Программирование на стороне сервера. PHP.

Возможности PHP

В первую очередь PHP (*Personal Home Page, а позднее Hypertext Preprocessor*) используется для создания скриптов, работающих на стороне сервера. PHP способен решать те же задачи, что и любые другие CGI-скрипты, в том числе, обрабатывать данные html-форм, динамически генерировать HTML-страницы и т.п.

Существует две основные области применения PHP:

•**Первая область** – создание приложений (скриптов), которые исполняются на стороне сервера. PHP наиболее широко используется именно для создания такого рода скриптов. В этом случае PHP-парсер (т.е. обработчик php-скриптов) может работать как модуль web-сервера или как CGI-программа на Web-сервере.

•**Вторая область** – это создание скриптов, выполняющихся в командной строке. То есть с помощью PHP можно создавать такие скрипты, которые будут исполняться, вне зависимости от web-сервера и браузера, на конкретной машине. Для такой работы потребуется лишь парсер PHP (в этом случае его называют интерпретатором командной строки (cli, command line interpreter)). Этот вариант подходит, например, для скриптов, которые должны выполняться регулярно с помощью различных планировщиков задач или для решения задач простой обработки текста.

Краткая историческая справка

В **1994 году** датский программист Расмус Лердорф создал набор скриптов на Perl/CGI для вывода и учёта посетителей его онлайн-резюме, обрабатывающий шаблоны HTML-документов. Лердорф назвал набор *Personal Home Page* (Личная Домашняя Страница). Вскоре функциональности и быстроты Perl — интерпретатора скриптов — перестало хватать, и Лердорф разработал с использованием языка C новый интерпретатор шаблонов PHP/FI (англ. Personal Home Page / Forms Interpreter — «Личная Домашняя Страница / Интерпретатор форм»).

В 1997 году после длительного бета-тестирования вышла вторая версия обработчика, написанного на C — PHP/FI 2.0. Её использовали около 1 % (приблизительно 50 тысяч) всех интернет-доменов мира.

Версия **PHP 3.0** подверглась значительной переработке, определившим современный облик и стиль языка программирования. В 1997 году два израильских программиста, Энди Гутманс и Зээв Сураски, полностью переписали код интерпретатора. PHP 3.0 был официально выпущен в июне 1998 года.

Одной из сильнейших сторон PHP 3.0 была *возможность расширения ядра дополнительными модулями*. Впоследствии интерфейс написания расширений привлёк к PHP множество сторонних разработчиков, работающих над своими модулями, что дало PHP возможность работать с огромным количеством баз данных, протоколов, поддерживать большое число API. Большое количество разработчиков привело к быстрому развитию языка и стремительному росту его популярности. Также язык был переименован в PHP.

К зиме 1998 года, практически сразу после официального выхода PHP 3.0, Энди Гутманс и Зээв Сураски начали переработку ядра PHP. В задачи входило увеличение производительности сложных приложений и улучшение модульности базиса кода PHP. Новый движок, названный Zend Engine, успешно справлялся с поставленными задачами и впервые был представлен в середине 1999 года. **PHP 4.0**, основанный на этом движке и принёсший с собой набор дополнительных функций, официально вышел в мае 2000 года.

Пятая версия PHP была выпущена разработчиками 13 июля 2004 года. Изменения включают обновление ядра Zend (Zend Engine 2), что существенно увеличило эффективность интерпретатора. Введена поддержка языка разметки XML. Полностью переработаны функции ООП, которые стали во многом схожи с моделью, используемой в Java. В частности, введён деструктор, открытые, закрытые и защищённые члены и методы, окончательные члены и методы, интерфейсы и клонирование объектов. В последующих версиях также были введены пространства имён и замыкания.

Шестая версия PHP находится *в стадии разработки* с октября 2006 года. В ней уже сделано множество нововведений, как, например, исключение из ядра регулярных выражений POSIX и «длинных» суперглобальных массивов, удаление директив safe_mode, magic_quotes_gpc и register_globals из конфигурационного файла php.ini.

Основной синтаксис PHP

Для размещения кода PHP внутри HTML страницы используется пара специальных тегов *<?php ?>*(или сокращенный вариант *<? ?>*).

1)

```
<html>
  <head>
  <title>Пример</title>
  </head>
  <body>
  <?php
 echo "<p> Строка выводимая
PHP!</p>";
  ?>
</body>
</html>
```

2)

```
<html>
   <head>
   <title>Пример</title>
   </head>
   <body>
   <p> Строка выводимая PHP!</p>
   </body>
</html>
```

Разделение инструкций

Программа на PHP (да и на любом другом языке программирования) – 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
echo "Hello, world!"; // точка с
запятой
                // в конце команды
                // обязательна
?>
```

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

Комментарии

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
echo "Выводимая строка";
  // Это однострочный комментарий
  // в стиле C++
echo "Вторая строка";
/* Это многострочный комментарий.
Здесь можно написать несколько строк.
*/
echo"И еще одна строка";
  # Это комментарий в стиле
  # оболочки Unix
?>
```

Переменные

•Переменная в PHP обозначается знаком доллара, за которым следует ее имя.

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 значение оригинального выражения копируется 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 "Переменая с именем first равна $first <br>";
      // выводим значение $first
echo "Переменая 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 появилась возможность присвоения значений переменным: присвоение по ссылке. Для этого 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 амперсант &.

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

Результат работы этого скрипта будет следующим:

Переменная с именем first равна New text.
Переменная с именем second равна New text.

Константы

Для хранения постоянных величин, т.е. таких величин, значение которых не меняется в ходе выполнения скрипта, используются константы.

У константы нет приставки в виде знака доллара и ее нельзя определить простым присваиванием значения.

Для определения констант в PHP существует специальная функция define().
Ее синтаксис:
define("Имя_константы","Значение_константы",
[Нечувствительность_к_регистру])

По умолчанию имена констант чувствительны к регистру. Для каждой константы это свойство 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);
// выведет 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 */
// выведет 3.14, поскольку константа PI
// регистронезависима по определению
echo pi;
?>
```

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.

Рассмотрим некоторые из них подробнее.

Арифметические операторы		
Обозначение	**Название**	**Пример**
+	Сложение	$a + $b
-	Вычитание	$a - $b
*	Умножение	$a * $b
/	Деление	$a / $b
%	Остаток от деления	$a % $b

Строковые операторы		
Обозначение	**Название**	**Пример**
.	Конкатенация (сложение строк)	$c = $a . $b (это строка, состоящая из $a и $b)

Операторы присваивания			
Обозначение	**Название**	**Описание**	**Пример**
=	Присваивание	Переменной слева от оператора будет присвоено значение, полученное в результате выполнения каких-либо операций или переменной/константы с правой стороны	$a = ($b = 4) +5; ($a будет равна 9, $b 4-м)
+=		Сокращение. Прибавляет к переменной число и затем присваивает ей полученное значение	$a += 5; (эквивалентно $a = $a + 5;)
.=		Сокращенно обозначает комбинацию операций конкатенации и присваивания (сначала добавляется строка, потом полученная строка записывается в переменную)	$b = “Привет”; $b .= “всем”; (эквивалентно $b = $b . “всем”;) В результате: $b=“Привет всем”

Логические операторы			
Обозначение	**Название**	**Описание**	**Пример**
and	И	$a и $b истинны (True)	$a and $b
&&	И		$a && $b
or	Или	Хотя бы одна из переменных $a или $b истинна (возможно, что и обе)	$a or $b
||	Или		$a || $b
xor	Исключающее или	Одна из переменных истинна. Случай, когда они обе истинны, исключается	$a xor $b
!	Инверсия (NOT)	Если $a=True, то !$a=False и наоборот	! $a

Операторы сравнения			
Обозначение	**Название**	**Пример**	**Описание**
==	Равенство	Значения переменных равны	$a == $b
===	Эквивалентность	Равны значения и типы переменных	$a === $b
!=	Неравенство	Значения переменных не равны	$a != $b
<>	Неравенство		$a <> $b
!==	Неэквивалентность	Переменные не эквивалентны	$a !== $b
<	Меньше		$a < $b
>	Больше		$a > $b
<=	Меньше или равно		$a <= $b
>=	Больше или равно		$a >= $b

Операторы инкремента и декремента			
Обозначение	**Название**	**Описание**	**Пример**
++$a	Пре-инкремент	Увеличивает $a на единицу и возвращает $a	*<?* *$a=4;* *echo "Должно быть 4:"* *.$a++;* *echo "Должно быть 5:"* *.$a;* *?>*
$a++	Пост-инкремент	Возвращает $a, затем увеличивает $a на единицу	
--$a	Пре-декремент	Уменьшает $a на единицу и возвращает $a	
$a--	Пост-декремент	Возвращает $a, затем уменьшает $a на единицу	

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.

Логический тип данных (boolean)

Этот простейший тип выражает истинность значения, то есть переменная этого типа может иметь только два значения – истина TRUE или ложь FALSE.
Используемые служебные слова TRUE или FALSE регистронезависимы.

```
<?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 $action со строкой
   "Изучить PHP". Если совпадает, то   возвращает true, иначе – false.
   Если возвращено true, то выполняется   то, что внутри фигурных скобок */
if ($action == "Изучить PHP")
{ echo"Начал изучать"; }
?>
```

Целочисленный тип данных (integer)

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;
?>
```

Правила работы с целыми

- Размер целого зависит от платформы, хотя, как правило, максимальное значение - 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 $a=0.5 к целому типу необходимо написать (integer)($a) или a или использовать сокращенную запись (int)($a). Возможность явного приведения типов по такому принципу существует не для всех типов данных (не всегда значение одного типа можно перевести в другой тип).

Числа с плавающей точкой (float)

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.

Строки (string)

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. В отличие от двух других синтаксисов, переменные и управляющие последовательности для специальных символов, встречающиеся в строках, заключенных в одинарные кавычки, не обрабатываются.

Пример работы со строками, сформированными при помощи одинарных кавычек:

```
<?php
echo 'Также вы можете вставлять в строки
    символ новой строки таким образом,
    поскольку это нормально';
echo 'Чтобы вывести \' надо перед ней поставить \\’
// Выведет: Чтобы вывести ' надо перед ней поставить \
echo 'Вы хотите удалить C:\\*.*?';
// Выведет: Вы хотите удалить C:\*.*?
echo 'Это не вставит: \n новую строку';
// Выведет: Это не вставит: \n новую строку
echo 'Переменные $expand , $either не подставляются';
// Выведет: Переменные $expand, $either не подставляются
?>
```

Двойные кавычки

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

Последовательность	**Значение**
\n	Новая строка (LF или 0x0A (10) в ASCII)
\r	Возврат каретки (CR или 0x0D (13) в ASCII)
\t	Горизонтальная табуляция (HT или 0x09 (9) в ASCII)
\\	Обратная косая черта
\$	Знак доллара
\"	Двойная кавычка

Самым важным свойством строк в двойных кавычках является обработка переменных, т.е. вместо имени переменной будет выведено ее значение:

```
<?php
$expand=1;
$either=’abc’;
 echo "Переменные $expand , $either  подставляются";
// Выведет: Переменные 1, abc подставляются
?>
```

Heredoc-синтаксис

Другой способ определения строк – это использование heredoc-синтаксиса. Строка должна начинаться с символа <<<, после которого идет идентификатор. Заканчивается строка этим же идентификатором. Закрывающий идентификатор должен начинаться в первом столбце строки.

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.

Замечание: Поддержка heredoc была добавлена в PHP 4.

Пример работы со строками, сформированными при помощи heredoc-синтаксиса:

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

Массивы (array)

Массив в PHP представляет собой упорядоченную хеш-таблицу – тип, который оперирует со значениями и ключами. Этот тип оптимизирован в нескольких направлениях, поэтому может быть использован как, собственно массив, список (вектор), хеш-таблица, стэк, очередь и т.д.

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

Определение массива при помощи служебного слова array

Языковая конструкция array() принимает в качестве параметров пары *ключ => значение*, разделенные запятыми. Символ => устанавливает соответствие между значением и его ключом:

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

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
$books = array ("php" =>
                "PHP users guide",
                12 => true);
echo $books["php"];
//выведет "PHP users guide"
echo $books[12];     //выведет 1
?>
```

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 это значение будет перезаписано.

Начиная с PHP 4.3.0, если максимальный ключ – отрицательное число, то следующим ключом массива будет ноль (0).

```
<?php
// массивы $arr и $arr1 эквиваленты
$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.

