search = array ("Привет","Здравствуйте");
   // значения, которые будем заменять
$replace = "День добрый";
   // значение, которым будем заменять
$new_greet = str_replace($search,
   $replace, $greeting);  // делаем замену
print_r($new_greet);
   //выводим полученный массив
?>
```

Получим:

Array (
[0] => День добрый
[1] => День добрый всем!
[2] => День добрый, дорогая!
[3] => День добрый
[4] => День добрый, товарищи
)

Функция str_replace() чувствительна к регистру, но существует ее регистронезависимый аналог – функция str_ireplace().

Еще один пример использования функции str_replace() – обработка шаблонов.

Обратимся в очередной раз к описанию какого-либо документа, например, статьи. Создадим форму для ввода данных пользователя и отобразим эти данные способом, заданным непосредственно пользователем.

```
<h2>Введите описание статьи</h2>
<form action=="sb1.php">
<table border=0>
<tr><td>Название </td><td><input
  type=text name=title > </td></tr>
<tr><td>Краткое содержание </td><td><input
  type=textarea name=description > </td></tr>
<tr><td>Автор </td><td><input
  type=text name=author > </td></tr>
<tr><td>Дата публикации </td><td><input
  type=text name=published ></td></tr>
<tr><td>Шаблон документа </td><td><textarea
  name=shablon ></textarea></td></tr>
</table>
<input type=submit value="Отправить">
</form>
```

Однако просто поля для ввода шаблона недостаточно. Один человек введет в него одно, другой – другое. Нужно договориться о том, как создавать шаблоны, что можно в них использовать, т.е. нужно придумать язык шаблонов. Например, мы договариваемся, что при создании шаблона можно задействовать любые html-теги, а набор спецсимволов вида <!имя_элемента> определяет значение элемента с именем имя_элемента. Далее, как обрабатывать такого рода шаблоны? Можно использовать функцию str_replace():

```
<?php
$tmpl = $_GET["shablon"];
/* шаблон, введенный пользователем.
Например, это может быть такая строка:
"<h1><!title></h1> <p><font
size=-1><!description></font></p><p
```

```
align=right><!author><br><!published></p>" */
function Show(){
  // функция, которая производит замену
  // элемента шаблона на его значение
global $tmpl;
foreach($_GET as $k => $v) {
   $tmpl = str_replace("<!$k>",$v,$tmpl);
}
echo $tmpl;
}
Show();
?>
```

Если мы введем в форму такие данные как показано на рисунке 2.1, то в результате получим:

Первая машина для переписи населения
Идея наносить данные на перфокарты и затем считывать и обрабатывать их автоматически принадлежала Джону Биллингсу, а ее техническое решение осуществил Герман Холлерит. Перфокарта Холлерита оказалась настолько удачной, что без малейших изменений просуществовала до наших дней.

А. М. Федотов
12.02.03

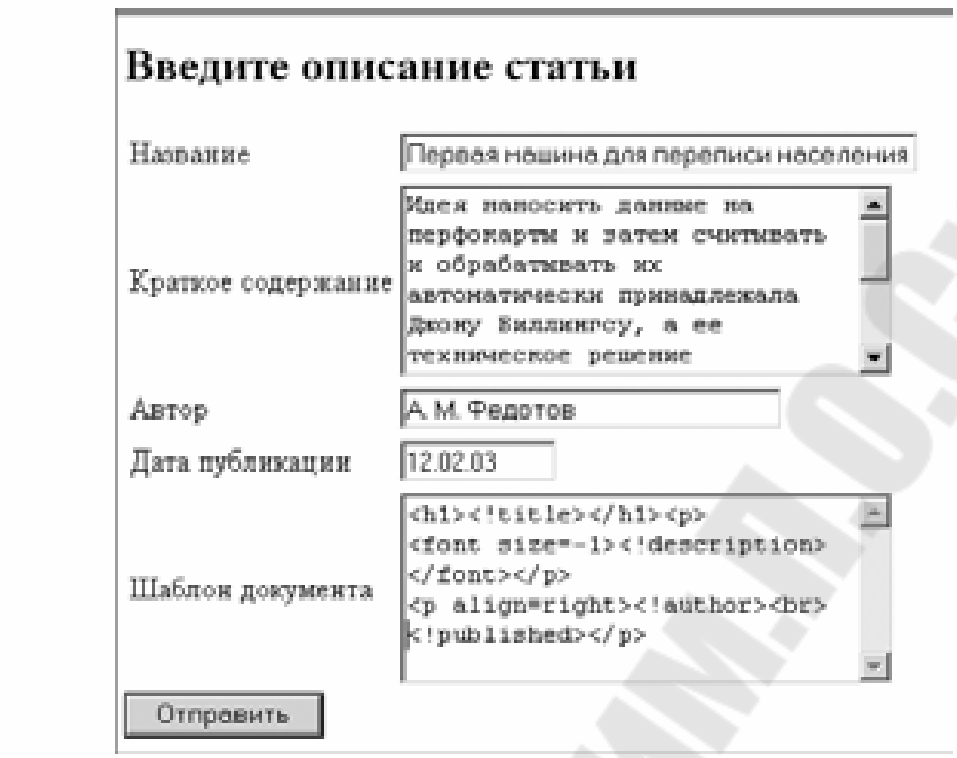


Рисунок 2.1 – Форма для ввода описания документа «статья» и шаблона для его отображения

Функция substr_replace

Эта функция сочетает в себе свойства двух уже рассмотренных нами функций – функции str_replace() и substr() . Ее синтаксис таков:

substr_replace (исходная строка,
строка для замены,
позиция начального символа [, длина])

Эта функция заменяет часть строки строкой, предназначенной для замены. Заменяется та часть строки (т.е. подстрока), которая начинается с позиции, указанной параметром позиция начального символа. С помощью дополнительного аргумента длина можно ограничить число заменяемых символов. То есть, фактически, мы не указываем конкретно строку, которую нужно заменить, мы только описываем, где она находится и, возможно, какую длину имеет. В этом отличие функции substr_replace() от str_replace() .

Как и в случае с функцией substr() аргументы позиция начального символа и длина могут быть отрицательными. Если позиция начального символа отрицательна, то замена производится, начиная с этой позиции относительно конца строки. Отрицательная длина задает, сколько символов от конца строки не должно быть заменено. Если длина не указывается, то замена происходит до конца строки.

```
<?php
$text = "Меня зовут Вася.";
echo "Исходная строка: $text<hr>\n";
/* Следующие две строки заменят всю
исходную строку строкой 'А меня – Петя' */
echo substr_replace($text, 'А меня – Петя',
     0) . "<br>\n";
echo substr_replace($text, 'А меня – Петя',
     0, strlen($text)) . "<br>\n";
// Следующая строка добавит слово 'Привет! '
// в начало исходной строки
echo substr_replace($text, 'Привет! ',
     0, 0) . "<br>\n";
// Следующие две строки заменят имя Вася
// на имя Иван в исходной строке
echo substr_replace($text, 'Иван', 11,
    -1) . "<br>\n";
echo substr_replace($text, 'Иван', -5,
    -1) . "<br>\n";
?>
```

В результате работы этого скрипта получим:
Исходная строка: Меня зовут Вася.

А меня – Петя
А меня – Петя
Привет! Меня зовут Вася.
Меня зовут Иван.
Меня зовут Иван.

2.9.4 Разделение и соединение строки

Очень полезные функции – функция разделения строки на части и обратная ей функция объединения строк в одну строку. Почему очень полезные? Например, если вы динамически генерируете форму по желанию пользователя, можно предложить ему вводить элементы для создания списка выбора, разделяя их каким-нибудь символом. И для того чтобы обработать полученный список значений, как раз и пригодится умение разбивать строку на кусочки. Для реализации такого разбиения в PHP можно использовать несколько функций:

explode(разделитель,исходная строка
[,максимальное число элементов]);

split (шаблон, исходная строка
[, максимальное число элементов]);

preg_split (шаблон, исходная строка
[, максимальное число элементов
[,флаги]])

Последние две функции работают с регулярными выражениями. Рассмотрим более простую функцию – explode() .

Функция explode() делит исходную строку на подстроки, каждая из которых отделена от соседней с помощью указанного разделителя, и возвращает массив полученных строк. Если задан дополнительный параметр максимальное число элементов, то число элементов в массиве будет не больше этого параметра, в последний элемент записывается весь остаток строки. Если в качестве разделителя указана пустая строка "", то функция explode() вернет false. Если символа разделителя в исходной строке нет, то возвращается исходная строка без изменений.

Кроме разделения строки на части иногда, наоборот, возникает необходимость объединения нескольких строк в одно целое. Функция, предлагаемая для этого языком PHP, называется implode() :

implode (string $glue , array $pieces)

Эта функция объединяет элементы массива с помощью переданного ей объединяющего элемента (например, запятой). В

отличие от функции explode(), порядок аргументов в функции implode() не имеет значения.

Допустим, мы храним имя, фамилию и отчество человека по отдельности, а выводить их на странице нужно вместе. Чтобы соединить их в одну строку, можно использовать функцию implode():

```
<?php
$data = array("Иванов","Иван","Иванович");
$str = implode(" ", $data);
echo $str;
?>
```

В результате работы этого скрипта получим строку:
Иванов Иван Иванович

У функции implode() существует псевдоним – функция join(), т.е. эти две функции отличаются лишь именами.

2.9.5 Строки, содержащие html-код

Достаточно часто мы работаем со строками, содержащими html-теги. Если отобразить такую строку в браузер с помощью обычных функций отображения данных echo() или print(), то мы не увидим самих html-тегов, а получим отформатированную в соответствии с этими тегами строку. Браузер обрабатывает все html-теги в соответствии со стандартом языка HTML. Иногда нам нужно видеть непосредственно строку, без обработки ее браузером. Чтобы этого добиться, нужно перед тем, как выводить, применить к ней функцию htmlspecialchars().

Функция htmlspecialchars (строка [, стиль кавычек [, кодировка]]) переводит специальные символы, такие как "<", ">", "&", """ , "'" в такие сущности языка HTML, как "<", ">", "&", """, "'" соответственно.

Дополнительный аргумент стиль кавычек определяет, как должны интерпретироваться двойные и одинарные кавычки. Он

может иметь одно из трех значений: ENT_COMPAT, ENT_QUOTES, ENT_NOQUOTES. Константа ENT_COMPAT означает, что двойные кавычки должны быть переведены в спецсимволы, а одинарные должны остаться без изменений. ENT_QUOTES говорит, что должны конвертироваться и двойные и одинарные кавычки, а ENT_NOQUOTES оставляет и те, и другие кавычки без изменений.

В параметре кодировка могут быть заданы такие кодировки, как UTF-8, ISO-8859-1 и другие (русские кодировки также поддерживаются).

```
<?php
$new = htmlspecialchars("<a
    href='mailto:au@mail.ru'>
    Написать письмо</a>", ENT_QUOTES);
echo $new;

/* наша строка перекодируется в такую:
&lt;a href=&#039;mailto:au@mail.ru&#039;&gt;
Написать письмо&lt;/a&gt; */
?>
```

В браузере мы увидим:

Написать письмо

Функция htmlspecialchars() перекодирует только наиболее часто используемые спецсимволы. Если необходимо конвертировать все символы в сущности HTML, следует задействовать функцию htmlentities(). Русские буквы при использовании этой функции тоже кодируются специальными последовательностями. Например, буква " А" заменяется комбинацией " À ". Ее синтаксис и принцип действия аналогичен синтаксису и принципу действия htmlspecialchars().

2.9.6 Механизм регулярных выражений

Регулярные выражения (РВ) пришли из UNIX и Perl. С помощью регулярных выражений можно искать и изменять текст, разбивать строку на подстроки и т.д. 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). Возникает вопрос – зачем придумывать что-то еще?

Основное преимущество РВ 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 РВ:

/\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 РВ 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]), которая возвращает TRUE, если совпадение найдено, и FALSE – в противном случае.

