

# Язык серверных сценариев PHP

Содержание данной дисциплины во многом опирается на *HTML*, *CSS*, язык написания клиентских сценариев *JavaScript*. Эти темы были рассмотрены в предыдущем курсе.

Для начала напомним технологию получения с Web-сервера HTML-файла. Хронология событий начинается с того, что пользователь работая на своем компьютере набирает в адресной строке окна браузера адрес необходимого Интернет-ресурса. Например, это может выглядеть так – *www.abcde.ru/st1.html*. Здесь предполагается название Web-сервера – *www.abcde.ru*, а имя запрашиваемого с этого сервера файла – *st1.html*. После этого запрос на получение данного файла по сети отправляется на указанный Web-сервер. В результате обработки данного запроса возможна ситуация, что указанного файла на сервере нет. В этом случае выводится сообщение об ошибке. Если же запрашиваемый файл существует, то он отправляется на пользовательский компьютер, с которого был сделан запрос. И после получения файла с Web-сервера браузер его отображает в окне в соответствии с имеющимися в нем тегами.

В случае запроса с Web-сервера не HTML, а *PHP-файла* картина несколько меняется. Во-первых, расширение файла содержащего программу на языке PHP должно быть *php*. По аналогии с только что рассмотренной ситуацией пример запроса к Web-серверу может выглядеть так – *www.abcad.ru/st2.php*. На сервере после получения подобного запроса имеющийся файл *st2.php* обрабатывает программа - интерпретатор *php.exe*.

Полное название PHP – *препроцессор гипертекста* (Hypertext Preprocessor). И этот препроцессор находится в файле *php.exe*.

После обработки препроцессором *st2.php* превращается в файл, содержащий необходимую информацию и теги (фактически это - обыкновенный *HTML*-файл). Далее он передается с сервера пользователю и в браузере просматривается как обыкновенный *HTML*-документ.

## ВНИМАНИЕ

Запрашиваемый php-файл располагается на Web-сервере, и там же предварительно обрабатывается перед отправкой пользователю.

Ранее мы сказали, в PHP-файле фрагменты HTML-кода могут перемешиваться с фрагментами на языке PHP. Для этого каждый фрагмент (даже если он единственный) программы на языке *php* должен отделяться тегами:

- `<?php` – начало фрагмента на PHP;
- `?>` - завершение фрагмента на PHP.

Вне подобного фрагмента размещаются обыкновенные конструкции на языке гипертекста HTML. Для иллюстрации на листинге 1 представлен пример простого файла с расширением *php*.

**Листинг 1.** Файл `index.php` с включением программы на языке PHP

```
<HTML> <HEAD>
<TITLE> Пример включения PHP - программы </TITLE>
</HEAD>
<BODY>
<?php
echo "Стартовая страница нового сайта";
?>
</BODY> </HTML>
```

Препроцессор PHP после получения запроса php-файла начинает такой запрос обрабатывать. Фактически обработка заключается в просмотре и построчном выполнении содержимого файла. Как только препроцессору PHP встречается открывающий тег фрагмента php-программы, то он начинает выполнять строки программы (последовательно строку за строкой).

Если в тексте встречается тег обозначающий завершение программы (сценария), то препроцессор опять переключается в режим просмотра кода. Описанные действия продолжаются до тех пор, пока не встретится очередной участок php-кода или не будет достигнут конец файла.

Возможна ситуация, когда весь файл целиком представляет *php-сценарий*. Однако чаще бывает по-другому – PHP-фрагменты перемежаются с фрагментами HTML-кода.

## ВНИМАНИЕ

В отличие от HTML-файлов пользователь не видит в окне браузера исходное содержимое PHP-файла. Поэтому в его содержимом может легко размещаться информация, которая должна быть закрыта от пользователя (например, пароли).

Комментарии в PHP могут быть однострочными и многострочными. В случае небольших комментариев занимающих одну строку удобнее воспользоваться двумя наклонными чертами:

```
// однострочный комментарий
```

Кроме того, аналогично синтаксису *JavaScript* можно определить фрагмент комментария из нескольких строк:

```
/*
```

```
многострочный комментарий
```

```
*/
```

Ранее на листинге 1 уже приводился файл с фрагментом *php-кода*. Этот файл (*index.php*) мы используем в качестве стартового файла данного созданного сайта, поэтому его текст следует набрать в *Блокноте* и сохранить под указанным именем в папке *www*.