Определение массива при помощи квадратных скобок

Создать массив можно, просто записывая в него значения. 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
?>
```

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)

Управляющие конструкции PHP

PHP как и любой другой язык программирования содержит такие управляющие конструкции как условные операторы, циклы, операторы включения(include).

PHP предлагает как традиционный(C-подобный) так и 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.

Условные операторы

Оператор if

Один из самых важных операторов многих языков, включая PHP, – оператор if.

Обобщенный вид оператора if представлен ниже:

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

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:

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;**
- **Специальный тип null;**

Все остальные значения преобразуются в TRUE.

Пример использования* конструкций *else и elseif, с альтернативным синтаксисом:

```
<?php
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

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
}
```

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

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 do... while

Циклы do..while очень похожи на циклы while, но блок_выполнения цикла do...while гарантированно выполняется хотя бы один раз.

Структура:

do {блок_выполнения} while (выражение);

Цикл for

Структура:

for (выражение1; выражение2; выражение3) {блок_выполнения}

либо

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

Каждое из выражений 1, 2, 3 аналогично соответствующему в цикле for языка С.

Цикл foreach

Конструкция foreach появилась только в PHP4 и предназначена исключительно для работы с массивами.

Синтаксис:

foreach ($array as $value) {блок_выполнения}

либо

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

```
<?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>";
}
?>
```

Операторы передачи управления

Иногда в случае особых обстоятельств требуется немедленно завершить работу цикла и передать управление первой инструкции программы, расположенной за последней инструкцией цикла. Для этого в PHP как и в C используют операторы **break** и **continue**.

Операторы включения

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");

require

Этот оператор действует примерно так же, как и #include в C++. 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.

Функции в PHP

Функция, определяемая пользователем, в PHP может быть описана с помощью следующего синтаксиса:

```
function Имя_функции (параметр1, параметр2,  … параметрN){
  Блок операторов
  return "значение возвращаемое функцией";
}
```

Аргументы функций

У каждой функции может быть список аргументов. Каждый аргумент представляет собой переменную или константу.

С помощью аргументов данные в функцию можно передавать тремя различными способами:

- ✓ **по значению** (используется по умолчанию),
- ✓ **по ссылке**
- ✓ **задание значения аргументов по умолчанию**.

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 амперсант «&».

```
<?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>"; //выведет уже отмеченную радио кнопку(с атрибутом checked)
?>
```

```
<?php
function Message($firstname="Вася",$secondname="Петров"){
  echo "$firstname $secondname";
}
Message();  // вызываем функцию без параметра.
   // В этом случае функция выведет – Вася Петров
Message("Петя","Иванов"); // В этом случае функция выведет -  Петя Иванов
?>
```

Списки аргументов переменной длины

В PHP (как и в JavaScript см. л.р. №3) можно создавать функции с переменным числом аргументов. В этом случае в PHP для работы с аргументами используются встроенные функции: **func_num_args(), func_get_arg(), func_get_args().**

```
<?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 (номер_аргумента)*** возвращает аргумент из списка переданных в функцию аргументов, порядковый номер которого задан параметром номер_аргумента. Аргументы функции считаются начиная с нуля.

Функция ***func_get_args()*** возвращает массив, состоящий из списка аргументов, переданных функции. Каждый элемент массива соответствует аргументу, переданному функции.

Использование переменных внутри функции

Глобальные переменные

Чтобы использовать внутри функции переменные, заданные вне нее, 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;//выведет - вне функции $a=1
echo "<br>";
Test_g();//выведет - в результате работы функции $a=2
echo "<br>";
echo 'вне функции $a='.$a;//выведет - вне функции $a=2
echo "<br>";
Test_g();//выведет - в результате работы функции $a=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 утверждения ***return***.

```
<?php
function Full_age($b_day, $b_month, $b_year){
  if ((date("m")>$b_month) && (date("d")>$b_day)) {
    $day = date ("d") – $b_day;
    $month = date("m") – $b_month;
    $year = date("Y") – $b_year;
  }
 else {
    $year = date("Y") – $b_year – 1;
    $day = 31 – $b_day – date ("d");
    $month = 12 – ($b_month – date("m"));
  }
return array ($day,$month,$year);
}
$age = Full_age("07","08","1974");
echo "Вам $age[2] лет, $age[1] месяцев  и $age[0] дней";
// выведет "Вам 29 лет, 11 месяцев и 5 дней"
?>
```

Возвращение по ссылке

Возвращение по ссылке используется в тех случаях, когда необходимо использовать функцию для выбора переменной(чаще всего глобальной), с которой должна быть связана данная ссылка. При возвращении по ссылке используется следующий синтаксис:

```
<?php
function &find_var($param)
{
    /* ... код ... */
    return $found_var;
}

$foo =& find_var($bar);
$foo = 2;
?>
```

В этом примере устанавливается свойство самой переменной, возвращённой функцией *find_var*, а не ее копии, как было бы без использования ссылки.
Важно использование **&** и в заголовке функции и при ее вызове.

Пример

Объектно-ориентированное программирование в PHP

В PHP 4 объектно-ориентированное программирование на PHP было возможно в минимальной степени. Поэтому в PHP 5 в первую очередь переработке подвергся весь механизм работы с объектами.

Основное отличие обработки объектов в PHP 5 от PHP 4 заключается в том, что теперь ***присвоение объекта или его передача в качестве параметра функции происходит по умолчанию по ссылке***, а не по значению, как в предыдущей версии.

И если в PHP 4 объекты обрабатывались также как и простые типы данных, что часто приводило к появлению нескольких копий одного и того же объекта, то в PHP 5 такого не происходит, так как каждый объект получает свой собственный числовой идентификатор (handle), который и используется при обращении к объекту.

Представленный ниже код, выполненный в PHP 4 и в PHP 5, очевидно может продемонстрировать различия в обработке объектов.

Объектно-ориентированное программирование в PHP

```
<?php
    class MyClass {
        var $property;
    }
    $obj1 = new MyClass;
    $obj1->property = 1;
    $obj2 = $obj1;
    $obj2->property = 2;
    echo $obj1->property; // Выводит 1 в PHP 4 и 2 в PHP 5
    echo $obj2->property; // Выводит 2
?>
```

В PHP 4 $obj2 представляет собой копию объекта $obj1, а в PHP 5 и $obj1 и $obj2 указывают на один и тот же объект, так как оператор $obj2 = $obj1 копирует не сам объект, а только его идентификатор.

Различные механизмы обработки объектов имеют место по причине того, что Zend Engine 1, исполнявший сценарии в PHP 4, хранил значения всех типов одинаковым образом в специальной структуре, называемой zval (Zend VALue). В PHP 5 также используется zval, однако теперь в нем хранятся все типы данных, за исключением объектов, которые располагаются в новой структуре, получившей название Object Store. Zval же хранит только идентификаторы объектов, вот почему при присвоении или передачи в функцию передается не сам объект, а только его идентификатор.

В том случае, если необходимо провести именно копирование объекта, как это делалось в PHP 4, то в PHP 5 придется явно использовать новый метод __clone(). При этом объект копируется со всеми своими методами, свойствами и их значениями:

```
<?php
    class MyClass{
        var $property;
    }
    $obj1 = new MyClass;
    $obj1->property = 1;
    $obj2 = clone $obj1;
    echo $obj1->property; // Выводит 1
    echo $obj2->property; // Выводит 1
    $obj2->property = 2;
    echo $obj2->property; // Выводит 2
?>
```

Следует обратить внимание на то, что к методу __clone() нельзя обратиться непосредственно и для копирования объекта используется ключевое слово clone.

Метод __clone() необязательно описывать в самом классе, однако его явное определение, т.е. *перегрузка*, позволит изменить значения свойств копируемого объекта:

```
<?php
    class MyClass{
        var $property;
        function __clone() {
            $this->property = 2;
        }
    }
    $obj1 = new MyClass;
    $obj1->property = 1;
    $obj2 = clone $obj1;
    echo $obj1->property; // Выводит 1
    echo $obj2->property; // Выводит 2
?>
```

Метод __clone() не может принимать никакие аргументы, однако позволяет обратиться к получаемому в результате объекту через указатель $this.

Объектно-ориентированное программирование в PHP

В PHP 5 введены спецификаторы доступа ***public***, ***protected*** и ***private***, которые позволяют указать степень доступа к свойствам и методам класса.

К общедоступным (public) свойствам и методам можно получить доступ без каких либо ограничений.

Защищенные (protected) элементы класса доступны внутри класса, в котором они объявлены, и в производных от него классах.

Частные (private) элементы доступны только в классе, в котором они объявлены.

```
<?php
    class MyClass {
        public $public_value = "общедоступный элемент";
        protected $protected_value = "защищенный элемент";
        private $private_value = "частный элемент";
        public function printPrivate() {
            echo $this->private_value.'</br>';
        }
    }
    $obj1 = new MyClass;
    echo $obj1->public_value.'</br>'; // Выводит "общедоступный элемент«
    class MyClass1 extends myClass {
        public function printProtected() {
            echo $this->protected_value.'</br>';
        }
    }
    $obj2 = new MyClass1();
    $obj2->printProtected(); // Выводит "защищенный элемент"

    $obj1->printPrivate(); //Выводит "частный элемент"

    echo $obj1->protected_value.'</br>'; // Вызывает ошибку доступа
    echo $obj1->private_value.'</br>'; // Вызывает ошибку доступа
?>
```

Объектно-ориентированное программирование в PHP

В PHP 5 введены **конструкторы** и **деструкторы**.

Метод-конструктор вызывается автоматически при каждом создании объекта. И хотя конструктор появился в PHP давно (эту роль выполнял метод, названный именем класса), но в PHP 5 была изменена схема именования конструктора - метод **__construct()** является теперь конструктором класса.

Аналогично, при уничтожении объекта вызывается специальный метод **__destruct()** – деструктор класса.

Для целей совместимости с предыдущей версией PHP 5 поступает следующем образом: *если при создании объекта в классе не найдет конструктор __construct(), то PHP пытается выполнить метод, имя которого совпадает с именем класса*. Т.о. конструкторы PHP 4 будут работать с PHP 5 без каких-либо изменений кода.

```
<?php
    class MyClass {
        function __construct() {
            echo "Запущен конструктор";
        }
        function __destruct() {
            echo "Запущен деструктор";
        }
    }
    $obj = new MyClass(); // Выводит "Запущен конструктор"
    unset($obj); // Выводит "Запущен деструктор"
?>
```

Объектно-ориентированное программирование в PHP

Если необходимо вызвать конструктор или деструктор базового класса, то необходимо это делать явно, через указатель parent.

```
<?php
    class MyClass {
        function __construct() {
            echo "Запущен конструктор базового класса";
        }
        function __destruct() {
            echo "Запущен деструктор базового класса";
        }
    }
    class MyClass1 extends MyClass {
        function __construct() {
            echo "Запущен конструктор дочернего класса";
            parent::__construct();
        }
        function __destruct() {
            echo "Запущен деструктор дочернего класса";
            parent::__destruct();
        }
    }
    $obj = new MyClass1();
    unset($obj);
?>
```

Объектно-ориентированное программирование в PHP

В PHP 5 впервые введены ***абстрактные*** (abstract) классы и методы.
Абстрактные методы имеют только объявление и не имеют реализации. Класс, который содержит такие методы, должен быть обязательно объявлен как абстрактный.

```
<?php
    abstract class MyClass {
        abstract public function abstrFunc();
    }
    class MyClass1 extends MyClass {
        public function abstrFunc() {
            echo 1;
        }
    }
    $obj = new MyClass1;
    $obj->abstrFunc(); // Выводит 1
?>
```

При этом невозможно создать объект абстрактного класса, можно только определять новые классы от базового абстрактного класса и создавать объекты уже от производных классов.

Стоит отметить, что абстрактные классы также могут содержать и обычные (не абстрактные) элементы.

Объектно-ориентированное программирование в PHP

В PHP 5 появилось понятие ***интерфейса***(interface) . Интерфейсами являются абстрактные классы, содержащие только абстрактные методы и не имеющие никаких свойств.

Основное отличие интерфейсов от абстрактных классов заключается в том, что в PHP 5 класс не может быть порожден от нескольких классов, в том числе и абстрактных, но зато может быть создан на основе любого числа интерфейсов.

При этом в интерфейсе методы объявляются ключевым словом function без указания каких-либо спецификаторов, в том числе и abstract.

```
<?php
    interface Int1 {
        function func1();
    }
    interface Int2 {
        function func2();
    }
    class MyClass implements Int1, Int2 {
        public function func1() {
            echo 1;
        }
        public function func2() {
            echo 2;
        }
    }
    $obj = new MyClass;
    $obj->func1(); // Выводит 1
    $obj->func2(); // Выводит 2
?>
```

Объектно-ориентированное программирование в PHP

PHP 5 введена новая возможность определять методы класса и сами классы как **финальные** (final).

Метод, при определении которого используется ключевое слово final, не может быть переопределен в классах, производных от данного класса.

```
<?php
    class MyClass {
        final public function func() {
            // Код метода
        }
    }
    class MyClass1 extends MyClass {
        // Следующий код вызывает ошибку
        // переопределения финального метода
        // базового класса MyClass
        public function func() {
            // Код метода
        }
    }
?>
```

Объектно-ориентированное программирование в PHP

Кроме этого, если final используется при определении самого класса, то порождение от него других классов становится невозможным.

```
<?php
    final class MyClass {
        // Код описания класса
    }
    // Следующий код вызывает ошибку
    // порождения от финального класса
    class MyClass1 extends MyClass {
        // Код описания класса
    }
?>
```

Если класс определен как final, то и все методы данного класса автоматически становятся финальными, таким образом, определять их явно как final уже нет необходимости.

Объектно-ориентированное программирование в PHP

В PHP 5 введен новый элемент класса – **константа**.

```
<?php
    class MyClass {
        const CONSTANT = "константа класса";
    }
    echo MyClass::CONSTANT; // Выводит "константа класса"
?>
```

Ранее написанные сценарии для PHP 4, не использующие функции или классы с именем const будут работать без модификации кода.

В PHP 5 возможно объявление **статических** свойств класса.

```
<?php
     class MyClass {
         static $static = 1;
     }
     echo MyClass::$static; // Выводит 1
?>
```

Статические свойства едины для всего класса и не могут принадлежать ни одному из объектов класса. Изменение такого свойства в одном из методов любого объекта приводит к его изменению для всех остальных объектов данного класса. Кроме этого, становится возможным обратиться к такому свойству вне контекста объекта.