```
<?
//строка, в которой нужно что-то найти
$str = "Мой телефонный номер: ".
"33-22-44. Номер моего редактора: ".
"222-44-55 и 323-22-33";
//шаблон, по которому искать.
//Задает поиск семизначных номеров.
$pattern = "/\d{3}-\d{2}-\d{2}/m";
//функция, осуществляющая поиск
$num_match = preg_match_all ($pattern,
                        $str, $result);
//вывод результатов поиска
for ($i=0;$i<$num_match;$i++)
    echo "Совпадение $i: ".
                $result[0][$i]."<br>";
?>
```

Механизм регулярных выражений присутствует почти во всех современных языках программирования с небольшими отличиями, но суть остается той же. Из-за ограничения объемов пособия мы не описываем синтаксис и семантику регулярных выражений. О регулярных выражениях пишут целые книги! Надеемся, что слушатели уже изучали механизм РВ.

3 Основы клиент-серверных технологий

3.1 Протокол HTTP и способы передачи данных на сервер

Internet построен по многоуровневому принципу, от физического уровня, связанного с физическими аспектами передачи двоичной информации, и до прикладного уровня, обеспечивающего интерфейс между пользователем и сетью.

HTTP (HyperText Transfer Protocol, протокол передачи гипертекста) – это протокол прикладного уровня, разработанный для обмена гипертекстовой информацией в Internet.

HTTP предоставляет набор методов для указания целей запроса, отправляемого серверу. Эти методы основаны на дисциплине ссылок, где для указания ресурса, к которому должен быть применен данный метод, используется универсальный идентификатор ресурсов (Universal Resource Identifier) в виде местонахождения ресурса (Universal Resource Locator, URL) или в виде его универсального имени (Universal Resource Name, URN).

Сообщения по сети при использовании протокола HTTP передаются в формате, схожем с форматом почтового сообщения Internet (RFC-822) или с форматом сообщений MIME (Multipurpose Internet Mail Exchange).

HTTP используется для коммуникаций между различными пользовательскими программами и программами-шлюзами, предоставляющими доступ к существующим Internet-протоколам, таким как SMTP (протокол электронной почты), NNTP (протокол передачи новостей), FTP (протокол передачи файлов), Gopher и WAIS. HTTP разработан для того, чтобы позволять таким шлюзам через промежуточные программы-серверы (proxy) передавать данные без потерь.

Протокол реализует принцип запрос/ответ. Запрашивающая программа– клиент инициирует взаимодействие с отвечающей программой–сервером, и посылает запрос, содержащий:

- метод доступа;

- адрес URL;
- версию протокола;
- сообщение (похожее по форме на MIME) с информацией о типе передаваемых данных, информацией о клиенте, пославшем запрос, и, возможно, с содержательной частью (телом) сообщения.

Ответ сервера содержит:

- строку состояния, в которую входит версия протокола и код возврата (успех или ошибка);
- сообщение (в форме, похожей на MIME), в которое входит информация сервера, метаинформация (т.е. информация о содержании сообщения) и тело сообщения.

В протоколе не указывается, кто должен открывать и закрывать соединение между клиентом и сервером. На практике соединение, как правило, открывает клиент, а сервер после отправки ответа инициирует его разрыв.

Рассмотрим более подробно, в какой форме отправляются запросы на сервер.

3.2 Форма запроса клиента

Клиент отсылает серверу запрос в одной из двух форм: в полной или сокращенной. Запрос в первой форме называется соответственно полным запросом, а во второй форме – простым запросом.

Простой запрос содержит метод доступа и адрес ресурса. Формально это можно записать так:

<Простой-Запрос> := <Метод> <символ пробел>
<Запрашиваемый-URL> <символ новой строки>

В качестве метода могут быть указаны GET, POST, HEAD, PUT, DELETE и другие. В качестве запрашиваемого URL чаще всего используется URL-адрес ресурса.

Пример простого запроса:

GET http://phpbook.info/

Здесь GET – это метод доступа, т.е. метод, который должен быть применен к запрашиваемому ресурсу, а http://phpbook.info/ – это URL-адрес запрашиваемого ресурса.

Полный запрос содержит строку состояния, несколько заголовков (заголовок запроса, общий заголовок или заголовок содержания) и, возможно, тело запроса. Формально общий вид полного запроса можно записать так:

<Полный запрос> := <Строка Состояния>
(<Общий заголовок>|<Заголовок запроса>|
<Заголовок содержания>)
<символ новой строки>
[<содержание запроса>]

Квадратные скобки здесь обозначают необязательные элементы заголовка, через вертикальную черту перечислены альтернативные варианты. Элемент <Строка состояния> содержит метод запроса и URL ресурса (как и простой запрос) и, кроме того, используемую версию протокола HTTP. Например, для вызова внешней программы можно задействовать следующую строку состояния:

POST http://phpbook.info/cgi-bin/test HTTP/1.0

В данном случае используется метод POST и протокол HTTP версии 1.0.

В обеих формах запроса важное место занимает URL запрашиваемого ресурса. Чаще всего URL используется в виде URL-адреса ресурса. При обращении к серверу можно применять как полную форму URL, так и упрощенную.

Полная форма содержит тип протокола доступа, адрес сервера ресурса и адрес ресурса на сервере (рисунок 3.1).

В сокращенной форме опускают протокол и адрес сервера, указывая только местоположение ресурса от корня сервера. Полную

форму используют, если возможна пересылка запроса другому серверу. Если же работа происходит только с одним сервером, то чаще применяют сокращенную форму.

Рисунок 3.1 – Полная форма URL

3.3 Использование HTML-форм для передачи данных на сервер

Для передачи данных на сервер в языке HTML есть специальная конструкция – формы. Формы предназначены для того, чтобы получать от пользователя информацию. Например, вам нужно знать логин и пароль пользователя для того, чтобы определить, на какие страницы сайта его можно допускать. Или вам необходимы личные данные пользователя, чтобы была возможность с ним связаться. Формы как раз и применяются для ввода такой информации. В них можно вводить текст или выбирать подходящие варианты из списка. Данные, записанные в форму, отправляются для обработки специальной программе (например, скрипту на PHP) на сервере. В зависимости от введенных пользователем данных эта программа может формировать различные web-страницы, отправлять запросы к базе данных, запускать различные приложения и т.п.

Для создания формы в языке HTML используется тег FORM. С помощью атрибутов action и method тега FORM задаются имя программы, которая будет обрабатывать данные формы, и метод запроса, соответственно. Внутри тега FORM размещаются различные элементы ввода – текстовые поля, кнопки и т.д. Отправка данных формы происходит после нажатия кнопки input типа submit.

Пусть у нас есть форма регистрации участников заочной школы программирования, включающая поля ввода, радиокнопки, текстовое поле и стандартные кнопки «Submit» и «Reset»

```
<h2>Форма для регистрации участников</h2>
<form action="1.php" method=POST> <!--создаем форму-->
<!--данные формы будет обрабатывать файл 1.php, при
    отправке запроса будет использован метод POST-->
Имя <br><input type=text name="first_name"
    value="Введите Ваше имя"><br>
Фамилия <br><input type=text name="last_name"><br>
E-mail <br><input type=text name="email"><br>
<p>
Выберите курс, который вы бы хотели посещать:<br>
<input type=radio name="kurs" value="PHP">PHP<br>
<input type=radio name="kurs" value="Lisp">Lisp<br>
<input type=radio name="kurs" value="Perl">Perl<br>
<input type=radio name="kurs" value="Unix">Unix<br>
<P>Что вы хотите, чтобы мы знали о вас? <BR>
<textarea name="comment" cols=32 rows=5></textarea>
<P><input name="confirm" type=checkbox
    checked>Подтвердить получение <br>
<input type=submit value="Отправить">
<input type=reset value="Отменить">
</form>
```

После обработки браузером этот файл будет выглядеть примерно так:

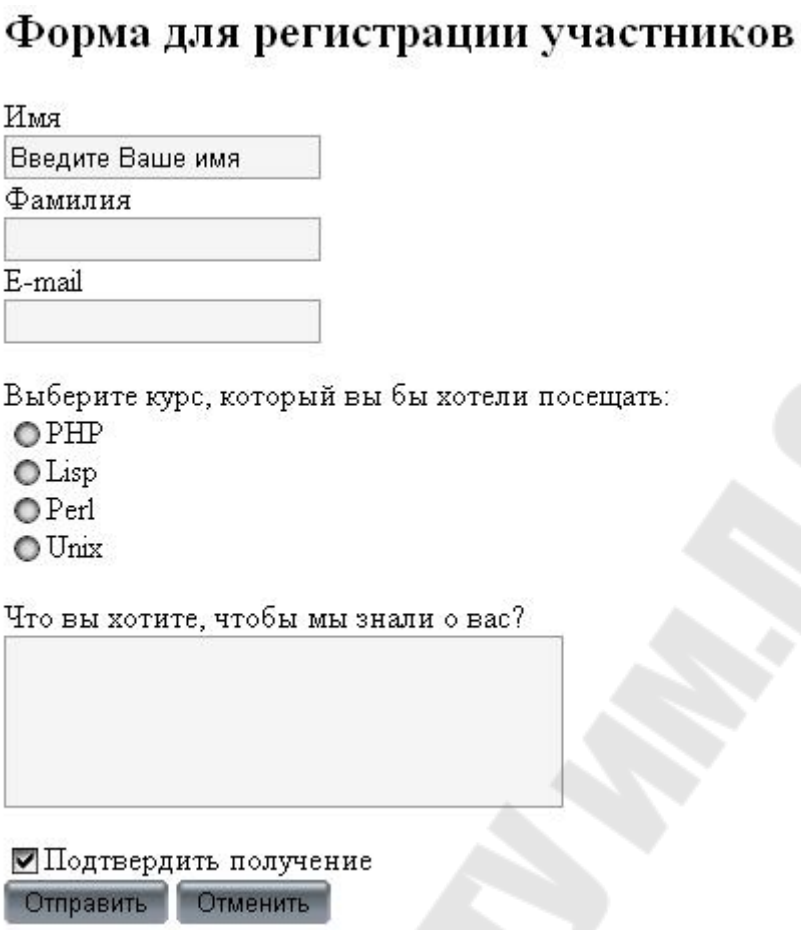


Рисунок 3.2 – Пример html-формы

При отправке данных формы с помощью метода GET содержимое формы добавляется к URL после знака вопроса в виде пар имя=значения, объединенных с помощью амперсанда &:

action?name1=value1&name2=value2&name3=value3

Здесь action – это URL-адрес программы, которая должна обрабатывать форму (это либо программа, заданная в атрибуте action тега form, либо сама текущая программа, если этот атрибут опущен). Имена name1, name2, name3 соответствуют именам элементов формы, а value1, value2, value3 – значениям этих элементов. Все специальные символы, включая = и &, в именах или значениях этих параметров будут закодированы. Поэтому не стоит использовать в названиях или значениях элементов формы эти символы и символы кириллицы в идентификаторах.

Если в поле для ввода ввести какой-нибудь служебный символ, то он будет передан в его шестнадцатеричном коде, например, символ $ заменится на %24. Так же передаются и русские буквы.

Для полей ввода текста и пароля (это элементы input с атрибутом type=text и type=password), значением будет то, что введет пользователь. Если пользователь ничего не вводит в такое поле, то в строке запроса будет присутствовать элемент name=, где name соответствует имени этого элемента формы.

Для кнопок типа checkbox и radio button значение value определяется атрибутом VALUE в том случае, когда кнопка отмечена. Не отмеченные кнопки при составлении строки запроса игнорируются целиком. Несколько кнопок типа checkbox могут иметь один атрибут NAME (и различные VALUE), если это необходимо. Кнопки типа radio button предназначены для одного из всех предложенных вариантов и поэтому должны иметь одинаковый атрибут NAME и различные атрибуты VALUE.

В принципе создавать HTML-форму для передачи данных методом GET не обязательно. Можно просто добавить в строку URL нужные переменные и их значения:

http://phpbook.info/test.php?id=10&user=pit

В связи с этим у передачи данных методом GET есть один существенный недостаток – любой может подделать значения параметров. Поэтому не советуем использовать этот метод для доступа к защищенным паролем страницам, для передачи информации, влияющей на безопасность работы программы или сервера. Кроме того, не стоит применять метод GET для передачи информации, которую не разрешено изменять пользователю.

Несмотря на все эти недостатки, использовать метод GET достаточно удобно при отладке скриптов (тогда можно видеть значения и имена передаваемых переменных) и для передачи параметров, не влияющих на безопасность.

При использовании метода POST содержимое формы кодируется точно так же, как для метода GET, но вместо добавления строки к URL содержимое запроса посылается блоком данных как часть операции POST. Если присутствует атрибут ACTION, то значение URL, которое там находится, определяет, куда посылать этот блок данных. Этот метод рекомендуется для передачи больших по объему блоков данных.

Информация, введенная пользователем и отправленная серверу с помощью метода POST, подается на стандартный ввод программе, указанной в атрибуте action, или текущему скрипту, если этот атрибут опущен. Длина посылаемого файла передается в переменной окружения CONTENT_LENGTH, а тип данных – в переменной CONTENT_TYPE.

Передать данные методом POST можно только с помощью HTML-формы, поскольку данные передаются в теле запроса, а не в заголовке, как в GET. Соответственно и изменить значение параметров можно, только изменив значение, введенное в форму. При использовании POST пользователь не видит передаваемые серверу данные.

Основное преимущество POST запросов – это их большая безопасность и функциональность по сравнению с GET-запросами. Поэтому метод POST чаще используют для передачи важной информации, а также информации большого объема. Тем не менее не стоит целиком полагаться на безопасность этого механизма, поскольку данные POST запроса также можно подделать, например создав html-файл на своей машине и заполнив его нужными данными. Кроме того, не все клиенты могут применять метод POST, что ограничивает варианты его использования.

При отправке данных на сервер любым методом передаются не только сами данные, введенные пользователем, но и ряд переменных, называемых переменными окружения, характеризующих клиента, историю его работы, пути к файлам и т.п. Вот некоторые из переменных окружения:

- REMOTE_ADDR – IP-адрес хоста (компьютера), отправляющего запрос;
- REMOTE_HOST – имя хоста, с которого отправлен запрос;
- HTTP_REFERER – адрес страницы, ссылающейся на текущий скрипт;
- REQUEST_METHOD – метод, который был использован при отправке запроса;
- QUERY_STRING – информация, находящаяся в URL после знака вопроса;
- SCRIPT_NAME – виртуальный путь к программе, которая должна выполняться;
- HTTP_USER_AGENT – информация о браузере, который использует клиент.

3.4 Обработка запросов с помощью PHP

До сих пор мы упоминали только, что запросы клиента обрабатываются на сервере с помощью специальной программы. На самом деле эту программу мы можем написать сами, в том числе и на языке PHP, и она будет делать с полученными данными все, что мы захотим. Для того, чтобы написать эту программу, необходимо познакомиться с некоторыми правилами и инструментами, предлагаемыми для этих целей PHP.

Внутри PHP-скрипта имеется несколько способов получения доступа к данным, переданным клиентом по протоколу HTTP. До версии PHP 4.1.0 доступ к таким данным осуществлялся по именам переданных переменных (напомним, что данные передаются в виде пар «имя переменной, символ «=», значение переменной»). Таким образом, если, например, было передано first_name=, то внутри скрипта появлялась переменная $first_name со значением Ivan. Если требовалось различать, каким методом были переданы данные, то использовались ассоциативные массивы $HTTP_POST_VARS и $HTTP_GET_VARS, ключами которых являлись имена переданных переменных, а значениями – соответственно значения этих

переменных. Таким образом, если пара first_name=Ivan передана методом GET, то $HTTP_GET_VARS["first_name"]="Ivan".

Использовать в программе имена переданных переменных напрямую небезопасно. Поэтому было решено начиная с PHP 4.1.0 задействовать для обращения к переменным, переданным с помощью HTTP-запросов, специальный массив – $_REQUEST. Этот массив содержит данные, переданные методами POST и GET, а также с помощью HTTP cookies. Это суперглобальный ассоциативный массив, т.е. его значения можно получить в любом месте программы, используя в качестве ключа имя соответствующей переменной (элемента формы).

Допустим, мы создали форму для регистрации участников заочной школы программирования (рисунок 3.2). Тогда в файле 1.php, обрабатывающем эту форму, можно написать следующее