После этого в адресной строке окна браузера необходимо набрать *www.newsait.ru*, т.е. мы обращаемся к нашему сайту по его названию (рис. 5) без указания конкретного файла. Однако с сайта получили файл *index.php*. Дело в том, что по умолчанию при обращении к сайту по его названию на пользовательский компьютер отправляется файл с именем *index*.

Хотя *php-программа* в тексте листинга 1 чисто символическая, ее необходимо прокомментировать. Часть текста занимает фрагмент на знакомом языке гипертекста HTML. Начиная с тега `<?php` начинается участок кода на PHP, который

завершается тегом `?>`. Содержательную часть программы составляет единственная команда вывода сообщения на экран:

```
echo " Стартовая страница нового сайта ";
```

В этом случае на экран выводится текст, который заключен в двойные кавычки.

#### ПРИМЕЧАНИЕ

В PHP двойные кавычки могут быть заменены на одинарные.

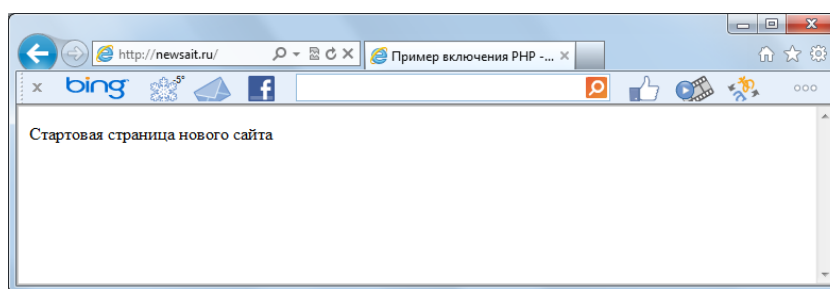


Рис. 5.

Обратите внимание, что каждая команда на языке PHP *заканчивается точкой с запятой*. Теперь, если мы решим посмотреть (рис. 6) в окне браузера HTML-код, то результат будет отличаться от исходного файла. Это как раз и есть *результат обработки* фрагмента php-кода препроцессором гипертекста.

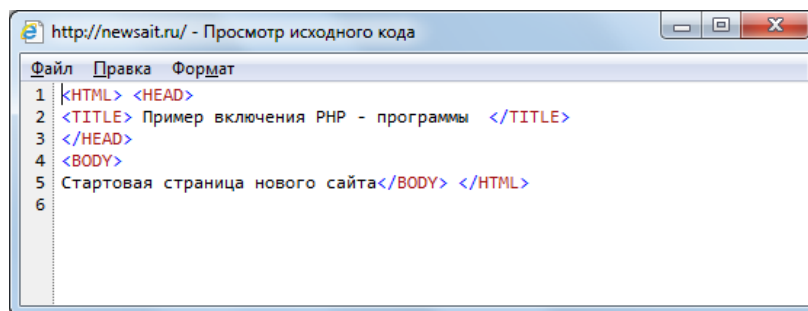


Рис. 6

## 4. Знакомство с конструкциями PHP

Как и в любом другом языке программирования в PHP имеются переменные. Их

имена обязательно должны начинаться со знака доллара (\$). Последующая часть имени может выбираться произвольно, но следует соблюдать правила:

- ❑ имя должно содержать только латинские буквы, цифры и символ подчеркивания;
- ❑ имя не должно начинаться с цифры.

Тип данных определяет, какой вид информации хранит переменная и какова область памяти отводимая под указанную переменную.

### ВНИМАНИЕ

Имя переменной чувствительно к регистру. Т.е. язык PHP является регистрозависимым. Например, переменные `$fam` и `$Fam` являются различными и под каждую из них отводится своя область памяти.

В PHP имеются разные типы переменных. В табл. 1. приведены типы данных, которыми мы будем пользоваться в данной главе.