Объектно-ориентированное программирование в PHP

Статические методы классов в PHP 5, также как и статические свойства, принадлежат всему классу в целом. Это позволяет использовать такой метод без создания какого-либо объекта такого класса.

```
<?php
    class MyClass {
        static function statFunc() {
            echo "статический метод";
        }
    }
    MyClass::statFunc(); // Выводит "статический метод"
?>
```

Однако *в статическом методе становится невозможным использовать указатель $this*, так как при вызове статического метода или неизвестно в контексте какого объекта он вызывается, или такого объекта может и не существовать вовсе.

Статические свойства могут использоваться, например, для счетчиков количества экземпляров объектов одного класса.

```
<?php
class Stats {
static $counter;
function __construct($name, $authors, $category,
Stats::$counter++;
}
// ...
function __destruct() {
parent::__destruct();
Stats::$counter--;
}
}
?>
```

Таким образом, обратившись к статическому свойству, можно определить количество инициализированных в данный момент экземпляров классов.

echo Stats::$counter;

Как было отмечено выше, что обращаться к статическим свойствам через указатель $this не допускается, так как статические свойства находятся вне контекста какого-либо одного объекта.

Статические методы классов в PHP 5, так же как и статические свойства, принадлежат всему классу в целом. Это позволяет использовать такой метод без инициализации объекта такого класса.

Чтобы проиллюстрировать эту возможность, добавим статический метод countItems(), возвращающий количество инициализированных объектов.

```
<?php
class Book extends Commodity implements Database, Stats {
static $counter;
// ...
static function countItems() {
return Book::$counter;
}
}
?>
```

Вызываться эта функция будет так:

echo Book::countItems();

Объектно-ориентированное программирование в PHP

Специальное ключевое слово **instanceof** в PHP 5 позволяет определять является ли объект экземпляром определенного класса, или же экземпляром класса производного от какого-либо класса.

```
<?php
    class MyClass { }
    $obj1 = new MyClass();
    if ($obj1 instanceof MyClass) {
        echo "\$obj1 - объект класса MyClass";
    }

    class MyClass1 extends MyClass { }
    $obj2 = new MyClass1();
    if ($obj2 instanceof MyClass) {
        echo "\$obj2 - объект класса, производного от MyClass";
    }

    interface Int { }
    class MyClass2 implements Int { }
    $obj3 = new MyClass2();
    if ($obj3 instanceof Int) {
        echo "\$obj3 - объект класса, созданного на основе интерфейса Int";
    }
?>
```

Также с помощью instanceof можно определить является ли объект экземпляром класса, созданного на основе определенного интерфейса.

Специальная функция **__autoload()** будет вызываться в PHP 5 в случае попытки создания объекта неопределенного класса.

```
<?php
    function __autoload($class) {
        echo "попытка создать объект неопределенного класса ", $class;
    }
    $obj = new MyClass;
?>
```

В качестве параметра функции __autoload() передается имя неопределенного класса.

Объектно-ориентированное программирование в PHP

В PHP 5 возможна **перегрузка** доступа к свойствам объектов.

```
<?php
    class MyClass {
        private $properties;
        function __set($name, $value) {
            echo "задание нового свойства $name = $value";
            $this->properties[$name]=$value;
        }
        function __get($name) {
            echo "чтение значения свойства ", $name;
            return $this->properties[$name];
        }
    }
    $obj = new MyClass;
    $obj->property = 1; // Выводит "задание нового свойства property=1"
    $a = $obj->property; // Выводит "чтение значения свойства property"
    echo $a; // выводит 1;
?>
```

Новые методы доступа **__get()** и **__set()** позволяют легко проводить динамическое назначение свойств объектам. В качестве параметров этим методам передаются имена свойств.

Метод __set() также получает и значение, которое устанавливается для свойства.

Пример

При вызове в *PHP 5* несуществующего метода объекта автоматически вызывается специальный метод **__call()**.

```
<?php
    class MyClass {
        function __call($name, $params) {
            echo "Вызван метод $name с параметром $params[0]";
        }
    }
    $obj = new MyClass;
    echo $obj->method(1); // Выводит "Вызван метод method
                          // с параметром 1"
?>
```

В качестве параметров __call() принимает имя вызываемого метода и передаваемые этому методу параметры.

В **PHP 5** возможен полный перебор всех свойств объекта в операторе *foreach*.

```
<?php
    class MyClass {
        public $a = 1;
        public $b = 2;
    }
    $obj = new MyClass;
    foreach ($obj as $name => $value) {
        echo "$name => $value "; // Выводит "a => 1 b => 2"
    }
?>
```

Специальные интерфейсы *PHP 5* **Iterator** и **IteratorAggregate** позволяют задавать поведение класса при его использовании с оператором foreach.

Объектно-ориентированное программирование в PHP

Интерфейс **Iterator** позволяет обработать все этапы просмотра объекта.

```
interface Iterator extends Traversable {
/* Methods */
abstract public mixed current ( void )
abstract public scalar key ( void )
abstract public void next ( void )
abstract public void rewind ( void )
abstract public boolean valid ( void )
}
```

Пустой интерфейс Traversable предназначен для идентификации всех классов для которых применимо перечисление с помощью foreach.

Пример

Интерфейс **IteratorAggregate** предназначен для создания внешнего итератора.

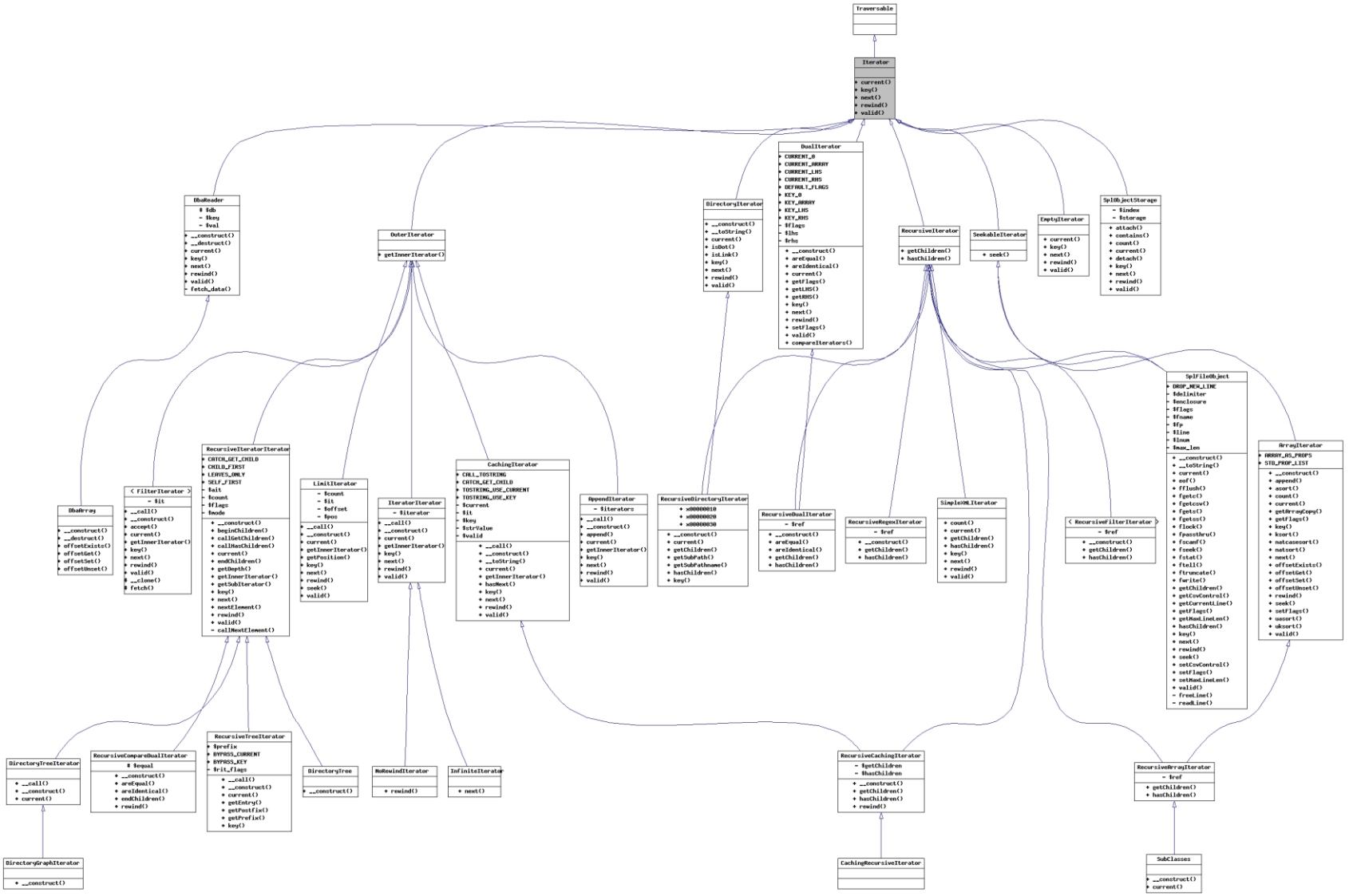
IteratorAggregate extends Traversable {
/* Methods */
abstract public Traversable getIterator (
void)
}

```
<?php
class myData implements IteratorAggregate {
    public $property1 = "Public property one";
    public $property2 = "Public property two";
    public $property3 = "Public property three";

    public function __construct() {
        $this->property4 = "last property";
    }

    public function getIterator() {
        return new ArrayIterator($this);
    }
}

$obj = new myData;

foreach($obj as $key => $value) {
    var_dump($key, $value);
    echo "\n";
}
?>
```

Пример

Объектно-ориентированное программирование в PHP

AppendIt
ArrayItera
CachingIt
Directory
EmptyIter
Filesyste
FilterItera
GlobItera
InfiniteIte
IteratorIte
LimitItera
MultipleIt
NoRewin
ParentIte
Recursive
Recursive
Recursive
Recursive
Recursive
Recursive
Recursive
RegexIte
SimpleXMLIterator



В **PHP 5** псевдо-константа **__METHOD__** возвращает имя класса и вызываемый метод.

```
<?php
    class MyClass {
        public function myMethod() {
            echo "вызов метода ", __METHOD__;
        }
    }
    $obj = new MyClass;
    $obj->myMethod(); // Выводит "вызов метода MyClass::myMethod"
    function myFunction() {
        echo "вызов функции ", __METHOD__;
    }
    myFunction(); // Выводит "вызов функции myFunction"
?>
```

При обращении к функции вне класса __METHOD__ возвращает только имя функции.

В PHP 5 введен еще один специальный метод класса - **__toString()**.

```
<?php
    class MyClass {
        function __toString() {
            return "вызван метод __toString()";
        }
    }
    $obj = new MyClass;
    echo $obj; // Выводит "вызван метод __toString()"
?>
```

Метод класса __toString() *позволяет выполнить перегрузку преобразования объекта в строку*.

Объектно-ориентированное программирование в PHP

В PHP 5 введена возможность **разыменования** (dereferencing) объектов, которые возвращаются функциями.

```
<?php
    class MyClass1 {
        public function showClassName() {
            echo "объект класса MyClass1";
        }
    }
    class MyClass2 {
        public function showClassName() {
            echo "объект класса MyClass2";
        }
    }
    function deref($obj) {
        switch ($obj) {
            case "MyClass1":
                return new MyClass1();
            case "MyClass2":
                return new MyClass2();
        }
    }
    deref("MyClass1")->showClassName(); // Выводит "объект  класса MyClass1"
    deref("MyClass2")->showClassName(); // Выводит "объект класса MyClass2“
?>
```

Данный механизм позволяет вызывать методы объектов, экземпляры классов которых возвращаются пользовательскими функциями.

Пространства имён

Пространства имён введены в PHP для решения проблем в больших PHP-библиотеках. В PHP все определения классов глобальны, поэтому авторы библиотек должны выбирать уникальные имена для создаваемых ими классов.

Это делается для того, чтобы при использовании библиотеки совместно с другими библиотеками не возникало конфликтов имён. Обычно это достигается введением в имена классов префиксов. Например: если мы будем использовать класс **dataBase** - велика вероятность, что такое имя класса будет присутствовать и в других библиотеках, а при их совместном использовании возникнет ошибка. Поэтому мы вынуждены использовать для класса другое имя. Например: **ourLibraryDataBase** Такие действия приводят к чрезмерному увеличению длины имён классов.

Пространства имён позволяют разработчику управлять зонами видимости имён, что избавляет от необходимости использования префиксов и чрезмерно длинных имён. Все это служит повышению читабельности кода.

Пространства имён доступны в PHP начиная с версии 5.3.0. Данная секция экспериментальна и возможно будет подвержена изменениям.

Пространства имён

Пространство имён определяется посредством ключевого слова *namespace*, которое должно находиться в самом начале файла. Пример:

```
<?php
    namespace MyProject::DB;
    const CONNECT_OK = 1;
    class Connection { /* ... */ }
    function connect() { /* ... */  }
    ?>
```

Это пространство имён может быть использовано в разных файлах.

Пространства имён могут включать определения классов, констант и функций. Но не должны включать обычного кода.