```
<?php
$str = "Здравствуйте,
    ".$_REQUEST["first_name"]. "
    ".$_REQUEST["last_name"]."! <br>";
$str .="Вы выбрали для изучения курс по
    ".$_REQUEST["kurs"];
echo $str;
?>
```

Тогда, если в форму мы ввели имя «Вася», фамилию «Петров» и выбрали среди всех курсов курс по PHP, на экране браузера получим такое сообщение:

Здравствуйте, Вася Петров!

Вы выбрали для изучения курс по PHP

После введения массива $_REQUEST массивы $HTTP_POST_VARS и $HTTP_GET_VARS для однородности были переименованы в $_POST и $_GET соответственно, но сами они из обихода не исчезли из соображений совместимости с предыдущими версиями PHP. В отличие от своих предшественников, массивы $_POST и $_GET стали суперглобальными, т.е. доступными

напрямую и внутри функций и методов. Они сипользуются точно также, как и массив $_REQUEST.

Для того, чтобы сохранить возможность обработки скриптов более ранних версий, чем PHP 4.1.0, была введена директива register_globals, разрешающая или запрещающая доступ к переменным непосредственно по их именам. Если в файле настроек PHP параметр register_globals=On, то к переменным, переданным серверу методами GET и POST, можно обращаться просто по их именам (т.е. можно писать $first_name). Если же register_globals=Off, то нужно писать $_REQUEST["first_name"] или $_POST["first_name"], $_GET["first_name"]. С точки зрения безопасности эту директиву лучше отключать (т.е. register_globals=Off). При включенной директиве register_globals перечисленные выше массивы также будут содержать данные, переданные клиентом.

Иногда возникает необходимость узнать значение какой-либо переменной окружения, например метод, использовавшийся при передаче запроса или IP-адрес компьютера, отправившего запрос. Получить такую информацию можно с помощью функции getenv(). Она возвращает значение переменной окружения, имя которой передано ей в качестве параметра.

```
<?
getenv('REQUEST_METHOD');
    // возвратит использованный метод
echo getenv ('REMOTE_ADDR');
    // выведет IP-адрес пользователя,
    // пославшего запрос
?>
```

Как мы уже говорили, если используется метод GET, то данные передаются добавлением строки запроса в виде пар «имя_переменной=значение к URL-адресу ресурса». Все, что записано в URL после знака вопроса, можно получить с помощью команды

getenv('QUERY_STRING');

Благодаря этому можно по методу GET передавать данные в каком-нибудь другом виде. Например, указывать только значения нескольких параметров через знак плюс, а в скрипте разбирать строку запроса на части или можно передавать значение всего одного параметра. В этом случае в массиве $_GET появится пустой элемент с ключом, равным этому значению (всей строке запроса), причем символ «+», встретившийся в строке запроса, будет заменен на подчеркивание «_».

Методом POST данные передаются только с помощью форм, и пользователь (клиент) не видит, какие именно данные отправляются серверу. Чтобы их увидеть, хакер должен подменить нашу форму своей. Тогда сервер отправит результаты обработки неправильной формы не туда, куда нужно. Чтобы этого избежать, можно проверять адрес страницы, с которой были посланы данные. Это можно сделать опять же с помощью функции getenv():

```
getenv('HTTP_REFERER');
```

Расширим сценарий регистрации участников заочной школы. По полученным сведениям от зарегистрировавшегося человека, скрипт генерирует соответствующее сообщение. Если человек выбрал какие-то курсы, то ему выводится сообщение о времени их проведения и о лекторах, которые их читают. Если человек ничего не выбрал, то выводится сообщение о следующем собрании заочной школы программистов (ЗШП).

```
<?php // создадим массивы соответствий
$times = array("PHP"=>"14.30","Lisp"=>"12.00",
    "Perl"=>"15.00","Unix"=>"14.00");
$lectors = array("PHP"=>"Василий Васильевич",
"Lisp"=>"Иван Иванович", "Perl"=>"Петр Петрович",
"Unix"=>"Семен Семенович");
define("SIGN","С уважением, администрация");
    // определяем подпись письма как константу
define("MEETING_TIME","18.00"); // задаем время собрания
студентов
```

```php
$date = "12 мая"; // задаем дату проведения и начинаем
составлять текст сообщения
$str = "Здравствуйте, уважаемый " . $_POST["first_name"]
    . " " . $_POST["last_name"]."!<br>";
$str .= "<br>Сообщаем Вам, что ";
$kurses = $_POST["kurs"];   // сохраним в этой переменной
список выбранных курсов
if (!isset($kurses)) { // если не выбран ни один курс
    $event = "следующее собрание студентов";
    $str .= "$event состоится $date ". MEETING_TIME . "<br>";
} else { // если хотя бы один курс выбран
    $event = "выбранные Вами лекции состоятся $date <ul>";
        //функция count вычисляет число элементов в массиве
    $lect = "";
    for ($i=0;$i<count($kurses);$i++){ // для каждого
выбранного курса
        $k = $kurses[$i]; // запоминаем название курса
        $lect = $lect . "<li>лекция по $k в $times[$k]"; //
составляем сообщение
        $lect .= " (Ваш лектор, $lectors[$k])";
    }
    $event = $event . $lect . "</ul>";
    $str .= "$event";
}
$str .= "<br>". SIGN; // добавляем подпись
echo $str; // выводим сообщение на экран ?>
```

3.5 Механизм сессий в PHP

Давайте разберем, что такое сессии и в чем их специфика в PHP, решим одну из основных задач, возникающих при построении более-менее сложных информационных систем (сайтов) – задачу авторизации доступа пользователей к ресурсам системы, а также обсудим безопасность построенного решения.

3.5.1 Авторизация доступа

Что такое авторизация доступа? Попробуем объяснить на примере из обычной жизни. 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, который приходит в голову, - 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 соответствие 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.

Пусть у нас имеется файл index.html - домашняя страничка Васи Петрова

```
<head><title>My home page</title></head>
<body>
Привет всем!
Меня зовут Вася Петров и
это моя домашняя страничка.
<a href="secret_info.html">Для Пети</a>
</body></html>
```

и файл secret_info.html, который содержит секретную информацию, читать которую разрешено только Васиному другу Пете.

```
<html>
<head><title>Secret info</title></head>
<body>
Здесь я хочу делиться секретами
с другом Петей.</p>
</body></html>
```

Если оставить оба эти файла как есть, то любой посетитель, кликнув на ссылку "Для Пети", попадет на секретную страничку. Чтобы этого избежать, нужно добавить промежуточный скрипт, который будет проверять, действительно ли Петя хочет попасть на секретную страничку. И сделать так, чтобы главный файл ссылался не сразу на secret_info.html, а сначала на этот скрипт.