Пространства имён

Определение пространства имён работает так:
Внутри пространства имён все имена классов, функций и констант автоматически будут префиксированы именем пространства имён. Имена классов при вызове должны быть полными, так например при вызове класса из примера выше должно использоваться следующее имя **MyProject::DB::Connection**.
Определения констант создают константы, состоящие из имени пространства имён и имени константы. Как и константы классов - константы пространства имён могут содержать данные только скалярного типа.
Поиск неквалифицированного имени класса (т.е. не содержащего *::*) осуществляется в следующей последовательности:

- Попытка найти класс в текущем пространстве имён (т.е. префиксирование класса именем текущего пространства имён) без попытки автозагрузки (autoload).
- Попытка найти класс в глобальном пространстве имён без попытки автозагрузки (autoload).
- Попытка автозагрузки в текущем пространстве имён.
- В случае неудачи предыдущего - отказ.

Поиск неквалифицированного имени функции (т.е. не включающего *::*) во время выполнения производится сначала в текущем пространстве имён, затем в глобальном пространстве имён.
Поиск неквалифицированного имени константы производится сначала в текущем пространстве имён, затем среди глобально объявленных констант.

Объектно-ориентированное программирование в PHP

Пространства имён

Использование пространств имён

На имена классов и функций, объявленных внутри пространства имён всегда можно сослаться по полному имени: **MyProject::DB::Connection** или **MyProject::DB::connect()** .

```
<?php
    require 'MyProject/Db/Connection.php';
    $x = new MyProject::DB::Connection;
    MyProject::DB::connect();
 ?>
```

Пространства имён могут быть импортированы в текущий контекст (глобальный или пространство имён) при помощи оператора *use*. Синтаксис оператора use:

```
<?php
    /* ... */
    use Some::Name as Othername;
    // Самая простая форма :
    use Foo::Bar;
    // это то же что и :
    use Foo::Bar as Bar;
?>
```

Импортированные пространства имён работают следующим образом: каждый раз, когда компилятор встречает локальное имя *Othername* (как простое имя или как префикс более длинного, разделенного *::* имени), оно заменяется на импортированное имя *Some::Name*.

Пространства имён

Оператор *use* может быть использован только в глобальном пространстве имён - не внутри класса или функции. Импортированные имена действуют с точки импортирования до конца текущего файла. Рекомендуется импортировать имена в начале файла во избежание путаницы.

```
<?php
    require 'MyProject/Db/Connection.php';
    use MyProject::DB;
    use MyProject::DB::Connection as DbConnection;

    $x = new MyProject::DB::Connection();
    $y = new DB::connection();
    $z = new DbConnection();
    DB::connect();
    ?>
```

Замечание: Операция импорта пространств имён выполняется только во время компиляции. Все локальные имена заменяются компилятором на их полные эквиваленты. Динамический импорт пространств имён невозможен.

Глобальное пространство имён

Без определения какого-либо пространства имён, все определения классов и функций оказываются в глобальном пространстве - как это было в PHP до появления поддержки пространств имён. Префиксирование имени знаками *::* указывает, что имя из глобального пространства имён - работает даже в контексте пространства имён.

```
<?php
    namespace A::B::C;

    /* This function is A::B::C::fopen */
    function fopen() {
    /* ... */
    $f = ::fopen(...); // вызывает глобальную fopen
    return $f;
    }
?>
```

Пространства имён

__NAMESPACE__

Константа времени компиляции **__NAMESPACE__** определяет текущее пространство имён. Вне пространства имён эта константа имеет значение пустой строки. Эта константа используется когда требуется сформировать полное имя класса или функции из текущего пространства имён.

```
<?php
    namespace A::B::C;

    function foo() {
    // do stuff
    }

    set_error_handler(__NAMESPACE__ . "::foo");
?>
```

Правила разбора имён

Разбор имён происходит по следующим правилам:

1. Все квалифицированные имена транслируются во время компиляции в соответствии с текущими импортированными пространствами имён. К примеру, если импортировано пространство имён A::B::C, вызов C::D::e() будет транслирован как A::B::C::D::e().
2. Неквалифицированные имена классов транслируются во время компиляции в соответствии с текущими импортированными пространствами имён (полные имена заменяют короткие импортированные имена). К примеру, если пространство имён *A::B::C* импортировано, new C() будет транслировано как new A::B::C().
3. Внутри пространства имён вызов неквалифицированных функций, определенных в этом же пространстве имён интерпретируется как вызов в данном пространстве имён во время компиляции.
4. Внутри пространства имён (например *A::B*) вызов неквалифицированных функций, не определенных в этом пространстве имён будет разрешаться во время выполнения. Вызов функции *foo()* будет разрешаться следующим образом:
    1. Поиск в текущем пространстве имён: *A::B::foo()*.
    2. Поиск ***внутренней*** PHP функции *foo()*.

В случае неудачи всех предыдущих попыток будет использован вызов определенной в глобальном пространстве имён функции *::foo()*.

Правила разбора имён

5. Внутри пространства имён (например *A::B*), вызов неквалифицированных классов разрешается во время выполнения. Например вызов new C() будет разрешаться следующим образом:
   1. Поиск класса в текущем пространстве имён: *A::B::C*.
   2. Попытка вызова *внутреннего* PHP-класса *C*.
   3. Попытка автозагрузки *A::B::C*.

В случае неудачи всех предыдущих, будет использован вызов new ::C().

6. Вызов квалифицированных функций разрешается во время выполнения. Например вызов *A::B::foo()* будет разрешаться следующим образом :
   1. Поиск функции *foo()* в пространстве имён *A::B*.
   2. Поиск класса *A::B* и вызов его статического метода *foo()*. Будет сделана автозагрузка класса, если необходимо.

7. Квалифицированные имена классов разрешаются во время компиляции, как классы соответствующего пространства имён. К примеру new A::B::C() будет ссылаться на класс **C** пространства имён *A::B*.

Исключения

Модель исключений (exceptions) в PHP 5 проще, чем в некоторых языках программирования.

Исключение можно сгенерировать при помощи оператора *throw*, и можно перехватить оператором *catch*.

Код генерирующий исключение, должен быть окружен блоком *try*, для того чтобы можно было перехватить и обработать исключение.

Каждый блок *try* должен иметь как минимум один соответствующий блок *catch*. Так же можно использовать несколько блоков *catch*, перехватывающих различные классы исключений.

Нормальное выполнение (когда не генерируется исключений в блоках *try* или когда класс сгенерированного исключения не совпадает с классами, объявленными в соответствующих блоках *catch*) будет продолжено за последним блоком *catch*.

Исключения так же могут быть сгенерированны (или перегерерированы) оператором *throw* внутри блока *catch*.

При генерации исключения, код следующий ниже оператора *throw* исполнен не будет, а PHP предпримет попытку найти первый блок *catch*, перехватывающий исключение данного класса. Если исключение не будет перехвачено, PHP выдаст сообщение об ошибке: "*Uncaught Exception ...*" (Неперехваченное исключение), если конечно не был определен обработчик ошибок при помощи функции set_exception_handler().

Исключения

```
<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Деление на ноль.');
    }
    else return 1/$x;
}

try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Выброшено исключение: ',  $e->getMessage(), "\n";
}

// Продолжение выполнения
echo 'Hello World';

?>
```

Объектно-ориентированное программирование в PHP

Исключения

Наследование исключений

Определенный пользователем класс исключения должен быть определен, как класс расширяющий (наследующий) встроенный класс Exception. Ниже приведены методы и свойства класса Exception, доступные дочерним классам.

```
<?php
class Exception
{
    protected $message = 'Unknown exception';   // сообщение
    protected $code = 0;                        // Код исключения,     определяемый пользователем
    protected $file;                            // файл в котором было     выброшено исключение
    protected $line;                            // строка в которой было     выброшено исключение
    function __construct($message = null, $code = 0); /* Если класс, наследуемый от Exception
переопределяет конструктор, необходимо вызвать в конструкторе parent::__construct(), чтобы быть
уверенным, что все данные будут доступны.  */
    final function getMessage();                // Возвращает сообщение     исключения
    final function getCode();                   // Код исключения
    final function getFile();                   // Файл, где выброшено     исключение
    final function getLine();                   // Строка, выбросившая исключение
    final function getTrace();                  // Массив backtrace()
    final function getTraceAsString();          // Обратная трассировка как строка
    /* Overrideable - т.е. то, что можно переопределить */
    function __toString();                      // должен вернуть  форматированную строку, для отоб
ражения
}
?>
```

Пример

Обработка HTML-форм с помощью PHP

Внутри PHP-скрипта существует несколько способов получения доступа к данным, переданным клиентом по протоколу HTTP.

До версии PHP 4.1.0 использовались ассоциативные массивы **$HTTP_POST_VARS** и **$HTTP_GET_VARS**, ключами которых являлись имена переданных переменных, а значениями – соответственно значения этих переменных.

Специальный массив – **$_REQUEST**. Этот массив содержит данные, переданные методами POST и GET, а также с помощью HTTP cookies. Это ***суперглобальный ассоциативный массив***.

После введения массива **$_REQUEST** массивы **$HTTP_POST_VARS** и **$HTTP_GET_VARS** для однородности были переименованы в **$_POST** и **$_GET**. В отличие от своих предшественников, массивы **$_POST** и **$_GET** стали ***суперглобальными***, т.е. доступными напрямую и внутри функций и методов.

```
<?
//если использован метод POST – надо выводить сообщение пользователю
if ($_SERVER["REQUEST_METHOD"]=="POST")
{  //формируем строку с именем пользователя
  $usr_name=$_POST["first_name"]." ".$_POST["last_name"];
  echo "Уважаемый, ".$usr_name."!<br> Ваше сообщение:<br> ".$_POST["message“];
}
//иначе просто выводим форму
else{
?>
  <html>
  <head>
            <title>Форма</title>
  </head>
  <body>
  <h1>Форма обратной связи</h1>
   <form action="<? echo $_SERVER["PHP_SELF"]?>" method=POST>
                <?//используем $_SERVER["PHP_SELF"] для отправки данных в этот же самый скрипт?>
  Имя <br><input type=text name="first_name"><br>
  Фамилия <br><input type=text name="last_name"><br>
  E-mail <br><input type=text name="email"><br>
  Сообщение <br><textarea name=message cols=50 rows=5></textarea><br>
  <input type=submit value="Отправить">
  <input type=reset value="Отменить">
   </form>
  </body>
  </html>
<?
}
?>
```

При приеме значений от checkbox, в случае если он не установлен, на сервер не передается никаких данных. Чтобы не получать предупреждений об обращении в несуществующей переменной, возможно использовать следующий подход:

```
<?php ## Прием значений от чекбокс.
if (@$_REQUEST['doGo']) {
  foreach (@$_REQUEST['known'] as $k=>$v) {
    if($v) echo "Вы знаете язык $k!<br>";
    else echo "Вы не знаете языка $k. <br>";
  }
}
?>
<form action="<?=$_SERVER['SCRIPT_NAME']?>" method=post>
Какие языки программирования вы знаете?<br>
<input type=hidden name="known[PHP]" value="0">
  <input type=checkbox name="known[PHP]" value="1">PHP<br>
<input type=hidden name="known[Perl]" value="0">
  <input type=checkbox name="known[Perl]" value="1">Perl<br>
<input type=submit name="doGo" value="Go!">
</form>
```

Предопределенные переменные

Любому запускаемому скрипту PHP предоставляет большое количество предопределенных переменных.

Начиная с версии 4.1.0, PHP предоставляет дополнительный набор предопределенных массивов, содержащих переменные web-сервера (если они доступны), окружения и пользовательского ввода.

Суперглобальные переменные PHP

$_SERVER

Переменные, установленные web-сервером либо напрямую связанные с окружением выполнения текущего скрипта. Аналог старого массива *$HTTP_SERVER_VARS* (который по-прежнему доступен, но не рекомендуется).

$_ENV

Переменные, передаваемые скрипту через окружение. Аналог старого массива *$HTTP_ENV_VARS* (который по-прежнему доступен, но не рекомендуется).

Все глобальные переменные.

Обработка строк в PHP

Для работы со строковыми(текстовыми) данными в PHP предусмотрен очень богатый набор функций. Для использования этих функций не требуется выполнения каких-либо дополнительных действий(подключения библиотек и т.п.), поскольку они являются частью ядра PHP.

Функции разделения/объединения строк

strtok()

string strtok(string arg1, string arg2)

Функция возвращает строку по частям. Она возвращает часть строки arg1 до разделителя arg2. При последующих вызовах функции возвращается следующая часть до следующего разделителя, и так до конца строки.

При первом вызове функция принимает два аргумента: исходную строку arg1 и разделитель arg2.

При каждом последующем вызове arg1 указывать не следует, иначе будет возвращаться первая часть строки.

Пример:

```
<?
    $str ="I am very glad to see%you% people";
   $tok = strtok($str, " ");
    while($tok)
    {
      echo ($tok);
      echo (" ");
      $tok = strtok(" %");
    };
?>
```

Если в строке последовательно встречаются два или более разделителей, функция возвращает пустую строку, что, к примеру, может прекратить цикл обработки, как в этом примере.

explode()

string explode(string arg, string str [, int maxlimit])

Функция explode() производит разделение строки в массив. Она возвращает массив строк, каждая из которых соответствует фрагменту исходной строки ***str***, находящемуся между разделителями, указанными аргументом ***arg***. Необязательный параметр ***maxlimit*** указывает максимальное количество элементов в массиве. Оставшаяся неразделенная часть будет содержаться в последнем элементе. Пример:

```
<?
    $str = "one two three for five";
    $str_exp = explode(" ", $str);
    /* теперь $str_exp = array([0] => one, [1] => two,
      [2] => three, [3] => for, [4] => five)
    */
  ?>
```

implode()

string implode(string var, array param)

Функция implode() является обратной функции explode() и производит объединение массива в строку. Функция возвращает строку, которая последовательно содержит все элементы массива, заданного в параметре param, между которыми вставляется значение, указанное в параметре var. Пример:

```
<?
    $str = "one two three four five";
    $str_exp = explode(" ", $str);
    /* $str_exp = array([0] => one, [1] => two,
      [2] => three, [3] => four, [4] => five)
    */
    $str_imp = implode(" ", $str_exp);
    echo($str_imp);
  //Результат:
  //one two three four five
?>
```

Работа с подстроками
substr()

string substr(string string, int start[, int length])

Эта функция возвращает часть строки. Первый аргумент – исходная строка; второй – индекс символа в строке(отсчет начинается с нуля), начиная с которого необходимо вернуть подстроку; третий – длина строки в символах, которую надо вернуть. Если третий аргумент не указан, то возвращается вся оставшаяся часть строки. Пример:

```
<?
    $string = substr("Hello, world!", 6, 2);
    echo ($string);
  ?>
```

strpos()

string strpos(string haystack, string needle[, int offset])

Эта функция возвращает позицию в строке haystack, в которой найдена переданная ей подстрока needle. Пример:

```
<?
    $string = strpos("Hello, world!", "world");
    echo($string);
  ?>
```

strstr()

string strstr(string haystack, string needle)

Функция strstr() возвращает участок строки, заданной в параметре haystack, начиная с первого фрагмента, указанного в параметре needle и до конца строки. В случае неудачи функция возвращает false. Пример:

```
<?
    $url = "http://www.sevntu.com.ua";
    $www = strstr($url,"w");
    echo ($www);
  ?>
```

str_replace()

string str_replace(string from, string to, string str)

Функция str_replace() заменяет в строке str все вхождения подстроки from на to и возвращает результат – обработанную строку. Если и *search*, и *replace* - массивы, то **str_replace()** использует все значения массива *search* и соответствующие значения массива *replace* для поиска и замены в *subject*. Если в массиве *replace* меньше элементов, чем в *search*, в качестве строки замены для оставшихся значений будет использована пустая строка. Если *search* - массив, а *replace* - строка, то *replace* будет использована как строка замены для каждого элемента массива *search*. Пример:

```
<?php ## Различия между strtr() и str_replace().
$text = "matrix has you";
$repl = array("matrix"=>"you", "you"=>"matrix");
echo "str_replace(): ".
  str_replace(array_keys($repl), array_values($repl), $text)."<br>";
echo "strtr(): ".
  strtr($text, $repl);
?>
```

Сравнение строк

strcmp()

int strcmp(string str1, string str2)

Эта функция сравнивает две строки и возвращает:

0 - если строки полностью совпадают;

>0 - если, строка str1 лексикографически больше str2;

<0 - если, наоборот, строка str1 лексикографически меньше str2.

Функция является чувствительной к регистру, т.е. регистр символов влияет на результаты сравнений(поскольку сравнение происходит побайтно). Пример:

```
<?
    $str1 = "ttt";
    $str2 = "ttttttttt";
    echo("Result of strcmp ($str1 , $str2) is ");
    echo(strcmp ($str1, $str2)); echo("<br>");
    echo("Result of strcmp ($str2, $str1)> is ");
    echo(strcmp ($str2, $str1)); echo("<br>");
    echo("Result of strcmp ($str1 , $str1) is ");
    echo(strcmp ($str1,$str1));
  ?>
```

strlen()

int strlen (string $string)

Данная функция возвращает длину строки, которую принимает в качестве аргумента. Пример:

```
<?
    $string = "Hello, world!!!";
    $string_len = strlen($string);
    echo ($string_len);
  ?>
```

htmlspecialchars()

htmlspecialchars(string str [, int quote_style [, string charset]])

Функция производит преобразование спецсимволов в их HTML эквиваленты (эскейп-последовательности). Это гарантирует, что введенный пользователем в форме текст отобразится браузером так же, как был введен.
Первый аргумент – строка, в которой надо выполнить преобразование. В качестве второго необязательного аргумента принимается константа, задающая режим преобразования кавычек. По умолчанию, используется ***ENT_COMPAT***, преобразующая двойные кавычки, при этом одиночные остаются без изменений. В режиме ***ENT_QUOTES*** преобразуются и двойные, и одиночные кавычки. а в режиме ***ENT_NOQUOTES*** и двойные, и одиночные кавычки остаются без изменений. Третий необязательный аргумент принимает строку, представляющую набор символов, используемых в преобразовании (по умолчанию ISO-8859-1) :

```
<?
$msg = $_POST["msg"];
echo $msg;
$msg = htmlspecialchars($msg);
?>
```

parse_url()

array parse_url(string url)

Эта функция возвращает ассоциативный массив, включающий множество различных существующих компонентов URL: "scheme", "host", "port", "user", "pass", "path", "query" и "fragment". Пример:

```
<?
    $url = "http://www.google.com. ua/search?hl=ru&ie=UTF-8&oe=UTF-8&q=php&lr=";
    $arr = parse_url($url);
    print_r($arr);
  ?>
```

Работа с файлами в PHP

Функция fopen

Функция **fopen()** используется для открытия файла. Синтаксис этой функции:

resource fopen (имя_файла, тип_доступа [, use_include_path])

Тип доступа	Описание
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 записи и для чтения; устанавливает указатель файла на его начало и усекает файл до нулевой длины. Если файл не существует, то пытается создать его.
a	Открывает файл только для записи; устанавливает указатель файла в его конец. Если файл не существует, то пытается создать его.
a+	Открывает файл для записи и для чтения; 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 записи и для чтения; 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.

```
<?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.ua/dir/file.php","r");
/* открывает на чтение файл, находящийся по указанному адресу*/
?>
```

Закрытие соединения с файлом

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 :

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);
?>
```

Запись данных в файл

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])

```
<?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);
?>
```

Чтение данных из файла

Чтобы прочитать данные из файла, нужно воспользоваться одной из специальных функций: **file, readfile, file_get_contents, fread, fgets** и т.п.

Функция fread

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

string fread (указатель на файл, длина)

```
<?php
$h = fopen("my_file.html","r+");
// отрываем 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
?>
```

Загрузка файлов на сервер

Загрузка файла на сервер включает в себя следующие этапы:

1. Пользователю предъявляется форма, которая позволяет ему выбрать файл для загрузки (поле ввода <input type='file'>).

2. Пользователь выбирает файл и отправляет форму.

3. Браузер отсылает файл и информацию о нем серверу как часть POST-запроса.

4. Сервер помещает файл во временный каталог.

5. PHP проверяет и перемещает файл из временного каталога в указанный каталог для постоянного хранения.

Загрузка файлов на сервер. Форма.

Особенности формы для отправки файла на сервер:

1. Использование для отправки формы метода POST(атрибут тега form - method = "post").

2. Указание специального атрибута **enctype** со значением “multipart/form-data”.

3. Указание скрипта выполняющего прием данных на сервере в атрибуте action тега form.

Загрузка файлов на сервер. Серверная часть.

Для обработки переданных файлов на сервере используется **глобальный двухмерный массив $_FILES**, содержащий в первом индексе имя поля ввода файла формы, а во втором следующие поля:

name - имя загружаемого файла.
type - MIME-type загружаемого файла. Важный параметр для обеспечения безопасности.
tmp_name - физический путь и имя временного файла.
error - код ошибки. Если **0**, то ошибок нет.
size - размер загружаемого файла.

Для копирование временного файла в требуемое место может быть использована функция move_uploaded_file:
bool **move_uploaded_file** (string $filename , string $destination)

Загрузка файлов на сервер. Пример.

```
<?php
if (@$_REQUEST['doUpload']){
echo '<pre>Содержимое $_FILES: '.print_r($_FILES, true)."</pre><hr>";
  if(copy($_FILES["messages "]["tmp_name"],
   $_FILES["messages "]["name"]))
  { echo("Файл успешно загружен");}
 else { echo("Ошибка загрузки файла");}
}
?>
<html> <head>
   <title> Загрузка файлов на сервер </title>
 </head>
<body> <h2> Форма для загрузки файлов </h2>
 <form action="<?=$_SERVER['SCRIPT_NAME']?>" method="post"
enctype="multipart/form-data">
  <input type="file" name="messages" <br>
  <input type="submit" name="doUpload" value="Загрузить"
</form>
</body>
</html>
```

Взимодействие PHP с СУБД. СУБД MySQL.

PHP обеспечивает возможность взаимодействия с большим количеством различных систем управления базами данных (MySQL, SQLite, PostgreSQL, Oracle, Microsoft SQL Server, …).
Наиболее часто используется связка PHP с СУБД MySQL.

MySQL — свободная система управления базами данных (СУБД). MySQL в настоящий момент является собственностью компании Oracle Corporation, получившей её вместе с поглощённой Sun Microsystems, осуществляющей разработку и поддержку приложения. Но распространяется MySQL под GNU General Public License или под коммерческой лицензией(на выбор пользователя).

MySQL входит в LAMP(Linux, Apache, MySQL, PHP) — акроним, обозначающий набор серверного программного обеспечения, широко используемый в Internet.

В дистрибутив PHP начиная с версии 4 входит расширение, содержащее встроенные функции для работы с СУБД MySQL.

Для администрирования СУБД MySQL в состав Денвер включен пакет phpMyAdmin

Взимодействие PHP с СУБД MySQL

Соединение PHP с сервером MySQL. Функция mysql_connect

Прежде чем работать с базой данных, необходимо установить с ней сетевое соединение, а также провести авторизацию пользователя. Для этого служит функция mysql_connect()

resource ***mysql_connect****([string $server[,string $username[,string $password]]])*

Эта функция устанавливает сетевое соединение с базой данных MySQL, расположенной на хосте $server (по умолчанию это localhost) и возвращает идентификатор (дескриптор) открытого соединения. *Вся дальнейшая работа ведется именно с этим идентификатором*. Все другие функции, принимающие этот идентификатор в качестве аргумента, будут однозначно определять выбранную СУБД.

Работа PHP с СУБД MySQL

```
<?
$dblocation = "localhost"; //Имя сервера
$dbuser = "root"; //Имя пользователя
$dbpasswd = ""; //Пароль
//Осуществляем соединение с сервером базы данных
//Подавляем вывод ошибок символом @ перед вызовом функции
$dbcnx = @ mysql_connect($dblocation, $dbuser, $dbpasswd);
if (!$dbcnx) //Если дескриптор равен 0, соединение не
установлено
{
//Выводим предупреждение
echo("<P>В настоящий момент сервер базы данных не
доступен, поэтому корректное отображение страницы
невозможно.</P>");
exit ();
}
?>
```

Разрыв соединения с сервером. Функция *mysql_close*

Соединение с MySQL – сервером будет автоматически закрыто по завершении работы сценария, либо при вызове функции mysql_close:

*bool **mysql_close** ([resource $link_identifier])*

```
<?
$dblocation = "localhost"; //Имя сервера
$dbuser = "root"; //Имя пользователя
$dbpasswd = ""; //Пароль
//Осуществляем соединение с сервером базы данных
//Подавляем вывод ошибок символом @ перед вызовом функции
$dbcnx = @ mysql_connect($dblocation, $dbuser, $dbpasswd);
if (!$dbcnx) //Если дескриптор равен 0, соединение не установлено
{
  //Выводим предупреждение
  echo("<P>В настоящий момент сервер базы данных не доступен, поэтому корректное отображение страницы невозможно.</P>");
  exit ();
}
echo("<P>Содержимое страницы...</P>");
if (mysql_close($dbcnx)) //разрываем соединение
 echo("Соединение с базой данных прекращено");
else echo("Не удалось завершить соединение");
?>
```

Выбор базы данных. Функция *mysql_select_db*

До того, как послать первый запрос серверу MySQL, необходимо указать, с какой именно базой данных будет происходить работа. Для этого предназначена функция mysql_select_db:

*bool **mysql_select_db**(string $database_name [,resource $link_identifier])*

```
<?
//….
//Соединение с СУБД
//…
if (! @mysql_select_db($dbname, $dbcnx))
{
//Выводим предупреждение
echo("<P> В настоящий момент база данных недоступна, поэтому корректное отображение страницы невозможно. </P>");
exit(); }
?>
```

Обработка ошибок

Если в процессе работы с MySQL возникают ошибки (например, в запросе не сбалансированы скобки или же не хватает параметров), то сообщение об ошибке и ее номер можно получить с помощью функций:

int ***mysql_errno*** *([int $link_identifier])*

string ***mysql_error****([int $link_identifier])*

Обычно mysql_error используют вместе с конструкцией **or die ()**, например:

```
@mysql_connect("localhost", "user", "password")
or die("Ошибка при подключении к базе данных:
".mysql_error());
```

Выполнение запросов к базе данных

Для отправки SQL-запроса серверу СУБД используется функция mysql_query:

resource mysql_query (string query)

Функция возвращает дескриптор запроса(*resource*) в случае успеха и false — в случае неудачного выполнения запроса. Дескриптор запроса используется впоследствии для доступа к данным, в случае выполнения запроса на выборку данных.

Для определения количества записей в результате запроса можно воспользоваться функцией mysql_num_rows:

int mysql_num_rows (resource result)

Для доступа к значениям полей записей может использоваться функция mysql_fetch_row:

array mysql_fetch_row (resource result)

Функция mysql_fetch_row в качестве аргумента принимает дескриптор запроса, а возвращает массив, содержащий данные записи результата запроса, или FALSE, если записей больше нет. Повторный вызов функции позволяет выбрать следующую запись, если в результате запроса было возвращено несколько записей.

Работа PHP с СУБД MySQL

Для выборки данных записи в виде ассоциативного массива, с именами полей в виде ключей массива, может использоваться функция mysql_fetch_array:

array mysql_fetch_array (resource result [, int result_type])

Эта функция в качестве аргумента принимает дескриптор запроса, возвращаемый функцией mysql_query, а возвращает значения полей в виде ассоциативного массива, численный массив или оба массива.

Второй необязательный аргумент result_type используется для указания типа возвращаемого массива и может принимать следующие значения: MYSQL_ASSOC, MYSQL_NUM и MYSQL_BOTH. Значением по умолчанию является: MYSQL_BOTH.

Для выборки данных записи в виде только ассоциативного массива может использоваться функция mysql_fetch_assoc:

array mysql_fetch_assoc (resource result)

С помощью функции mysql_result:

mixed mysql_result (resource result, int row [, mixed field])

можно получить доступ к отдельному полю field записи с номером row результата запроса.

Для ускорения доступа при работе с большими результатами запросов, целесообразнее использовать одну из функций, обрабатывающих сразу целый ряд результата (mysql_fetch_array, mysql_fetch_row).

Комплексный пример работы PHP с СУБД MySQL

```
$dblocation = "localhost"; //Имя сервера
$dbuser = "root"; //Имя пользователя
$dbpasswd = ""; //Пароль
$dbname="test";
//Осуществляем соединение с сервером базы данных
//Подавляем вывод ошибок символом @ перед вызовом функции
$dbcnx = @ mysql_connect($dblocation, $dbuser, $dbpasswd);
if (!$dbcnx) { //Если дескриптор равен 0, соединение не установлено
  //Выводим предупреждение
  echo("<P>В настоящий момент сервер базы данных недоступен, поэтому
корректное отображение страницы невозможно.</P>");
  exit ();
 }
//Соединение с базой данных
if (! @mysql_select_db($dbname, $dbcnx)) {
  //Выводим предупреждение
  echo("<P> В настоящий момент база данных недоступна, поэтому
корректное отображение страницы невозможно. </P>");
  exit();
 }
```

Работа PHP с СУБД MySQL

```
//обработка операции добавления записи
if ($_SERVER["REQUEST_METHOD"]=="POST"){
  $sql="insert into messages (messages_id, messages_date,
messages_autor,messages_mail, messages_subj, messages_text)
//при вставке 0 в автоинкрементное поле генератор сформирует очередное значение
//функция NOW() используется в MySQL для получения текущих даты времени
values(0,        NOW(),'${_POST["first_name"]}',
                '${_POST["email"]}','${_POST["subj"]}','${_POST["message"]}')";
  $do_insert = mysql_query($sql)or die("Insert_Error:". mysql_error());
}?>
<h2>Форма добавления записей</h2>
<form action="<? echo $_SERVER["PHP_SELF"]?>" method=POST>
  ФИО <br><input type=text name="first_name"><br>
  E-mail <br><input type=text name="email"><br>
  Тема сообщения <br><input type=text name="subj"><br>
  Сообщение <br><textarea name=message cols=50 rows=5></textarea><br>
  <input type=submit value="Отправить">
  <input type=reset value="Отменить">
</form>
<h2>Сообщения пользователей</h2>
 <table border=1>
<tr><th>Дата</th><th>Автор</th><th>Тема</th><th>Email</th><th>Сообщение
</th></tr>
```

Постраничный вывод данных из БД

Довольно часто при работе с базами данных в WEB возникает необходимость в отображении большого объема однотипной информации, при этом, для удобства восприятия, её следует разбивать на части, т.е. реализовать постраничный просмотр этой информации.

Для решения задачи частичной выборки записей из таблицы базы данных в синтаксисе оператора SELECT языка SQL в СУБД MySQL предусмотрен оператор LIMIT:

LIMIT [offset,] rows

Как видно, LIMIT принимает один или два числовых аргумента. Эти аргументы должны быть целочисленными константами. Если заданы два аргумента, то первый указывает на индекс первой возвращаемой строки, а второй задает максимальное количество возвращаемых строк. При этом индекс первой строки 0:

*mysql> SELECT * FROM table LIMIT 5,10; # возвращает строки 6-15*

Если задан один аргумент, то он показывает максимальное количество возвращаемых строк:

*mysql> SELECT * FROM table LIMIT 5; # возвращает первых 5 строк*

Т.е. *LIMIT n* эквивалентно *LIMIT 0,n*. Пример

При реализации постраничного вывода информации из БД кроме частичной выборки необходимо также решить задачу ***организации строки ссылок на страницы***, для обеспечения пользователя возможностью «перелистывания» страниц. При формировании строки ссылок на страницы необходимо знать общее количество записей. Для получения общего количества отображаемых записей возможно использование дополнительного SQL-запроса, вычисляющего и возвращающего общее количество записей. Например:

SELECT COUNT(*) FROM table1

Однако, выполнение дополнительного запроса можно избежать благодаря появившемуся в MySQL 4.0.0 параметру *SQL_CALC_FOUND_ROWS* оператора SELECT.

Например, выполняя запрос:

*SELECT SQL_CALC_FOUND_ROWS * FROM table1 LIMIT 40, 20*

MySQL подсчитает полное число строк, подходящих под условие запроса ***без учета ограничений, вызванных оператором LIMIT***, и сохранит это число в памяти. Сразу после выполнения запроса на выборку для получения количества записей нужно выполнить SELECT-запрос:

SELECT FOUND_ROWS ()

В результате MySQL без дополнительных вычислений вернет сохраненный в памяти параметр - полное количество строк в результате запроса.

```
$per_page=3; // количество записей, выводимых на странице
// получаем номер страницы
if (isset($_GET['page'])) $page=($_GET['page']-1); else $page=0;
$start=abs($page*$per_page); // вычисляем первый операнд для LIMIT
// выполняем запрос и выводим записи
$q="SELECT * FROM `table` ORDER BY field LIMIT $start,$per_page";
$res=mysql_query($q);
while($row=mysql_fetch_array($res)) {
  echo ++$start.". ".$row['field']."<br>\n";
}
// выводим ссылки на страницы:
$q="SELECT count(*) FROM `table`";
$res=mysql_query($q);
$row=mysql_fetch_row($res);
$total_rows=$row[0];
//Определяем общее количество страниц
$num_pages=ceil($total_rows/$per_page);
for($i=1;$i<=$num_pages;$i++) {
//текущую страницу выводим без ссылки
  if ($i-1 == $page) {
    echo $i." ";
  } else {
    echo '<a href="'.$_SERVER['PHP_SELF'].'?page='.$i.'">'.$i."</a> ";
  }
}
```

Работа PHP с СУБД MySQL

```
<?//запрос на выборку записей таблицы messages
    $sql="Select * from messages order by messages_date desc";
    $do_get = mysql_query($sql) or die("Ошибка выполнения запроса!!!".
mysql_error());
    $num_rows = @mysql_numrows($do_get);
    if($num_rows){
      while ($results = mysql_fetch_array($do_get)){
            echo
"<tr><td>$results[1]</td><td>$results[2]</td><td>$results[3]</td><td>$results[4]</td><t
d>$results[5]</td></tr>";
      }
    }else echo "<tr><td colspan=5>Сообщения отсутствуют в базе
данных.</td></tr>"
?>
</table>
```

ПРИМЕР

Расширение *mysqli*

Расширение *mysqli*, или как его еще называют *улучшенное (improved)* MySQL расширение, было разработано, чтобы дать возможность программистам в полной мере воспользоваться функционалом MySQL сервера версий 4.1.3 и выше и возможностями ООП. Расширение *mysqli* включается в поставку PHP версий 5 и выше.

mysqli имеет ряд преимуществ и усовершенствований по сравнению с *mysql*, которые заключаются в следующем:

- Объектно-ориентированный интерфейс
- Поддержка подготавливаемых запросов
- Поддержка мультизапросов
- Поддержка транзакций
- Улучшенные возможности отладки
- Поддержка встроенного сервера

Расширение mysqli предоставляет двойной интерфейс программисту. Поддерживаются как процедурная, так и объектно-ориентированная парадигмы программирования.

Процедурный интерфейс весьма схож с интерфейсом старого расширения, и во многих случаях функции отличаются только префиксом в имени. Некоторые mysqli функции принимают дескриптор соединения первым аргументом, в отличие от соответствующих им функций старого расширения, которые принимают его в качестве последнего необязательного аргумента.

Расширение *mysqli*

```
<?php
$mysqli = mysqli_connect("example.com", "user", "password", "database");
$res = mysqli_query($mysqli, "SELECT 'Пожалуйста, не используюте ' AS _msg FR
OM DUAL");
$row = mysqli_fetch_assoc($res);
echo $row['_msg'];

$mysql = mysql_connect("localhost", "root", "");
mysql_select_db("test");
$res = mysql_query("SELECT 'расширение mysql в новых проектах.' AS _msg FR
OM DUAL", $mysql);
$row = mysql_fetch_assoc($res);
echo $row['_msg'];
?>
```

Расширение *mysqli*

Объектно-ориентированный интерфейс предлагает функции сгруппированные по цели их применения, что облегчает их поиск и освоение.
Каких-либо принципиальных отличий в производительности между интерфейсами нет. Пользователи вольны в выборе интерфейса, основываясь на личных предпочтениях.

```
<?php
$mysqli = mysqli_connect("example.com", "user", "password", "database");
if (mysqli_connect_errno($mysqli)) {
    echo "Не удалось подключиться к MySQL: " . mysqli_connect_error();
}
$res = mysqli_query($mysqli, "SELECT 'A world full of ' AS _msg FROM DUAL");
$row = mysqli_fetch_assoc($res);
echo $row['_msg'];

$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Не удалось подключиться к MySQL: " . $mysqli->connect_error;
}
$res = $mysqli-
>query("SELECT 'choices to please everybody.' AS _msg FROM DUAL");
$row = $res->fetch_assoc();
echo $row['_msg'];
?>
```

Расширение *mysqli*

За выполнение запросов отвечают функции mysqli_query(), mysqli_real_query() и mysqli_multi_query(). Чаще всего применяется функция mysqli_query(), так как она выполняет сразу две задачи: выполняет запрос и буферизует на клиенте результат этого запроса (если он есть). Вызов mysqli_query() идентичен последовательному вызову функций mysqli_real_query() и mysqli_store_result().

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Не удалось подключиться к MySQL: (" . $mysqli-
>connect_errno . ") " . $mysqli->connect_error;
}

if (!$mysqli->query("DROP TABLE IF EXISTS test") ||
    !$mysqli->query("CREATE TABLE test(id INT)") ||
    !$mysqli->query("INSERT INTO test(id) VALUES (1)")) {
    echo "Не удалось создать таблицу: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

Расширение *mysqli*

Буферизация результатов запроса

После выполнения запроса его результаты можно целиком буферизовать на клиенте, либо читать построчно с сервера. Буферизация на клиенте позволяет серверу как можно быстрее освобождать занятые запросом ресурсы. Построчное же чтение и дальнейшая обработка результатов клиентом довольно медленный процесс. Поэтому рекомендуется использовать буферизацию результирующих наборов. Функция mysqli_query() совмещает в себе операции выполнения запроса и буферизации результирующего набора.

PHP приложения могут свободно оперировать данными внутри буферизованных результирующих наборов. Быстрая навигация по строкам наборов обусловлена тем, что наборы полностью располагаются в памяти клиента. Следует помнить, что зачастую обработка результатов на клиенте проще, нежели средствами сервера.

Расширение *mysqli*

```
<?php
$mysqli = new mysqli("example.com", "user", "password", "database");
if ($mysqli->connect_errno) {
    echo "Не удалось подключиться к MySQL: (" . $mysqli->connect_errno . ") « .
$mysqli->connect_error;
}

$res = $mysqli->query("SELECT id FROM test ORDER BY id ASC");

echo "Обратный порядок...\n";
for ($row_no = $res->num_rows - 1; $row_no >= 0; $row_no--) {
    $res->data_seek($row_no);
    $row = $res->fetch_assoc();
    echo " id = " . $row['id'] . "\n";
}

echo "Исходный порядок строк...\n";
$res->data_seek(0);
while ($row = $res->fetch_assoc()) {
    echo " id = " . $row['id'] . "\n";
}
?>
```

Расширение *mysqli*

Подготавливаемые запросы

СУБД MySQL поддерживает подготавливаемые запросы. Подготавливаемые (или параметризованные) запросы используются для повышения эффективности, когда один запрос выполняется многократно.

Выполнение подготавливаемого запроса проводится в два этапа: подготовка и исполнение. На этапе подготовки на сервер посылается шаблон запроса. Сервер выполняет синтаксическую проверку этого шаблона, строит план выполнения запроса и выделяет под него ресурсы.

MySQL сервер поддерживает неименованные, или позиционные, псевдопеременные *'?'*.

```
<?php
/* обычный запрос */
if (!$mysqli->query("DROP TABLE IF EXISTS test") || !$mysqli-
>query("CREATE TABLE test(id INT)")) {
    echo "Не удалось создать таблицу: (" . $mysqli->errno . ") " . $mysqli->error;
}
/* подготавливаемый запрос, первая стадия: подготовка */
if (!($stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)"))) {
    echo "Не удалось подготовить запрос: (" . $mysqli->errno . ") " . $mysqli->error;
}
?>
```

Расширение *mysqli*

За подготовкой идет выполнение. Во время запуска запроса клиент привязывает к псевдопеременным реальные значения и посылает их на сервер. Сервер, в свою очередь, подставляет их в шаблон и запускает уже готовый запрос на выполнение.