```
<html>
```

```
<head><title>My home page</title></head>
<body>
<p>Привет всем!
Меня зовут Вася Петров и
это моя домашняя страничка.
</p>
<a href="authorize.php">Для Пети</a>
</body>
</html>
```

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 он таковой имеет). Чтобы 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.

3.5.2 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, Session IDentifier). 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.

Существует несколько способов передачи идентификатора сессии:

- с помощью cookies;
- с помощью параметров адресной строки.

Cookies были созданы специально как метод однозначной идентификации клиентов и представляют собой расширение протокола HTTP. В этом случае идентификатор сессии сохраняется во

временном файле на компьютере клиента, пославшего запрос. Метод, несомненно, хорош, но многие пользователи отключают поддержку cookies на своем компьютере из-за проблем с безопасностью.

Если используется адресная строка, в этом случае идентификатор сессии автоматически встраивается во все запросы (URL), передаваемые серверу, и хранится на стороне сервера.

Например: адрес http://green.nsu.ru/test.php превращается в адрес http://green.nsu.ru/test.php?PHPSESSID=ac4f4a45bdc893434c95dcaffb1c1811

Этот способ передачи идентификатора используется автоматически, если у браузера, отправившего запрос, выключены cookies. Он достаточно надежный - передавать параметры в адресной строке можно всегда. С другой стороны, идентификатор сессии можно подглядеть, воспользоваться сохраненным вариантом в строке браузера или подделать. Хотя, конечно, все эти проблемы либо надуманны либо их можно решить. Например, кто сможет запомнить строку из 32 различных символов? А если правильно организовать работу с сессиями (вовремя их уничтожать), то даже сохранившийся в браузере номер сессии ничего не даст. К вопросам безопасности мы еще вернемся в конце лекции.

Кроме перечисленных вариантов передачи идентификатора сессии, известно еще несколько, но мы их рассматривать не будем ввиду их сложности.

Настройка сессий

Прежде чем начать работать с сессиями, следует разобраться в том, как корректно настраивать их обработку интерпретатором PHP. Сама работа с сессиями в PHP поддерживается по умолчанию. Это значит, что устанавливать никаких дополнительных элементов не нужно. А вот знать, что записано в настройках этого модуля, полезно, чтобы избежать ошибок при работе с ним.

Настройки PHP, в том числе и для работы с сессиями, прописываются в файле php.ini. Обратимся к этому файлу.

Как мы уже знаем, идентификатор сессии (число, по которому можно уникально идентифицировать клиента, пославшего запрос) сохраняется либо на компьютере-сервере, либо на компьютере-клиенте, либо и там, и там.

Параметр session.save_path в php.ini, определяет, где на сервере будут храниться данные сессии. Из-за него чаще всего возникают проблемы для Windows-серверов, потому что по умолчанию значение session.save_path установлено в /tmp. И если в корневой директории сервера такой папки нет, то при запуске сессий будет выдаваться ошибка.

Сервер может обрабатывать большое количество сессий одновременно, и все их временные файлы будут храниться в директории, заданной параметром session.save_path. Если система плохо работает с папками большого размера, то удобно использовать поддиректории. Для этого, кроме названия папки, в значение параметра добавляют еще и число, определяющее глубину вложенности поддиректорий в этой папке: N;/dir. Это значение нужно обязательно взять в кавычки, поскольку точка с запятой является одним из символов комментариев в файле настроек PHP. Все директории и поддиректории для хранения данных сессии нужно создать самостоятельно.

Например: 2;/Temp определяет, что переменные сессий будут храниться в папках вида c:\Temp\0\a\, c:\Temp\0\b\ и т.п.

Хранение данных на стороне клиента осуществляется с помощью cookies. Работу PHP с cookies можно настроить, в частности, с помощью параметров session.use_cookies, session.cookie_lifetime и т.п.

Параметр session.use_cookies определяет, использовать ли cookies при работе с сессиями. По умолчанию эта опция включена (т.е. принимает значение "1").

Параметр session.cookie_lifetime задает длительность жизни cookies в секундах. По умолчанию это "0", т.е. данные в cookies считаются правильными до закрытия окна браузера.

Кроме этих параметров, полезными могут оказаться session.name, определяющий имя сессии, session.auto_start, позволяющий автоматически запускать сессии, session.serialize_handler, задающий способ кодировки данных сессии, и параметр session.cache_expire, определяющий, через сколько минут устаревает документ в кэше.

Имя сессии session.name по умолчанию устанавливается как PHPSESSID и используется в cookies как имя переменной, в которой хранится идентификатор сессии. Автоматический запуск сессий по умолчанию отключен, но его можно задать, сделав значение session.auto_start равным "1". Для кодирования данных сессии по умолчанию используется php. Устаревание данных, сохраненных в кэше, происходит через 180 минут.

Существует еще множество настроек, с которыми можно познакомиться в документации или непосредственно в файле настроек php.ini. На наш взгляд, знакомства с перечисленными выше параметрами достаточно для работы с сессиями в PHP. Так что приступим.

Создание сессии

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 authorize.php, 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).

Регистрация переменных сессии

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, логин 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']. 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 страничку secret_info.php, да и на любую другую страницу сайта, мы сможем работать с введенными

пользователем логином и паролем, которые будут храниться в массиве $_SESSION. Таким образом, если изменить код секретной странички (заметьте, мы переименовали ее в secret_info.php) 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>
```

То мы получим в браузере на секретной странице следующее:

Array ([login] => pit [passwd] => 123)
Здесь я хочу делиться секретами с другом Петей.

В итоге получим список переменных, зарегистрированных на authorize.php и, собственно, саму секретную страничку.

Что это нам дает? Допустим, хакер хочет прочитать секреты Васи и Пети. И он как-то узнал, как называется секретная страничка (или странички). Тогда он может попытаться просто ввести ее адрес в строке браузера, минуя страницу авторизации (ввода пароля). Чтобы избежать такого проникновения в наши тайны, нужно дописать всего пару строк в код секретных страничек:

```
<?php
session_start();
    // создаем новую сессию или
    // восстанавливаем текущую
print_r($_SESSION);
    // выводим все переменные сессии
if (!($_SESSION['login']=="pit" &&
    $_SESSION['passwd']==123))
```

```
        // проверяем правильность
        // пароля-логина
        Header("Location: authorize.php");
        // если ошибка, то перенаправляем на
        // страницу авторизации
    ?>
    <html>
    <head><title>Secret info</title></head>
    ... // здесь располагается
        //секретная информация :)
    </html>
```

Удаление переменных сессии

Кроме умения регистрировать переменные сессии (т.е. делать их глобальными на протяжении всего сеанса работы), полезно также уметь удалять такие переменные и сессию в целом.

Функция session_unregister(имя_переменной) удаляет глобальную переменную из текущей сессии (т.е. удаляет ее из списка зарегистрированных переменных). Если регистрация производилась с помощью $_SESSION, то используют языковую конструкцию unset(). Она не возвращает никакого значения, а просто уничтожает указанные переменные.

Где это может пригодиться? Например, для уничтожения данных о посетителе (в частности, логина и пароля) после его ухода с секретной странички. Если правильные логин и пароль сохранятся и окно браузера после посещения сайта не закрыли, то любой другой пользователь этого компьютера сможет прочитать закрытую информацию.

Для того чтобы сбросить значения всех переменных сессии, можно использовать функцию session_unset();

Уничтожить текущую сессию целиком можно командой session_destroy(); Она не сбрасывает значения глобальных переменных сессии и не удаляет cookies, а уничтожает все данные, ассоциируемые с текущей сессией.

```
    <?
    session_start(); // инициализируем сессию
    $test = "Переменная сессии";
    $_SESSION['test']= $test;
```

```
// регистрируем переменную $test.
// если register_globals=on,
// то можно использовать
// session_register('test');

print_r($_SESSION);
// выводим все глобальные переменные

echo session_id();
// выводим идентификатор сессии

echo "<hr>";
session_unset();
// уничтожаем все глобальные
// переменные сессии
print_r($_SESSION);
echo session_id();
echo "<hr>";
session_destroy(); // уничтожаем сессию
print_r($_SESSION);
echo session_id();
?>
```

В результате работы этого скрипта будут выведены три строки: в первой - массив с элементом test и его значением, а также идентификатор сессии, во второй - пустой массив и идентификатор сессии, в третьей - пустой массив. Таким образом, видно, что после уничтожения сессии уничтожается и ее идентификатор, и мы больше не можем ни регистрировать переменные, ни вообще производить какие-либо действия с сессией.

3.5.3 Безопасность

Вообще говоря, следует понимать, что использование механизма сессий не гарантирует полной безопасности системы. Для этого нужно принимать дополнительные меры. Обратим внимание на проблемы с безопасностью, которые могут возникнуть при работе с сессиями и, в частности, с теми программами, что мы написали.

Во-первых, опасно передавать туда-сюда пароль, его могут перехватить. Кроме того, мы зарегистрировали его как глобальную

переменную сессии, значит, он сохранился в cookies на компьютере-клиенте. Это тоже плохо. И вообще, пароли и логины по-хорошему должны храниться в базе данных. Пусть информация о пользователях хранится в базе данных "test" (в таблице "users"), а мы имеем к ней доступ под логином my_user и паролем my_passwd.

Во-вторых, что делать, если кто-то написал скрипт подбора пароля для секретной страницы? В этом случае на страницу авторизации много раз должен стучаться какой-то посторонний скрипт. Поэтому нужно просто проверять, с нашего ли сайта пришел запрос на авторизацию, и если нет, то не пускать его дальше. Адрес страницы, с которой поступил запрос, можно получить с помощью глобальной переменной $_SERVER['HTTP_REFERER']). Хотя, конечно, если за взлом сайта взялись всерьез, то значение этой переменной тоже подменят (например, с помощью того же PHP). Тем не менее проверку ее значения можно считать одним из важнейших шагов на пути к обеспечению безопасности своего сайта

Пусть у нас имеется файл авторизации.

```
<?
session_start();
    // создаем новую сессию или
    // восстанавливаем текущую
$conn = mysql_connect("localhost",
       "my_user","my_passwd");
    // устанавливаем соединение с сервером БД
mysql_select_db("test");
    // выбираем рабочую базу данных

$SERVER_ROOT = "http://localhost/~user/tasks/sessions/";
    // где находятся наши скрипты

/* с помощью регулярного выражения
^$SERVER_ROOT и функции eregi проверяем,
начинается ли адрес ссылающегося скрипта,
т.е. строка $_SERVER['HTTP_REFERER'])
со строки $SERVER_ROOT (как у нас) */

if(eregi("^$SERVER_ROOT",
         $_SERVER['HTTP_REFERER'])){
// если да, то делаем почти то же, что и
// раньше, пароль регистрировать не будем
```

```
    if (!isset($_POST['go'])){
        echo "<form method=POST >
            Login: <input type=text name=login>
            Password: <input type=password name=passwd>
            <input type=submit name=go value=Go>
        </form>";
    }else {
    /* запрос к базе данных: выбираем из таблицы
    users login, который совпадает с переданным
    по запросу, причем пароль у него тоже должен
    совпасть с введенным пользователем.
    Если этого нет, то считаем, что логин и
    пароль введены неверно */
    $sql = "SELECT login FROM users
      WHERE login='" . $_POST['login'] . "' AND passwd='" .
$_POST['passwd'] . "';";
    $q = mysql_query($sql,$conn); // отправляем запрос к БД
    $n = mysql_num_rows($q); // число строк в ответе на
запрос
    if (!$n==0){
        $_SESSION['user_login']=$_POST['login'];
        // регистрируем переменную login
        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
    }
    ?>
```

Вроде бы первые две проблемы решены. Но есть еще одна. Что делать, если хакер просто допишет в строку запроса значение какой-нибудь глобальной переменной (например, логина)? Вообще это возможно, только если register_globals=On. Просто иначе мы используем для работы с глобальными переменными массив $_SESSION и с ним такие фокусы не проходят. Все же попробуем решить и эту проблему. Для этого нужно очистить строку запроса перед тем, как сравнивать значения параметров. То есть сначала сбросим значение $user_login. Потом данную переменную нужно опять зарегистрировать, но не как новую, а как уже существующую. Для этого знак доллара при регистрации НЕ опускается. Вот что получилось:

```php
<?php
unset($user_login); // уничтожаем переменную
session_start(); // создаем новую сессию или
                 // восстанавливаем текущую
session_register($user_login);
    // регистрируем переменную
    // как уже существующую
if (!($user_login=="pit")) // проверяем логин
    Header("Location: authorize.php");
    // если ошибка, то перенаправляем
    // на страницу авторизации
?>
<html>
<head><title>Secret info</title></head>
... // здесь располагается
    // секретная информация :)
</html>
```

4 Работа с файловой системой

Как и большинство языков программирования, PHP поддерживает работу с файлами, которые являются одним из способов хранения информации. Напомним, что клиентские скриптовые языки не могут работать с файловой системой.

4.1 Чтение и запись файлов

Функция fopen

Вообще говоря, в PHP не существует функции, предназначенной именно для создания файлов. Большинство функций работают с уже существующими файлами в файловой системе сервера. Есть несколько функций, которые позволяют создавать временные файлы, или, что то же самое, файлы с уникальным для текущей директории именем. А вот для того, чтобы создать самый обычный файл, нужно воспользоваться функцией, которая открывает локальный или удаленный файл. Называется эта функция fopen(). Что значит «открывает файл»? Это значит, что fopen связывает данный файл с потоком управления программы. Причем связывание бывает различным в зависимости от того, что мы хотим делать с этим файлом: читать его, записывать в него данные или делать и то и другое. Синтаксис этой функции такой:

```
resource fopen ( имя_файла, тип_доступа
   [, use_include_path])
```

В результате работы эта функция возвращает указатель (типа ресурс) на открытый ею файл. В качестве параметров этой функции передаются: имя файла, который нужно открыть, тип доступа к файлу (определяется тем, что мы собираемся делать с ним) и, возможно, параметр, определяющий, искать ли указанный файл в include_path. Обсудим подробнее каждый из этих трех параметров.

Параметр имя_файла должен быть строкой, содержащей правильное локальное имя файла или URL-адрес файла в сети. Если имя файла начинается с указания протокола доступа (например, http://... или ftp://...), то интерпретатор считает это имя адресом URL и ищет обработчик указанного в URL протокола. Если обработчик найден, то PHP проверяет, разрешено ли работать с объектами URL как с обычными файлами (директива allow_url_fopen). Если allow_url_fopen=off, то функция fopen вызывает ошибку и генерируется предупреждение. Если имя файла не начинается с протокола, то считается, что указано имя локального файла. Чтобы открыть локальный файл, нужно, чтобы PHP имел соответствующие права доступа к этому файлу.

Параметр use_include_path, установленный в значение 1 или TRUE, заставляет интерпретатор искать указанный в fopen() файл в include_path. Напомним, что include_path – это директива из файла настроек PHP, задающая список директорий, в которых могут находиться файлы для включения. Кроме функции fopen() она используется функциями include() и require().

Значения, которые может принимать параметр тип_доступа, приведены в таблице 1.

Таблица 1 – Значения параметра тип_доступа

Значение параметра	Действия
r	Открывает файл только для чтения; устанавливает указатель позиции в файле на начало файла.
r+	Открывает файл для чтения и записи; устанавливает указатель файла на его начало.
w	Открывает файл только для записи; устанавливает указатель файла на его начало и усекает файл до нулевой длины. Если файл не существует, то пытается создать его.
w+	Открывает файл для чтения и записи; устанавливает указатель файла на его начало и усекает файл до нулевой длины. Если файл не существует, то пытается создать его.

Значение параметра	Действия
a	Открывает файл только для записи; устанавливает указатель файла в его конец. Если файл не существует, то пытается создать его.
a+	Открывает файл для чтения и записи; устанавливает указатель файла в его конец. Если файл не существует, то пытается создать его.
x	Создает и открывает файл только для записи; помещает указатель файла на его начало. Если файл уже существует, то `fopen()` возвращает `false` и генерируется предупреждение. Если файл не существует, то делается попытка создать его. Этот тип доступа поддерживается начиная с версии PHP 4.3.2 и работает только с локальными файлами.
x+	Создает и открывает файл для чтения и записи; помещает указатель файла на его начало. Если файл уже существует, то `fopen()` возвращает `false` и генерируется предупреждение. Если файл не существует, то делается попытка создать его. Этот тип доступа поддерживается, начиная с версии PHP 4.3.2, и работает только с локальными файлами.

Чтобы создать файл, нужно, как бы нелепо это ни звучало, открыть несуществующий файл на запись.

```
<?php
$h = fopen("my_file.html","w");
/* открывает на запись файл my_file.html,
если он существует, или создает пустой
файл с таким именем, если его еще нет */
$h = fopen("dir/another_file.txt","w+");
/* открывает на запись и чтение или создает
файл another_file.txt в директории dir */
$h = fopen(
  "http://www.server.ru/dir/file.php","r");
/* открывает на чтение файл, находящийся по
указанному адресу*/
```

```
?>
```

Создавая файл, нужно учитывать, под какой операционной системой вы работаете, и под какой ОС предположительно этот файл будет читаться. Дело в том, что разные операционные системы по-разному отмечают конец строки. В Unix-подобных ОС конец строки обозначается \n, в системах типа Windows - \r\n. Windows предлагает специальный флаг t для перевода символов конца строки систем типа Unix в свои символы конца строки. В противоположность этому существует флаг b, используемый чаще всего для бинарных файлов, благодаря которому такой трансляции не происходит. Использовать эти флаги можно, просто дописав их после последнего символа выбранного типа доступа к файлу. Например, открывая файл на чтение, вместо r следует использовать rt, чтобы перекодировать все символы конца строки в \r\n. Если не использовать флаг b при открытии бинарных файлов, то могут появляться ошибки, связанные с изменением содержимого файла. Из соображений переносимости программы на различные платформы рекомендуется всегда использовать флаг b при открытии файлов с помощью fopen().

Если открыть или создать файл с помощью fopen не удается, PHP генерирует предупреждение, а функция fopen возвращает как результат своей работы значение false. Такого рода предупреждения можно «подавить» (запретить) с помощью символа @.

Например, такая команда не выведет предупреждения, даже если открыть файл не удалось:

```
$h = @fopen("dir/another_file.txt","w+");
```

Таким образом, функция fopen() позволяет создать только лишь пустой файл и сделать его доступным для записи.

Функция fclose

После выполнения необходимых действий с файлом, будь то чтение или запись данных или что-либо другое, соединение,

установленное с этим файлом функцией fopen(), нужно закрыть. Для этого используют функцию fclose(). Синтаксис у нее следующий:

fclose (указатель на файл)

Эта функция возвращает TRUE, если соединение успешно закрыто, и FALSE - в противном случае. Параметр этой функции должен указывать на файл, успешно открытый, например, с помощью функции fopen().

```
<?php
$h = fopen("my_file.html","w");
fclose($h);
?>
```

Конечно, если не закрывать соединение с файлом, никаких ошибок выполнения скрипта не произойдет. Но в целом для сервера это может иметь серьезные последствия. Например, хакер может воспользоваться открытым соединением и записать в файл вирус, не говоря уже о лишней трате ресурсов сервера. Так что советуем всегда закрывать соединение с файлом после выполнения необходимых действий.

Функция fwrite

Для того чтобы записать данные в файл, доступ к которому открыт функцией fopen(), можно использовать функцию fwrite(). Синтаксис у нее следующий:

int fwrite (указатель на файл,
строка [, длина])

Эта функция записывает содержимое строки в файл, на который указывает указатель на файл. Если указан дополнительный аргумент длина, то запись заканчивается после того, как записано количество символов, равное значению этого аргумента, или когда будет

достигнут конец строки. В результате своей работы функция fwrite() возвращает число записанных байтов или false, в случае ошибки.

```
<?php
$h = fopen("my_file.html","w");
$text = "Этот текст запишем в файл.";
if (fwrite($h,$text))
  echo "Запись прошла успешно";
else
  echo "Произошла ошибка при записи данных";
fclose($h);
?>
```

В результате работы этого скрипта в браузере мы увидим сообщение о том, что запись прошла успешно, а в файле my_file.html появится строка "Этот текст запишем в файл.". Если бы этот файл существовал до того, как мы выполнили этот скрипт, все находящиеся в нем данные были бы удалены.

Если же мы напишем такой скрипт:

```
<?php
$h = fopen("my_file.html","a");
$add_text = "Добавим текст в файл.";
if(fwrite($h,$add_text,7))
  echo "Добавление текста прошло
    успешно<br>";
else echo "Произошла ошибка при
   добавлении данных<br>";
fclose($h);
?>
```

то к строке, уже существующей в файле my_file.html, добавится еще семь символов из строки, содержащейся в переменной $add_text, т.е. слово «Добавим».

Функция fwrite() имеет псевдоним fputs(), используемый таким же образом, что и сама функция.

Функция fread

Эта функция осуществляет чтение данных из файла. Ее можно использовать и для чтения данных из бинарных файлов, не опасаясь их повреждения. Синтаксис fread() такой:

string fread (указатель на файл, длина)

При вызове этой функции происходит чтение данных длины (в байтах), определенной параметром длина, из файла, на который указывает указатель на файл. Параметр указатель на файл должен быть реально существующей переменной типа ресурс, содержащей в себе связь с файлом, открытую, например, с помощью функции fopen(). Чтение данных происходит до тех пор, пока не встретится конец файла или пока не будет прочитано указанное параметром длина число байтов.

В результате работы функция fread() возвращает строку со считанной из файла информацией.

Как вы заметили, в этой функции параметр длина - обязательный. Следовательно, если мы хотим считать весь файл в строку, нужно знать его длину. PHP может самостоятельно вычислить длину указанного файла. Для этого нужно воспользоваться функцией filesize(имя файла). В случае ошибки эта функция вернет false. К сожалению, ее можно использовать только для получения размера локальных файлов.

```
<?php
$h = fopen("my_file.html","r+");
// открываем файл на запись и чтение
$content = fread($h,
     filesize("my_file.html"));
// считываем содержимое файла в строку
fclose($h); // закрываем соединение с файлом
echo $content;
// выводим содержимое файла
// на экран браузера
?>
```

Для того чтобы считать содержимое бинарного файла, например изображения, в таких системах, как Windows, рекомендуется

открывать файл с помощью флага rb или ему подобных, содержащих символ b в конце.

Функция filesize() кэширует результаты своей работы. Если изменить содержимое файла my_file.html и снова запустить приведенный выше скрипт, то результат его работы не изменится. Более того, если запустить скрипт, считывающий данные из этого файла с помощью другой функции (например, fgets), то результат может оказаться таким, как если бы файл не изменился. Чтобы этого избежать, нужно очистить статический кэш, добавив в код программы команду clearstatcache();.

Функция fgets

С помощью функции fgets() можно считать из файла строку текста. Синтаксис этой функции практически такой же, как и у fread(), за исключением того, что длину считываемой строки указывать необязательно:

string fgets (указатель на файл, [длина])

В результате работы функция fgets() возвращает строку длиной (длина – 1) байт из файла, на который указывает указатель на файл. Чтение заканчивается, если прочитано (длина – 1) символов или встретился символ перевода строки или конец файла. Напомним, что в PHP один символ – это один байт. Если длина считываемой строки не указана (данная возможность появилась начиная с PHP 4.2.0), то считывается 1 Кбайт (1024 байт) текста или, что то же самое, 1024 символа. Начиная с версии PHP 4.3, если параметр длина не задан, считывается строка целиком. В случае ошибки функция fgets() возвращает false. Для версий PHP начиная с 4.3 эта функция безопасна для двоичных файлов.

```
<?php
$h = fopen("my_file.html","r+");
$content = fgets($h,2);
// считает первый символ из
// первой строки файла my_file.html
```

```
fclose($h);
echo $content;
?>
```

Обе функции, fread() и fgets(), прекращают считывание данных из файла, если встречают конец файла. В PHP есть специальная функция, проверяющая, смотрит ли указатель позиции файла на конец файла. Это булева функция feof(), в качестве параметра которой передается указатель на соединение с файлом.