```
<?php
/* подготавливаемый запрос, вторая стадия: привязка и выполнение */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
    echo "Не удалось привязать параметры: (" . $stmt->errno . ") " . $stmt->error;
}
if (!$stmt->execute()) {
    echo "Не удалось выполнить запрос: (" . $stmt->errno . ") " . $stmt->error;
}
?>
```

Расширение *mysqli*

Повторное выполнение запроса

Подготовленный запрос можно запускать многократно. Перед каждым запуском значения привязанных переменных будут передаваться на сервер и подставляться в текст запроса. Сам текст запроса повторно не анализируется, равно как и не отсылается повторно шаблон.

```
<?php
 /* подготавливаемый запрос, первая стадия: подготовка */
if (!($stmt = $mysqli->prepare("INSERT INTO test(id) VALUES (?)"))) {
    echo "Не удалось подготовить запрос: (" . $mysqli->errno . ") " . $mysqli->error;
}
/* подготавливаемый запрос, вторая стадия: привязка и выполнение */
$id = 1;
if (!$stmt->bind_param("i", $id)) {
    echo "Не удалось привязать параметры: (" . $stmt->errno . ") " . $stmt->error;
}
if (!$stmt->execute()) {
    echo "Не удалось выполнить запрос: (" . $stmt->errno . ") " . $stmt->error;
}
/* повторные выполнения, на сервер передаются только значения переменных */
for ($id = 2; $id < 5; $id++) {
    if (!$stmt->execute()) {
        echo "Не удалось выполнить запрос: (" . $stmt->errno . ") " . $stmt->error;
    }
```

Механизм сессий в PHP

Сессии - это механизм, который позволяет создавать и использовать переменные, сохраняющие свое значение при переходе от одной страницы сайта к другой в течение всего времени работы пользователя с сайтом.
Задача идентификации пользователя в течение сеанса (или сессии) работы с сайтом решается путем присвоения каждому пользователю уникального номера, так называемого идентификатора сессии (SID, Session IDentifier).

Создание сессии

Первое, что нужно сделать для работы с сессиями (если они уже настроены администратором Web-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 вызова функции ***session_start();***

Регистрация переменных сессии

Для того, чтобы передавать и сохранять в течение сессии какие-либо переменные (например, логин и пароль), нужно просто зарегистрировать переменные в сессии. Это можно реализовать двумя способами:

Первый способ - с помощью функции **session_register**:

session_register(имя_переменной1, имя_переменной2, ...);

Второй способ зарегистрировать переменную - просто записать ее значение в ассоциативный массив **$_SESSION.**

Механизм сессий в PHP

```
<?  // создаем новую сессию или  восстанавливаем текущую
session_start();
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 {
   // регистрируем переменную login
   $_SESSION['login']=$_GET['login'];
   // регистрируем переменную passwd
   $_SESSION['passwd']=$_GET['passwd'];
// теперь логин и пароль - глобальные
// переменные для этой сессии
    if ($_GET['login']=="pit" &&   $_GET['passwd']=="123") {
        // перенаправляем на страницу  secret_info.php
        Header("Location: secret_info.php");
    }else echo "Неверный ввод,   попробуйте еще раз<br>";
}
?>
```

Механизм сессий в PHP

```
<?php
session_start();// создаем новую сессию или восстанавливаем текущую
?>
<html>
<head><title>Secret info</title></head>
<body>
<?
print_r($_SESSION);  // выводим все переменные сессии
?>
<p>Здесь размещена секретная информация.
</body>
</html>
```

ПРИМЕР

Механизм сессий в PHP

Чтобы избежать проникновения на данную страницу не авторизированного пользователя (который просто наберет в строке браузера адрес секретной странички secret_info.php), нужно добавить проверку логина и пароля из переменных сессии:

```
<?php
session_start();// создаем новую сессию или восстанавливаем текущую
// проверяем правильность  пароля-логина
if (!($_SESSION['login']==pit && $_SESSION['passwd']==123))
    Header("Location: authorize.php");
    // если авторизация неудачна, то переадресовываем пользователя на страницу авторизации
//Если авторизация прошла успешно выведется содержимое страницы
?>
<html>
<head><title>Secret info</title></head>
<body>
<?
print_r($_SESSION);  // выводим все переменные сессии
?>
<p>Здесь размещена секретная информация.
</body>
</html>
```

Удаление переменных сессии

Кроме возможности регистрировать переменные сессии (т.е. делать их глобальными на протяжении всего сеанса работы), полезно также иметь возможность 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 $_SESSION($HTTP_SESSION_VARS для версии PHP 4.0.6 и более ранних), то используют языковую конструкцию **unset().** Она не возвращает никакого значения, а просто уничтожает указанные переменные:

```
unset($_SESSION['login']).
```

Для того, чтобы уничтожить логин и пароль, введенные ранее, и убедиться, что они уничтожены, можно использовать следующий код:

```
<?
session_start();
unset($_SESSION[' passwd']);     // уничтожаем пароль
unset($_SESSION['login']);     // уничтожаем логин
print_r($_SESSION);    // выводим глобальные переменные сессии
?>
```

Механизм сессий в PHP

Уничтожить текущую сессию целиком можно командой **session_destroy();**

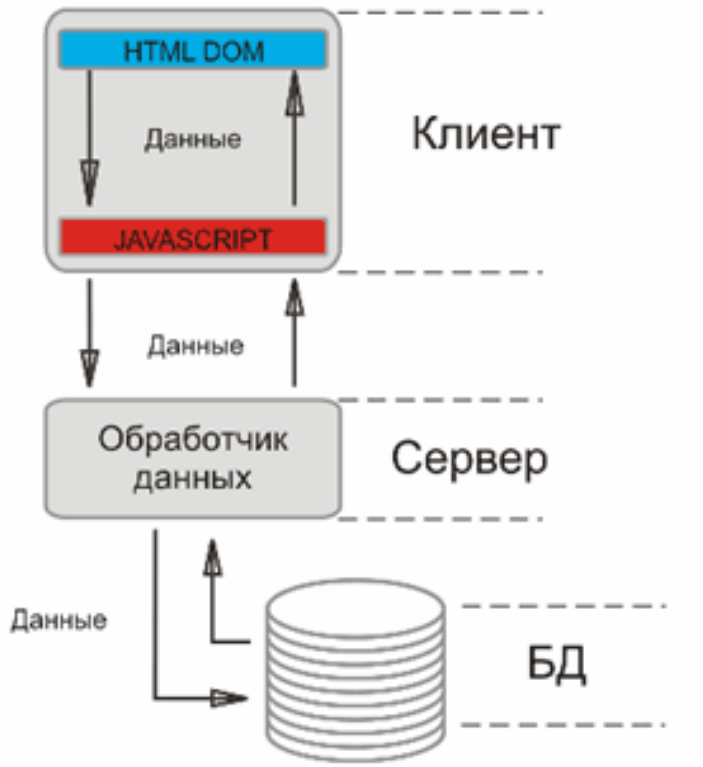
```
<?
session_start(); // инициализируем сессию
$test = "Переменная сессии";
$_SESSION['test']= $test;
// регистрируем переменную $test.
// если register_globals=on, то можно использовать  session_register('test');

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

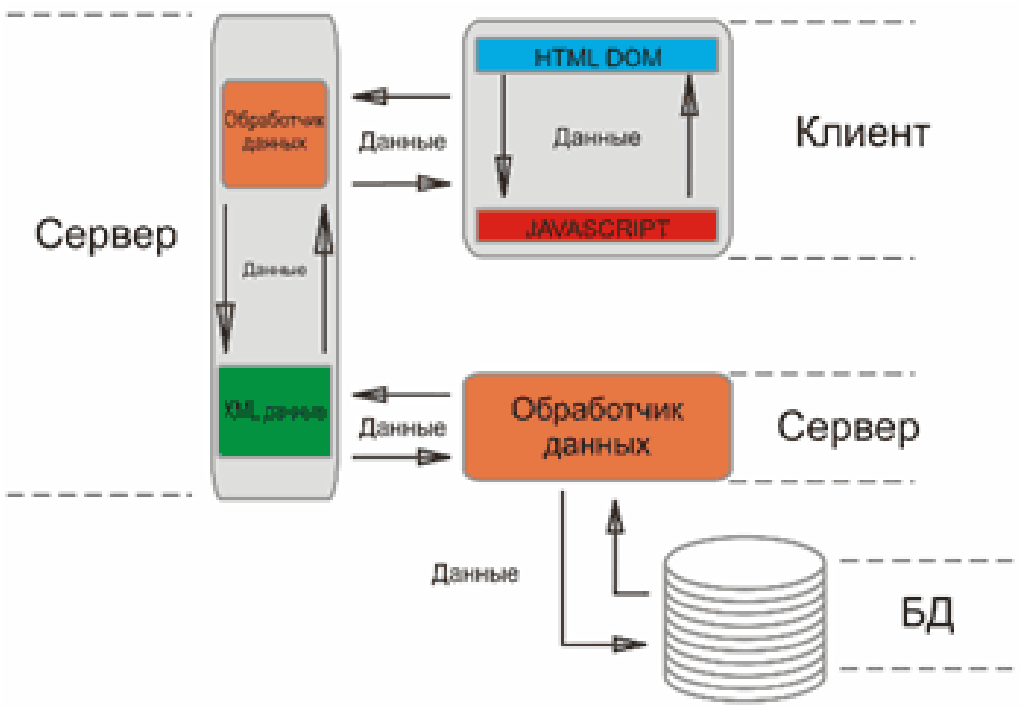
ПРИМЕР

Асинхронное взаимодействие с сервером. Технология AJAX

AJAX (Asynchronous Javascript and XML —асинхронный JavaScript и XML) — подход к построению интерактивных пользовательских интерфейсов web-приложений, заключающийся в асинхронном («фоновом») обмене данными между браузером и web-сервером без необходимости полной перезагрузки web-страницы.

В результате web-приложения становятся более быстрыми и удобными.





Асинхронное взаимодействие с сервером. Технология AJAX

AJAX не является самостоятельной технологией, а представляет собой методику использования нескольких смежных технологий: JavaScript, XMLHttpRequest; HTML, XML или JSON(JavaScript Object Notation); DOM.

AJAX базируется на двух основных принципах:

1. Использование одного из возможных вариантов динамического обращения к серверу без перезагрузки всей страницы, например:
   - с использованием ***XMLHttpRequest*** (набор API, используемый в языках JavaScript, JScript, VBScript для пересылки различных данных (XML, XHTML, JSON и т.д.) по HTTP-протоколу между браузером и веб-сервером);
   - через ***динамическое создание дочерних фреймов***;
   - через ***динамическое создание тега <script>***.
2. Использование DHTML для динамического изменения содержания страницы, для информирования пользователя о ходе обмена данными с сервером и отображения полученных данных.

В качестве формата передачи данных в AJAX, не смотря на наличие XML в аббревиатуре технологии, кроме XML можно использовать HTML или JSON(JavaScript Object Notation).

Варианты динамического обращения к серверу

1. XMLHttpRequest

Впервые XMLHttpRequest был разработан компанией Microsoft, появившись в компоненте Outlook Web Access программы Microsoft Exchange Server 2000. Он был назван IXMLHTTPRequest. Позднее, наработки были включены в состав MSXML 2.0(Microsoft XML) в виде объекта ActiveX, доступного через JScript, VBScript, или другие скриптовые языки, поддерживающиеся браузером. MSXML 2.0 был включён в состав браузера Internet Explorer 5.

Программисты проекта Mozilla разработали совместимую версию, называющуюся nsIXMLHttpRequest в Mozilla 0.6. Доступ к компоненту был реализован через JavaScript-объект, названный XMLHttpRequest. Полной функциональности удалось добиться только в Mozilla 1.0. В дальнейшем эта возможность также была реализована компаниями Apple начиная с Safari 1.2, родственным браузером Konqueror, компанией Opera Software начиная с Opera 8.01.

Методы объекта XMLHttpRequest

Метод	Описание
abort()	Отменяет текущий запрос, удаляет все заголовки, ставит текст ответа сервера в null.
getAllResponseHeaders()	Возвращает полный список HTTP-заголовков в виде строки. Заголовки разделяются знаками переноса (CR+LF). Если флаг ошибки равен true, возвращает пустую строку. Если статус 0 или 1, вызывает ошибку INVALID_STATE_ERR.
getResponseHeader(headerName)	Возвращает значение указанного заголовка. Если флаг ошибки равен true, возвращает null. Если заголовок не найден, возвращает null. Если статус 0 или 1, вызывает ошибку INVALID_STATE_ERR.
open(method, URL, async, userName, password)	Определяет метод, URL; async определяет, происходит ли работа в асинхронном режиме. Последние три параметра необязательны.
send(content)	Отправляет запрос на сервер.
setRequestHeader(label, value)	Добавляет HTTP-заголовок к запросу.
overrideMimeType(mimeType)	Позволяет указать mime-type документа, если сервер его не передал или передал неправильно. **Внимание**: метод отсутствует в Internet Explorer!