Например, вот так можно считать все строки файла my_file.html:

```
<?php
$h = fopen("my_file.html","r");
while (!feof ($h)) {
    $content = fgets($h);
    echo $content,"<br>";
}
fclose($h);
?>
```

Функция fgetss

Существует разновидность функции fgets() - функция fgetss(). Она тоже позволяет считывать строку из указанного файла, но при этом удаляет из него все встретившиеся html-теги, за исключением, быть может, некоторых. Синтаксис fgetss() такой:

string fgetss(указатель на файл,
  длина [, допустимые теги])

Обратите внимание, что здесь аргумент длина обязательный.

Пусть у нас имеется файл my_file.html следующего содержания:

```
<h1>Без труда не вынешь и рыбку из пруда.</h1>
<b>Тише едешь - дальше будешь</b> У семи нянек<i> дитя без глазу</i>.
```

Выведем на экран все строки файла my_file.html, удалив из них все теги, кроме <b> и <i>:

```php
<?php
$h = fopen("my_file.html","r");
while (!feof ($h)) {
    $content = fgetss($h,1024,'<b><i>');
   echo $content,"<br>";
}
fclose($h);
?>
```

В результате работы этого скрипта получим:
Без труда не вынешь и рыбку из пруда.
Тише едешь - дальше будешь У семи нянек *дитя без глазу*.

Функция fgetc

Естественно, если можно считывать информацию из файла построчно, то можно считывать ее и посимвольно. Для этого предназначена функция fgetc(). Легко догадаться, что синтаксис у нее следующий:

string fgetc (указатель на файл)

Эта функция возвращает символ из файла, на который ссылается указатель на файл, и значение, вычисляемое как FALSE, если встречен конец строки.

Вот так, например, можно считать файл по одному символу:

```php
<?php
$h = fopen("my_file.html","r");
while (!feof ($h)) {
    $content = fgetc($h);
    echo $content,"<br>";
}
fclose($h);
?>
```

На самом деле для того чтобы прочитать содержимое файла, открывать соединение с ним посредством функции fopen() совсем не обязательно. В PHP есть функции, которые позволяют делать это,

используя лишь имя файла. Это функции readfile( ), file( ) и file_get_contents( ). Рассмотрим каждую из них подробнее.

Функция readfile

Синтаксис:

int readfile (имя_файла
[, use_include_path])

Функция readfile() считывает файл, имя которого передано ей в качестве параметра имя_файла, и выводит его содержимое на экран. Если дополнительный аргумент use_include_path имеет значение TRUE, то поиск файла с заданным именем производится и по директориям, входящим в include_path.

В программу эта функция возвращает число считанных байтов (символов) файла, а в случае ошибки - FALSE. Сообщения об ошибке в этой функции можно подавить оператором @.

Пример 7.7. Следующий скрипт выведет на экран содержимое файла my_file1.html и размер этого файла, если он существует. В противном случае выведется наше сообщение об ошибке - строка "Error in readfile".

```
<?php
$n = @readfile ("my_file1.html");
/* выводит на экран содержимое файла и
записывает его размер в переменную $n */
if (!$n) echo "Error in readfile";
/* если функция readfile() выполнилась
с ошибкой, то $n=false и выводим
сообщение об ошибке */
else echo $n;
  // если ошибки не было, то выводим число
  // считанных символов
?>
```

С помощью функции readfile() можно читать содержимое удаленных файлов, указывая их URL-адрес в качестве имени файла, если эта опция не отключена в настройках сервера.

Сразу же выводить содержимое файла на экран не всегда удобно. Порой нужно записать информацию из файла в переменную, чтобы в дальнейшем произвести с ней какие-либо действия. Для этого можно использовать функцию file() или file_get_contents().

Функция file

Функция file() предназначена для считывания информации из файла в переменную типа массив. Синтаксис у нее такой же, как и у функции readfile(), за исключением того, что в результате работы она возвращает массив:

```
array file ( имя_файла
   [, use_include_path])
```

Что за массив возвращает эта функция? Каждый элемент данного массива является строкой в файле, информацию из которого мы считываем (его имя задано аргументом имя_файла). Символ новой строки тоже включается в каждый из элементов массива. В случае ошибки функция file(), как и все уже рассмотренные, возвращает false. Дополнительный аргумент use_include_path опять же определяет, искать или нет данный файл в директориях include_path. Открывать удаленные файлы с помощью этой функции тоже можно, если не запрещено сервером. Начиная с PHP 4.3 работа с бинарными файлами посредством этой функции стала безопасной.

Например, у нас имеется файл my_file.html следующего содержания:

```
<h1>Без труда не вынешь
   и рыбку из пруда.</h1>
<b>Тише едешь - дальше будешь</b>
```

Прочитаем его содержимое с помощью функции file():

```
<?php
$arr = file ("my_file.html");
```

```
foreach($arr as $i => $a) echo $i,": ",
    htmlspecialchars($a), "<br>";
?>
```

В результате на экран будет выведено следующее сообщение:

0: <h1>Без труда не вынешь
и рыбку из пруда.</h1>
1: Тише едешь - дальше будешь

Функция file_get_contents

В версиях PHP начиная с 4.3 появилась возможность считывать содержимое файла в строку. Делается это с помощью функции file_get_contents(). Как и две предыдущие функции, в качестве параметров она принимает значение имени файла и, возможно, указание искать его в директориях include_path. Для порядка все равно приведем ее синтаксис:

string file_get_contents (
имя_файла [, use_include_path])

Эта функция абсолютно идентична функции file(), только возвращает она содержимое файла в виде строки. Кроме того, она безопасна для обработки бинарных данных и может считывать информацию из удаленных файлов, если это не запрещено настройками сервера.

Функция file_exists

Синтаксис:

bool file_exists (имя файла или директории)

Функция file_exists() проверяет, существует ли файл или директория, имя которой передано ей в качестве аргумента. Если директория или файл в файловой системе сервера существует, то функция возвращает TRUE, в противном случае - FALSE. Результат работы этой функции кэшируется. Соответственно очистить кэш

можно, как уже отмечалось, с помощью функции clearstatcache(). Для нелокальных файлов использовать функцию file_exists() нельзя.

```
<?php
$filename = 'c:/users/files/my_file.html';
if (file_exists($filename)) {
  print "Файл <b>$filename</b> существует";
} else {
  print "Файл <b>$filename</b>
        НЕ существует";
}
?>
```

Функция is_writable

Если кроме проверки существования файла нужно узнать еще, разрешено ли записывать информацию в этот файл, следует использовать функцию is_writable() или ее псевдоним - функцию is_writeable().

Синтаксис:

bool is_writable (имя файла или директории)

Эта функция возвращает TRUE, если файл (или директория) существует и доступен для записи. Доступ к файлу осуществляется под той учетной записью пользователя, под которой работает сервер (чаще всего это пользователь nobody или www). Результаты работы функции is_writable кэшируются.

Функция is_readable

Если кроме проверки *существования файла* нужно узнать еще, разрешено ли читать информацию из него, нужно использовать функцию is_readable().

Синтаксис:

bool is_readable (имя файла)

Эта функция работает подобно функции is_writable().

```
<?php
$filename = 'c:/users/files/my_file.html';
```

```
if (is_readable($filename)) {
  print "Файл <b>$filename</b> существует
     и доступен для чтения";
} else {
  print "Файл <b>$filename</b>
     НЕ существует или
     НЕ доступен для чтения";
}
?>
```

Функция unlink

Последнее, что мы хотим изучить из действий над файлами, - это удаление файлов. Для того чтобы удалить файл с помощью языка PHP, нужно воспользоваться функцией unlink(). Синтаксис этой функции можно описать следующим образом:

bool unlink (имя_файла)

Данная функция удаляет файл, имеющий имя имя_файла, возвращает TRUE в случае успеха этой операции и FALSE - в случае ошибки. Чтобы удалить файл, нужно тоже иметь соответствующие права доступа к нему (например, доступа только на чтение для удаления файла недостаточно).

```
<?php
$filename = 'c:/users/files/my_file.html';
unlink($filename);
  // удаляем файл с именем
  // c:/users/files/my_file.html
?>
```

4.2 Загрузка файла на сервер

Теперь решим более сложную и часто возникающую на практике задачу загрузки файла на сервер. Первое, что нужно сделать, чтобы загрузить файл на сервер, это создать html-форму. Для того чтобы с помощью этой формы можно было загружать файлы, она

должна содержать атрибут enctype в теге form со значением multipart/form-data, а также элемент input типа file.

```
<form enctype="multipart/form-data"
    action="parse.php" method="post">
  <input type="hidden" name="MAX_FILE_SIZE"
    value="30000" />
  Загрузить файл: <input type="file"
    name="myfile" /><br>
  <input type="submit"
    value="Отправить файл" />
</form>
```

Заметим, что мы добавили в форме скрытое поле, которое содержит в себе максимальный допустимый размер загружаемого файла в байтах. При попытке загрузить файл, размер которого больше указанного в этом поле значения, будет зафиксирована ошибка. В браузере созданная нами форма будет выглядеть как строка для ввода текста с дополнительной кнопкой для выбора файла с локального диска (рисунок 4).

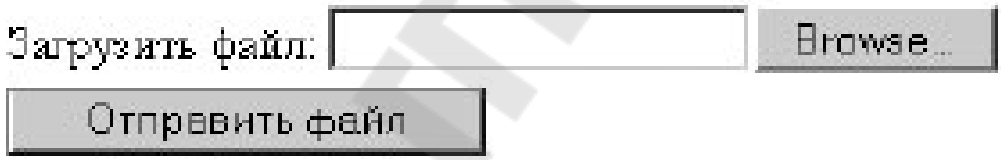


Рисунок 4 – Пример формы для загрузки файла на сервер

Теперь нужно написать скрипт, который будет обрабатывать полученный файл.

Вся информация о загруженном на сервер файле содержится в глобальном массиве $_FILES. Этот массив появился начиная с PHP 4.1.0. Если включена директива register_globals, то значения переданных переменных доступны просто по их именам.

Если мы загрузили с компьютера-клиента файл с именем critics.htm размером 15136 байт, то скрипт с единственной командой print_r($_FILES); выведет на экран следующее:

```
Array ( [myfile] =>
  Array ( [name] => critics.htm
    [type] => text/html
    [tmp_name] => C:\WINDOWS\TEMP\php49F.tmp
    [error] => 0
    [size] => 15136
    )
)
```

Вообще говоря, массив $_FILES всегда имеет следующие элементы:

$_FILES['myfile']['name'] - имя, которое имел файл на машине клиента.

$_FILES['myfile']['type'] - mime-тип отправленного файла, если браузер предоставил эту информацию. В нашем примере это text/html.

$_FILES['myfile']['size'] - размер загруженного файла в байтах.

$_FILES['myfile']['tmp_name'] - временное имя файла, под которым он был сохранен на сервере.

$_FILES['myfile']['error'] - код ошибки, появившейся при загрузке.

Здесь 'myfile' - это имя элемента формы, с помощью которого была произведена загрузка файла на сервер. То есть оно может быть другим, если элемент формы назвать иначе. Но вот другие ключи (name, type и т. д.) остаются неизменными для любой формы.

Если register_globals=On, то доступны также дополнительные переменные, такие как $myfile_name, которая эквивалентна $_FILES['myfile']['name'], и т.п.

Ошибок при загрузке в PHP выделяют пять типов и соответственно $_FILES['myfile']['error'] может иметь пять значений:

0 - ошибки не произошло, файл загружен успешно

1 - загружаемый файл превышает размер, установленный директивой upload_max_filesize в файле настроек php.ini

2 - загружаемый файл превышает размер, установленный элементом MAX_FILE_SIZE формы html

3 - файл был загружен частично

4 - файл загружен не был

По умолчанию загруженные файлы сохраняются во временной директории сервера, если другая директория не указана с помощью опции upload_tmp_dir в файле настроек php.ini. Переместить загруженный файл в нужную директорию можно с помощью функции move_uploaded_file().

Функция move_uploaded_file() имеет следующий синтаксис:

bool move_uploaded_file (временное_имя_файла,
место_назначения)

Эта функция проверяет, действительно ли файл, обозначенный строкой временное_имя_файла, был загружен через механизм загрузки HTTP методом POST. Если это так, то файл перемещается в файл, заданный параметром место_назначения (этот параметр содержит как путь к новой директории для хранения, так и новое имя файла).

Если временное_имя_файла задает неправильный загруженный файл, то никаких действий произведено не будет, и move_uploaded_file() вернет FALSE. То же самое произойдет, если файл по каким-то причинам не может быть перемещен. В этом случае интерпретатор выведет соответствующее предупреждение. Если файл, заданный параметром место_назначения, существует, то функция move_uploaded_file() перезапишет его.

```
<?
/* в версиях PHP, более ранних,
чем 4.1.0, вместо массива
$_FILES нужно использовать
массив $HTTP_POST_FILES */

$uploaddir = 'c:/uploads/';
    // будем сохранять загружаемые
    // файлы в эту директорию
$destination = $uploaddir.
     $_FILES['myfile']['name'];
    // имя файла оставим неизменным
```

```
print "<pre>";
if (move_uploaded_file(
    $_FILES['myfile']['tmp_name'],
    $destination)) {
/* перемещаем файл из временной папки
в выбранную директорию для хранения */

    print "Файл успешно загружен <br>";
} else {
  echo "Произошла ошибка при загрузке файла.
    Некоторая отладочная информация:<br>";
    print_r($_FILES);
}
print "</pre>";
?>
```

5 Объекты и классы в PHP

Начнем с основных понятий объектно-ориентированного программирования – класса и объекта. Существует множество определений этих понятий. Мы дадим следующее: объект – это структурированная переменная, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии, класс – это описание таких объектов и действий, которые можно с ними выполнять.

В PHP класс определяется с помощью следующего синтаксиса:

```
class Имя_класса{
  var $имя_свойства;
    /*список свойств*/
  function имя_метода( ){
   /* определение метода */
  }
  /*список методов*/
}
```

Имена свойств объектов класса объявляются с помощью ключевого слова var, методы, применимые к объектам данного класса, описываются функциями. Внутри определения класса можно использовать ключевое слово this для обращения к текущему представителю класса.

Например, нам нужно создать класс, описывающий категорию статей. У каждой статьи имеются такие свойства, как название, автор и краткое содержание. Какие действия мы хотим совершать со статьями? Возможно, нам понадобится задавать значения перечисленным свойствам статьи, отображать статью в браузере. Тогда определение этого класса может выглядеть следующим образом:

```
<?
```

```
class Articles { // Создаем класс Статей
  var $title;
  var $author;
  var $description;
// метод, который присваивает значения
// атрибутам класса
  function make_article($t, $a, $d){
    $this->title = $t;
    $this->author = $a;
    $this->description = $d;
  }
//метод для отображения экземпляров класса
  function show_article(){
    $art = $this->title . "<br>" .
           $this->description .
      "<br>Автор: " . $this->author;
    echo $art;
  }
}
?>
```

Итак, для описания физических объектов типа «статья» мы создали класс с именем Articles, состоящий из трех переменных, содержащих характеристики статьи, и двух функций для создания конкретной статьи и для ее отображения.

Как известно, работая с PHP, можно периодически переключаться в режим HTML. В этом случае программа состоит из нескольких кусков (блоков) кода. Определение класса нельзя разносить по разным блокам php-кода и тем более по разным файлам. То есть если написать:

```
<?php
class Articles { // Начало описания класса
  var $title;
?>
```

```
<?php
// продолжение описания класса
   function show_article(){
   // содержание метода
   }
} // конец описания класса
?>
```

то программа не будет работать корректно.

Несколько замечаний по поводу имен классов. Имя класса должно удовлетворять правилам именования объектов в языке PHP, но есть ряд имен, которые зарезервированы разработчиками для своих целей. В первую очередь это имена, начинающиеся с символа подчеркивания «_». Для создания классов и функций нельзя использовать такие имена. Кроме того, зарезервировано имя stdClass, поскольку оно используется внутри движка PHP.

Инициализация переменных

Часто некоторым атрибутам класса бывает необходимо присваивать значения сразу после создания представителя класса. Когда мы создавали класс статей, для присваивания значений атрибутам (свойствам) класса мы использовали специальную функцию make_article(). Вообще говоря, мы поступили не совсем верно, потому что занялись изобретением велосипеда. Специально для задания начальных значений атрибутам класса существует два стандартных метода. В PHP4 можно инициализировать значения с помощью оператора var или с помощью функции конструктора. С помощью var можно инициализировать только константные значения. Для задания не константных значений используют функцию конструктор, которая вызывается автоматически, когда объект конструируется из класса. Функция-конструктор должна иметь имя, совпадающее с именем всего класса, в котором она определена.

Приведем пример. Допустим, при создании объекта «статья» мы хотим установить его свойства следующим образом: автора – равным строке «Иванов», название и краткое содержание – соответствующим элементам глобального массива $_POST, а дату публикации статьи – текущей дате. Тогда следующее описание класса не является корректным в PHP:

```
<?
class Articles { // Создаем класс Статей
  var $title= $_POST["title"];
  var $author = "Иванов";
  var $description = $_POST["description"];
```

```
  var $published = date("Y-m-d");
// метод, который присваивает значения
// атрибутам класса
  }
?>
```

А вот такое описание класса в PHP будет работать так, как нужно:

```
<?
class Articles { // Создаем класс Статей
  var $title;
  var $author = "Иванов";
  var $description;
  var $published;
// метод, который присваивает значения
// атрибутам класса
 function Articles(){
  $this->title = $_POST["title"];
  $this->description = $_POST["description"];
  $this ->published = date("Y-m-d");
 }
}
?>
```

В PHP5 конструктор класса именуется __construct. Кроме того, в PHP5 появились и деструкторы – функции, которые вызываются автоматически перед уничтожением объекта. В PHP5 функция-деструктор должна быть названа __destruct.

Объекты

В одной из первых лекций мы упоминали о существовании в PHP такого типа данных, как объект. Класс – это описание данных одного типа, данных типа объект. Классы являются как бы шаблонами для реальных переменных. Переменная нужного типа создается из класса с помощью оператора new. Создав объект, мы можем применять к нему все методы и получать все свойства, определенные в описании класса. Для этого используют такой синтаксис: $имя_объекта->название_свойства или $имя_объекта-

>название_метода(список аргументов). Заметим, что перед названием свойства или метода знак $ не ставят.

```
<?php
$art = new Articles;
    // создаем объект $art
echo ($art ->title);
    // выводим название объекта $art
$another_art = new Articles;
    // создаем объект $another_art
$another_art->show_article();
    // вызываем метод для
    // отображения объекта в браузер
?>
```

Каждый из объектов класса имеет одни и те же свойства и методы. Так, у объекта $art и у объекта $another_art есть свойства title, description, author и методы Articles(), show_article(). Но это два разных объекта. Представим себе объект как директорию в файловой системе, а его характеристики – как файлы в этой директории. Очевидно, что в каждой директории могут лежать одинаковые файлы, но тем не менее они считаются различными, поскольку хранятся в разных директориях. Точно так же свойства и методы считаются различными, если они применяются к разным объектам. Чтобы получить нужный файл из директории верхнего уровня, мы пишем полный путь к этому файлу. При работе с классами нужно указывать полное имя функции, которую мы хотим вызвать. Директорией верхнего уровня в PHP будет пространство глобальных переменных, а путь указывается с помощью разделителя ->. Таким образом, имена $art->title и $another_art->title обозначают две разные переменные. Переменная в PHP имеет только один знак доллара перед именем, поэтому нельзя писать $art->$title. Эта конструкция будет рассмотрена не как обращение к свойству title объекта $art, а как обращение к свойству, имя которого задано переменной $title (например, $art->"").

```
<?php
$art->title = "Введение в Internet";
```

```
        // так можно установить
        // значение свойства объекта
    $art->$title = "Введение в Internet";
        // так нельзя установить
        // значение свойства объекта
    $property = "title";
    $art->$property = "Введение в Internet";
        // так можно установить значение
        // свойства объекта
    ?>
```

Создавая класс, мы не можем знать, какое имя будет иметь объект этого класса, тем более что объектов может быть много и все могут иметь разные имена. Соответственно мы не знаем, как обращаться к объекту внутри определения класса. Для того чтобы иметь доступ к функциям и переменным внутри определения класса, нужно использовать псевдопеременную $this. Например, $this->title возвращает значение свойства title у текущего объекта данного класса. Иногда эту переменную предлагают читать как «мое собственное» (к примеру, по отношению к свойству).

Наследование

Механизм наследования – очень важная часть всего объектно-ориентированного подхода. Попытаемся объяснить его суть на примере. Допустим, мы создаем описание человека. Очевидно, что сделать это мы можем по-разному, в зависимости от того, для чего нужно это описание. Можно описать человека как программиста: он знает такие-то языки программирования, операционные системы, участвовал в стольких-то проектах. Однако если человек программист, то он не перестает быть человеком вообще, т.е. он имеет имя, фамилию, место жительства и т.п. Если перевести наши рассуждения в термины объектно-ориентированного программирования, то можно сказать, что мы описали два класса – класс людей и класс программистов, каждый со своими свойствами и методами. Причем класс программистов, очевидно, обладает всеми свойствами класса людей и при этом имеет свои специфические характеристики, т.е. класс программистов является подклассом класса

людей. Так, если у человека вообще есть имя, то у программиста оно тоже должно быть, но не наоборот. Кроме программистов можно выделить еще множество классов по профессиональной принадлежности людей. И все они будут подклассами класса людей. Часто на практике удобно определять общий класс, который может использоваться сразу в нескольких проектах (например, класс людей или личностей), и адаптировать его для специфических нужд каждого проекта (например, как класс программистов). Как это можно реализовать? С помощью механизма расширений. Любой класс может быть расширением другого класса. Расширяющий (или производный) класс, кроме тех свойств и методов, которые описаны в его определении, имеет все функции и свойства основного (базового класса). В нашем примере класс программистов – расширяющий, а класс всех людей – базовый. Из класса нельзя удалить никакие существующие свойства и функции, класс можно только расширить. Расширяющий класс в PHP4 всегда зависит только от одного базового класса, поскольку множественное наследование в PHP не поддерживается. Расширяются классы в PHP с помощью ключевого слова extends.

```
<?php
class Person { // определяем класс Личности
  var $first_name; // имя личности
  var $last_name; // фамилия личности
  function make_person($t,$a){
  // метод устанавливает
  // значения имени и фамилии объекта
    $this->first_name = $t;
    $this->last_name = $a;
  }
  function show_person(){
  // метод отображает информацию о личности
    echo ("<h2>" . $this->first_name . " " .
      $this->last_name . "</h2>");
  }
}
class Programmer extends Person{
    // определяем класс
    // Programmer, расширяющий Person
  var $langs = array ("Lisp");
    // константным массивом
```

```
        // задать переменную в var можно
        function set_lang($new_lang){
        // метод добавляет еще
        // один язык к списку известных
        $this->langs[] = $new_lang;
    }
}
?>
```

Класс Programmer имеет те же переменные и функции, что и класс Person, плюс переменную $langs, в которой содержится список изученных программистом языков, и функцию set_lang для добавления еще одного языка к списку изученных. Создать представителя класса программистов можно обычным способом с помощью конструкции new. После этого можно устанавливать и получать список языков, которые знает программист, и в то же время можно использовать функции, заданные для класса Person, т.е. устанавливать и получать имя и фамилию программиста и отображать сведения о нем в браузере:

```
<?php
$progr = new Programmer;
$progr->set_lang("PHP");
    // методы, определенные для
    // класса Programmer
print_r ($progr->langs);
// методы, определенные для класса Person
$progr->make_person("Bill","Gates");
$progr->show_person();
?>
```

Отношения, в которых состоят созданные нами классы Person и Programmer, называют также отношениями родитель–потомок. Класс Person – родитель, а его потомки, такие как класс Programmer, создаются, основываясь на нем, с помощью расширений. Любой класс может стать родительским и соответственно породить потомков.