Свойства объекта XMLHttpRequest

Свойство	Тип	Описание
onreadystatechange	EventListener	Обработчик события, которое происходит при каждой смене состояния объекта. Имя должно быть записано в нижнем регистре.
readyState	unsigned short	Текущее состояние объекта (0 — не инициализирован, 1 — открыт, 2 — отправка данных, 3 — получение данных и 4 — данные загружены)
responseText	DOMString	Текст ответа на запрос. Если статус не 3 или 4, возвращает пустую строку.
responseXML	Document	Текст ответа на запрос в виде XML, который затем может быть обработан посредством DOM. Если статус не 4, возвращает null.
status	unsigned short	HTTP-статус в виде числа (404 — «Not Found», 200 — «OK» и т. д.)
statusText	DOMString	Статус в виде строки («Not Found», «OK» и т. д.). Если статус не распознан, браузер пользователя должен вызвать ошибку INVALID_STATE_ERR.

Варианты динамического обращения к серверу

Использование объекта XMLHttpRequest включает следующие этапы:

1. Создание экземпляра объекта XMLHttpRequest;
2. Установка обработчика события;
3. Открытие соединения и отправка запроса.

На стадии создания экземпляра объекта XMLHttpRequest необходима отдельная реализация для разных браузеров. Конструкция создания объекта отличается: в IE 5 - IE 6 она реализована через ActiveXObject, а в остальных браузерах (IE 7 и выше, Mozilla, Opera, Chrome, Netscape и Safari) — как встроенный объект типа XMLHttpRequest.

Вызов для ранних версий Internet Explorer выглядит так:

```
var req = new ActiveXObject("Microsoft.XMLHTTP");
```

В остальных браузерах:

```
var req = new XMLHttpRequest();
```

Для обеспечения кросс-браузерности кода, нужно проверять наличие объектов window.XMLHttpRequest и window.ActiveXObject:

```
if (window.XMLHttpRequest) {
    try {
        req = new XMLHttpRequest();
    } catch (e){}
} else if (window.ActiveXObject) {
    try {
        req = new ActiveXObject('Msxml2.XMLHTTP');
    } catch (e){
        try {
            req = new ActiveXObject('Microsoft.XMLHTTP');
        } catch (e){}
    }
}
```

Установка обработчика событий, открытие соединения и отправка запросов выглядят так:

req.open(<"GET"|"POST"|...>, <url>, <asyncFlag>);
req.onreadystatechange = processReqChange;

После определения всех параметров запроса его остается только отправить. Делается это функцией send():

```
req.send(null);
```

После этого начинает работать обработчик событий. Он — фактически основная часть программы. В обработчике обычно происходит перехват всех возможных кодов состояния запроса и вызов соответствующих действий, а также перехват возможных ошибок.

Варианты динамического обращения к серверу

```
function processReqChange()
{
  try {
     if (req.readyState == 4) {// только при состоянии "complete"
        // для статуса "OK"
        if (req.status == 200) {
            // обработка ответа – отображение полученных данных
        } else {
            alert("Не удалось получить данные:\n" +
                req.statusText);
        }
    }
  }
  catch( e ) {
     // alert('Caught Exception: ' + e.description);
     //
  }
}
```

ПРИМЕР

Создание плавающего фрейма iFrame

Альтернативный вариант обмена данными с сервером – **использование невидимого iFrame** (тег HTML <iFrame> - плавающий фрейм - позволяет встраивать в любое место документа фрейм без необходимости использования тега <frameset>). *Динамическое изменение URL* iFrame инициирует GET-запрос по указанному URL, таким образом, iFrame может использоваться для отправки запроса серверу.

Кроме того, iFrame может использоваться для отправки файлов на сервер. Для этого необходимо реализовать POST-запрос для формы, содержащей элемент ввода файла, предварительно установив в атрибут target формы имя iFrame(атрибут name). При этом iFrame выполнит POST-запрос по URL, указанному в форме.

Изменение URL iFrame для выполнения запроса к серверу рекомендуется осуществлять вызовом:

iframeDocument.location.src.replace(newURL).

Такой синтаксис в отличие от iframeDocument.location.src=..., не отражается на истории посещений(объект History) в большинстве браузеров, что делает запрос невидимым для посетителя.

Получение ссылки на документ, содержащийся в iFrame может зависеть от типа браузера: iframe.contentDocument, iframe.contentWindow.document, либо iframe.document:

```
// получить окно по тегу iFrame
function getIframeDocument(iframeNode) {
  if (iframeNode.contentDocument) return iframeNode.contentDocument
  if (iframeNode.contentWindow) return iframeNode.contentWindow.document
return iframeNode.document
}
```

Создание плавающего фрейма iFrame

Момент окончания получения ответа от сервера в iFrame для большинства браузеров может быть установлен с использованием обработчика события onLoad, вызывающегося после загрузки iFrame. В браузерах IE для этой цели следует использовать событие onreadystatechange, которое имеют все загружающиеся элементы(document, frame, frameSet, iframe, img, link, object, script):

```
if (navigator.userAgent.indexOf("MSIE") > -1 && !window.opera){
    iframe.onreadystatechange = function(){
        if (iframe.readyState == "complete"){
            alert("iframe загружен.");
        }
    };
} else {
    iframe.onload = function(){
        alert("iframe загружен.");
    };
}
```

ПРИМЕР

Создание <script>

Тот факт, что с использованием DOM возможно создавать динамические теги, дает возможность использования для обмена данными с сервером динамическое создание тега <script>.
Добавление нового <script> в документ заставляет браузер немедленно запрашивать скрипт по URL, указанному в свойстве src.

Следующая функция добавляет в head документа тег <script с нужным id и src:

```
function attachScript(id, src){
  var element = document.createElement("script")
  element.type = "text/javascript"
  element.src = src
  element.id = id
  document.getElementsByTagName("head")[0].appendChild(element)
}
```

Асинхронное взаимодействие с сервером. Технология AJAX

Варианты динамического обращения к серверу

Создание <script>

Вызов

attachScript('script1','translate.php?word=apple')

добавит в head документа тег:

<script src='translate.js?word=apple'></script>

Браузер тут же обработает тег: запросит translate.php и попытается выполнить возвращенный сервером код как JavaScript-программу .

После полной загрузки кода, для отображения полученных данных, в конце скрипта, возвращаемого сервером обычно выполняется вызов специальной функции.

Так как браузер получает готовый JavaScript код, то этот метод зачастую является самым лаконичным на стороне клиента.

Кроме того, по сравнению с методами XMLHttpRequest/iFrame он имеет одно важное преимущество: AJAX-запросы можно направлять не только к собственному, но и к любым другим серверам.

ПРИМЕР

Форматы передачи данных в AJAX

Формат HTML

В самом простом случае - ответом на AJAX-запрос является фрагмент HTML. Например:
* Файл успешно загружен на сервер *

Такой фрагмент может быть сразу отображен с использованием свойства innerHTML какого-либо узла DOM:
domObject.innerHTML = data

К преимуществам такого подхода относится отсутствие необходимости дополнительной обработки полученных данных.

В качестве недостатка можно отметить возможную избыточность при передаче - вместе с данными передается HTML-разметка.

Формат XML

В том случае, когда скрипт сервера получает и отправляет XML-данные, необходимо перед отправкой данных на клиенте преобразовать их в формат XML, а при получении XML-документа с сервера его необходимо обработать с использованием JavaScript и отобразить нужным образом.
Например, если пользователь набирает свое имя и адрес в какой-либо форме на своей Web-странице, вы можете получить из формы примерно такие данные:
firstName='Иван'
lastName='Иванов'
street='Ленина, 22'
city='Киев'
В формате XML эти данные могут выглядеть следующим образом:

```
<profile>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <street>Ленина, 22</street>
  <city>Киев</city>
</ profile>
```

Форматы передачи данных в AJAX

Формат XML

Таким образом, необходимо преобразовать данные формы в формат XML с использованием JavaScript, после чего они могут быть отправлены. В примере ниже отправка происходит с использованием объекта xmlHtttp типа XMLHttpRequest, проинициализированного ранее:

```
 var firstName = document.getElementById("firstName").value;
 var lastName = document.getElementById("lastName").value;
 var street = document.getElementById("street").value;
 var city = document.getElementById("city").value;
  var xmlString = "<profile>" +
   "  <firstName>" + escape(firstName) + "</firstName>" +
   "  <lastName>" + escape(lastName) + "</lastName>" +
   "  <street>" + escape(street) + "</street>" +
   "  <city>" + escape(city) + "</city>" +
   "</profile>";
var url = "saveAddress.php“; // URL для отправки
 xmlHttp.open("POST", url, true); // Откроем соединение с сервером (xmlHtttp типа XMLHttpRequest, проинициализирован ранее)
xmlHttp.setRequestHeader("Content-Type", "text/xml"); // Сообщим серверу, что посылаются данные в формате XML
   xmlHttp.onreadystatechange = confirmUpdate; // Установим функцию уведомления о завершении операции на сервере
 // Отправим данные
 xmlHttp.send(xmlString);
```

Форматы передачи данных в AJAX

Формат XML

Доступ к данным, отправленным методом POST не из формы, а с использованием объекта XMLHttpRequest возможен на сервере с помощью переменной *$HTTP_RAW_POST_DATA* или с использованием чтения входного потока PHP: *php://input.*

Для обработки полученных XML-данных на сервере возможно использование расширения PHP5 SimpleXML или расширения XML Parser Functions, встроенного в PHP по умолчанию. В простейшем случае возможно использование функций работы со строками для выделения данных из полученной XML строки.

ПРИМЕР

Обработка XML на стороне клиента

Обработка сформированных сервером XML-данных на стороне клиента возможна с использованием XML-анализаторов, встроенных во все современные браузеры. Имеются некоторые отличия между анализаторами в Microsoft и в других браузерах. Первый поддерживает как загрузку XML файлов, так и текстовых строк, содержащих XML код, в то время как в других браузерах используются раздельные анализаторы. При этом все анализаторы имеют функции для перемещения по дереву(DOM) XML документа, доступа, вставки и удаления узлов в дереве.

Асинхронное взаимодействие с сервером. Технология AJAX

Форматы передачи данных в AJAX

Формат XML

```
<html>
<body>
<script type="text/javascript">
try //Internet Explorer
  {
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  }
catch(e)
  {
  try //Firefox, Mozilla, Opera, etc.
    {
    xmlDoc=document.implementation.createDocument("","",null);
    }
  catch(e) {alert(e.message)}
  }
try
  {
  xmlDoc.async=false;
  xmlDoc.load("timetable.xml");
  document.write("xmlDoc is loaded, ready for use");
  }
catch(e) {alert(e.message)}
</script>
</body>
</html>
```

Асинхронное взаимодействие с сервером. Технология AJAX

Форматы передачи данных в AJAX

Формат XML

```
<html>
<body>
<script type="text/javascript">
text="<timetable><lesson><timeFrom>08.15</timeFrom><subject>Web</subject>
<teacher>Ovchinnikov</teacher</lesson></timetable>";

try //Internet Explorer
  {
  xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
  xmlDoc.async="false";
  xmlDoc.loadXML(text);
  }
catch(e)
  {
  try //Firefox, Mozilla, Opera, etc.
    {
    parser=new DOMParser();
    xmlDoc=parser.parseFromString(text,"text/xml");
    }
  catch(e) {alert(e.message)}
  }
document.write("xmlDoc is loaded, ready for use");
</script>
</body>
</html>
```

ПРИМЕР

Формат JSON

В простейшем случае JSON позволяет преобразовывать данные, представленные в объектах JavaScript, в строку, которую можно легко передавать от одной функции к другой или – в случае асинхронного приложения – от Web-клиента к серверной программе.

Преимущество использования JSON состоит в том, что этот формат можно легко интерпретировать в JavaScript.

По сути, JSON - это JavaScript-код, описывающий некую структуру данных. В нем используются две основные синтаксические конструкции:

```
// объявление массива:
var array = [ v1, v2, ... ];

// объявление ассоциативного массива:
var hash = { "key1" : v1, "key2" : v2, ... };
```

Форматы передачи данных в AJAX

Формат JSON

С их помощью можно описать структуру данных произвольной сложности. Например:

```
{
    "firstName": "Иван",
    "lastName": "Иванов",
    "address": {
        "street": "Ленина, 22",
        "city": "Киев",
    },
    "phoneNumbers": [
        "044 765-1234",
        "044 123-4567"
    ]
}
```

Если предположить, что вышеприведенный текст находится в переменной JSON_text, то для его преобразования в JavaScript объект достаточно всего лишь вызвать функцию eval, передав в нее переменную, JSON_text как показано ниже:

```
var p = eval("(" + JSON_text + ")");

div.innerHTML = p.firstName+" "+p.lastName+" живет в городе "+p.address.city;
```

Форматы передачи данных в AJAX

Формат JSON

Для работы на сервере с форматом JSON в PHP версии 5.2.0 и выше встроены функции json_encode, json_decode:

*string **json_encode** (mixed $value [, int $options = 0])*
*mixed **json_decode** (string $json [, bool $assoc = false [, int $depth = 512]])*

Например, для преобразования массива в JSON -строку:

```
<?php
$arr = array ('a'=>1,'b'=>2,'c'=>3,'d'=>4,'e'=>5);
echo json_encode($arr);//выведет {"a":1,"b":2,"c":3,"d":4,"e":5}
?>
```

Для обратного преобразования в объект:

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
var_dump(json_decode($json));
?>
```

А для преобразования в ассоциативный массив:

```
<?php
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
var_dump(json_decode($json,true));
?>
```

ПРИМЕР