Порядок определения классов имеет значение. Нельзя сначала определить класс Programmer, расширяющий класс Person, а уже потом сам класс Person. Класс должен быть определен перед тем, как он будет использоваться (расширяться).

Оператор ::

Иногда внутри описания класса возникает необходимость сослаться на функции или переменные из базового класса. Бывает, что нужно ссылаться на функции в классе, ни один представитель которого еще не создан. Как быть в таком случае? В PHP4 для этого существует специальный оператор «::»

Например, вот так можно вызвать в описании класса Programmer функцию show_name() из базового класса Person и функцию say_hello(), заданную в описании класса Programmer, когда ни один объект этого класса еще не был создан:

```
<?php
class Person { // определяем класс Личности
  var $first_name;
  var $last_name;
  function Person($t,$a){ // конструктор
    $this->first_name = $t;
    $this->last_name = $a;
  }
function show_name(){
  // метод отображает информацию о личности
    echo ("Меня зовут, " .
          $this->first_name . " " .
          $this->last_name . "!<br>");
  }
}
class Programmer extends Person{
    // определяем класс
    // Programmer, расширяющий Person
  function set_lang($new_lang){
    // метод добавляет еще
    // один язык к списку известных
    $this->langs[] = $new_lang;
    Person::show_name();
    // вызываем функцию из базового класса
    echo "И я знаю теперь еще и " .
          $new_lang;
  }
  function show_name(){
  echo ("Я программист, " .
        $this->first_name . " " .
        $this->last_name . "!<br>");
  }
  function say_hello(){
    echo "Привет!<br>";
```

```
    }
}
Programmer::say_hello();
    // вызываем функцию, когда ни
    // один объект ее класса еще не создан
$new_progr = new Programmer("Вася","Сидоров");
$new_progr->set_lang("PHP");
?>
```

В результате работы этой программы получим следующее:
Привет!
Меня зовут Вася Сидоров!
И я знаю теперь еще и PHP

С помощью команды Programmer::say_hello(); мы вызываем функцию say_hello класса Programmer как таковую, а не как метод, применяемый к объекту данного класса. В этот момент переменных класса нет. Поэтому функции, вызываемые до создания объекта, не могут пользоваться переменными класса и конструкцией this, но могут пользоваться локальными и глобальными переменными.

В определении класса Programmer мы переопределили функцию show_name(), поэтому вызвать функцию show_name() из базового класса Person можно только с помощью оператора «::» Вообще говоря, внутри определения класса мы можем вызывать любые методы и свойства, заданные в его базовом классе с помощью обычного $this, если только порожденный класс не переопределяет эти свойства и методы, как в нашем примере.

Оператор parent

В приведенном выше примере, обращаясь к базовому классу, мы использовали его имя (мы писали Person::show_name()). Это не совсем удобно, потому что имя класса или иерархия классов может измениться, и тогда придется переписывать код описаний всех классов с тем, чтобы привести используемые в них имена в соответствие с новой иерархией. Чтобы избежать подобной ситуации, вместо имени базового класса нужно использовать ключевое слово parent (например, parent::show_name()). Parent ссылается на класс,

прописанный после extends в объявлении вашего класса. Поэтому если вдруг иерархия классов изменится, то достаточно будет внести изменения в имена, указанные после extends в описаниях классов.

Объектная модель PHP5

Кроме нового названия для конструкторов и появления деструкторов в PHP5 произошло еще достаточно много изменений. Мы не будем обсуждать их подробно, только опишем в общих чертах. Основное изменение – это передача значений параметров класса по ссылке и присвоение объектов по ссылке, а не по значению, как это было в PHP4. В PHP5 если создаются две равные переменные типа объект, то они указывают на одно значение и изменяются одновременно (мы приводили похожий пример с переменными строкового типа). В связи с этим появился новый механизм для создания копий объектов – так называемое клонирование. В PHP4 все методы и переменные класса доступны извне, т.е. они всегда являются открытыми. В PHP5 переменные и методы можно делать открытыми (доступными отовсюду), закрытыми (доступными только внутри класса) и защищенными (доступными внутри класса и в его производных классах). Кроме того, появилась возможность создавать интерфейсы и абстрактные классы и многое другое. В целом объектная модель в PHP5 значительно усовершенствована для более точного соответствия объектно-ориентированной парадигме программирования.

Итак, мы хотели по выбору пользователя генерировать форму для ввода описания статьи или человека и отображать данные, введенные в эту форму. Попробуем решить эту задачу, используя объектно-ориентированный подход. Для начала создадим форму, где пользователь выбирает, что он хочет создать, – описание статьи или человека (точнее, это будут две формы):

```
<form action="task1.php">
Создать описание статьи: <input type=submit
    name=art_create
    value="Create Article">
</form>
```

```
<form action="task1.php">
Создать описание личности: <input
   type=submit name=pers_create
   value="Create Person">
</form>
```

Теперь напишем файл для обработки этих форм. В нем создадим два класса – статьи и личности. У каждого класса имеется метод для инициализации его переменных и метод для отображения объектов данного класса. При решении задачи будут использованы две функции, встроенные в PHP для работы с классами и объектами. Это функция get_class(объект), возвращающая имя класса, экземпляром которого является объект, переданный ей в качестве параметра. И функция get_class_vars(имя класса), которая возвращает массив всех свойств класса и их значений по умолчанию. Аналогично можно получить массив имен всех методов класса: get_class_methods (имя класса)

```
<?php
// Создаем классы Статей и Личностей.
// Статья имеет заголовок, автора и
// описание. Личность имеет имя, фамилию
// и e-mail
class Article {
  var $title;
  var $author;
  var $description;
// метод, который присваивает значения
// атрибутам класса
  function Article($t="Название отсутствует",
      $a="Автор отсутствует",
      $d="Описание отсутствует"){
    $this->title = $t;
    $this->author = $a;
    $this->description = $d;
  }
//метод для отображения экземпляров класса
  function show(){
   $art = "<h2>$this->title</h2><font
   size=-1>$this->description</font><p>Автор:
   $this->author</p>";
    echo $art;
  }
}
```

```
    // Определение класса Личностей
    class Person {
      var $first_name;
      var $last_name;
      var $email;
    //метод, который присваивает значения атрибутам класса
      function Person($t="Имя не введено",
        $a="Фамилия не введена",$d="Email не указан"){
        $this->first_name = $t;
        $this->last_name = $a;
        $this->email = $d;
      }
    //метод для отображения экземпляров класса
      function show(){
        $art = "<h2>$this->first_name</h2><font
          size=-1>$this->last_name</font><p>e-mail:
    $this->email</p>";
        echo $art;
      }
    }
    // Далее следует собственно создание и отображение
    // экземпляров выбранного класса
    if (isset($_GET["art_create"])){ //Если была выбрана
статья
      $art = new Article; // создаем представителя класса
статей
      $art_vars = get_class_vars(get_class($art)); //какие
                          // аргументы этого класса нужно
задать
      Make_form($art,$art_vars,"art_create"); //вызов функции
                          // создания формы
      if (isset($_GET["create_real"])){ Show_($art_vars); }
      // если данные этой формы отправлены, то вызываем
      // функцию показа
    }
    //то же самое, если была выбрана личность
    if (isset($_GET["pers_create"])){
      $art = new Person;
      $art_vars = get_class_vars(get_class($art));
      Make_form($art,$art_vars,"pers_create");
      if (isset($_GET["create_real"])){ Show_($art_vars); }
    }
    // функция создания формы
    function Make_form($art,$art_vars,$glob){
      $str = "<form>";   // html код формы записывается
                         // в строку $str
      //перебираем список переменных класса объекта $art
      foreach ($art_vars as $var_name => $var_value){
```

```
        $str .="$var_name<input type=text
name=$var_name><br>";
        //создаем элемент формы с именем свойства класса
      }
      $str .= "<input type=hidden name=$glob>"; // чтобы не
                                        // забыть, что мы
создаем
      $str .= "<input type=submit name=create_real
        value='Create and Show'></form>";
    echo "$str";   // выводим форму
    }
    // функция показа объекта
    function Show_($art_vars){
      global $art; //используется глобальное имя объекта
      $k = count($art_vars); //число свойств класса
                             // (переменных в форме)
      $p=0; //вспомогательная переменная
      foreach ($art_vars as $name => $value){
        $p++;
          if ($_GET["$name"]=="") $val= $art->$name;
          else $val = $_GET["$name"];
          if ($p<>$k) $par .="'". $val."',";
          else $par .="'". $val."'";
      }
      $const=get_class($art);
      $par = '$art->'.$const ."(" .$par.");";
      // теперь $par представляет собой php-код для вызова
      // метода класса $art, изначально
      // записанного в $par
      // например,
      // $art->Person('Vasia','Petrov','vas@server.ru');
      eval($par); // функция eval выполняет код,
                  // содержащийся в $par
      $art->show();
    }
    ?>
```

6 Работа с базой данных. MySQL

Соединение с СУБД MySQL

Прежде чем получить доступ к данным необходимо соединиться с сервером базы данных.

Существует два способа: объектно-ориентированный и процедурный.

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Создание соединения
$conn = new mysqli($servername, $username, $password);

// Проверка соединения
if ($conn->connect_error) {
   die("Соединение отсутствует: " . $conn->connect_error);
}
echo "Соединение установлего";
?>
```

Для совместимости с более старыми версиями PHP используется процедурный подход:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Создание соединения
$conn = mysqli_connect($servername, $username, $password);
```

```
// Проверка соединения
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

После работы с СУБД необходимо закрыть соединение с БД

```
$conn->close();
```

или

```
mysqli_close($conn);
```

Работа с СУБД

Создание БД

ООП

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

// Create database
```

```
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
   echo "Database created successfully";
} else {
   echo "Error creating database: " . $conn->error;
}

$conn->close();
?>
```

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
   echo "Database created successfully";
} else {
   echo "Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Для манипулирование данными используется запрос на языке SQL к БД, предварительно указав с какой БД работаем.

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
```

Или

```
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}
```

Выполнение запроса

```
// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
```

```
email VARCHAR(50),
reg_date TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
   echo "Table MyGuests created successfully";
} else {
   echo "Error creating table: " . $conn->error;
}
```

или

```
// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if (mysqli_query($conn, $sql)) {
   echo "Table MyGuests created successfully";
} else {
   echo "Error creating table: " . mysqli_error($conn);
}
```

Выборка данных

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
```

```
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
   // output data of each row
   while($row = $result->fetch_assoc()) {
      echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
   }
} else {
   echo "0 results";
}
$conn->close();
?>
```

или

```
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}
```

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0) {
   // output data of each row
   while($row = mysqli_fetch_assoc($result)) {
      echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
$row["lastname"]. "<br>";
   }
} else {
   echo "0 results";
}
```

Список использованных источников

Web Technologies: HTML, Java Script, PHP, Java, JSP, XML and AJAX = Сетевые технологии: HTML, Java Script, PHP, Java, JSP, XML and AJAX. - Delhi : Dreamtech Press, 2010. - 1354 p

Дунаев, В. В. Сценарии для Web-сайта. PHP и Javascript / Вадим Дунаев. - Санкт-Петербург : БХВ-Петербург, 2006. - 555 с.

Котеров Д.В. PHP 5. - Санкт-Петербург : БХВ-Петербург, 2005. - 1120с.

Маклафлин, Б. PHP и MySQL : исчерпывающее руководство / Бретт Маклафлин ; [перевел с англ. О. Сивченко]. - 2-е изд.. - Санкт-Петербург [и др.] : Питер, 2014. - 543 с.

СЕРВЕРНЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ WEB-САЙТОВ

Пособие
по одноименной дисциплине
для слушателей специальности переподготовки
1-40 01 74 «Web-дизайн и компьютерная графика»
заочной формы обучения

Составитель **Самовендюк** Николай Владимирович

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 22.02.19.
Рег. № 9Е.
http://www.gstu.by