**Таблица 1.** Простые типы данных

Названия типа данных	Обозначения в PHP	Комментарий
Строковый (текстовый)	<code>string</code>	Печатаемый символ или текст
Целочисленный	<code>integer</code>	Целое число
Вещественный	<code>float (double)</code>	Число с плавающей точкой
Логический	<code>bool</code>	Может принимать значения <code>true</code> или <code>false</code>

Тип данных переменной определяется PHP автоматически по ее значению. Так в строке

```
$soob3= "Дорогие, коллеги! Поздравляем с наступающим Новым Годом!";
```

переменной `$soob3` назначается строковый тип данных. На это указывает наличие кавычек.

## 5. Операторы PHP

Операторы позволяют произвести определенные действия с данными. Операторы делятся на категории. Так, к категории математических операторов относятся:

- ❑ + - сложение;
- ❑ - - вычитание;
- ❑ \* - умножение;
- ❑ / - деление;
- ❑ % - остаток от деления;
- ❑ ++ - инкремент (увеличение значения переменной на единицу);
- ❑ -- - декремент (уменьшение значения переменной на единицу).

Операторы инкремента и декремента могут использоваться соответственно в постфиксной или префиксной формах:

- ❑ постфиксная форма - `$perem++` и `$perem--`;
- ❑ префиксная форма - `++$perem` и `--$perem` ;

Разница здесь в том, что при постфиксной форме ( `$perem++` ) возвращается значение переменной перед операцией, а при префиксной вначале производится операция, а потом возвращается значение.

## 6. Преобразование типов данных

Интерпретатор PHP при выполнении действия над операндами относящимися к разным типам данных самостоятельно осуществляет преобразование типов. В тех ситуациях, когда мы сами собираемся управлять преобразованием переменных к необходимому типу данных, следует воспользоваться функцией `settype()`. Синтаксис ее использования выглядит следующим образом:

```
settype( переменная, тип )
```

В результате возвращаемое значение будет относиться к указанному типу.

Например,

```
$a=5;
```

```
$b=settype($a,"string");
```

Также существует еще одна (более удобная) форма преобразования типов. В этом случае перед переменной в круглых скобках необходимо указать необходимый тип данных. Важно заметить, что подобное преобразование типов не меняет тип исходной переменной. Пример на данную тему:

```
$a="123";  
  
$b=15 + (integer)$a;
```

## 7. Функции

**Функции** представляют собой некоторый участок кода с определенным именем, к которому можно многократно обращаться к программе. Особенность функции в том, что она всегда возвращает определенный результат в то место программы, откуда произошел ее вызов.

Функция может располагаться в файле, в котором произошел ее вызов, а может располагаться в совершенно другом месте. На листинге 2 представлен вариант создания таблицы с использованием вспомогательной функции. Отображение в окне браузера показано на рис. 7.

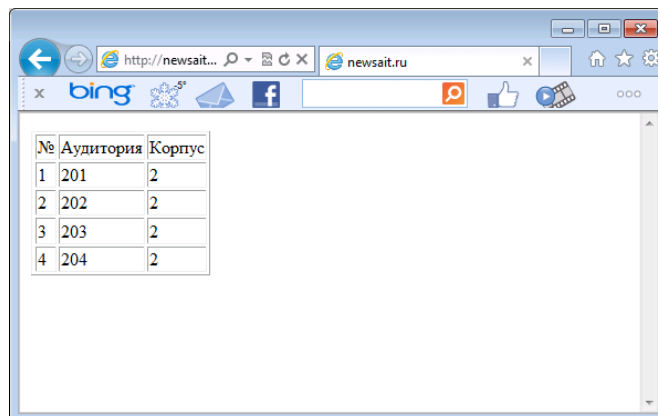
**Листинг 2.** Пример использования функции при формировании таблицы

```
<?php  
function ayd($i)  
{  
    $number="20".$i;  
    return $number;  
}  
  
echo "<TABLE border=1>";  
  
echo "<TR><TD>№</TD><TD>Аудитория</TD><TD>Корпус</TD></TR>";  
  
for( $i=1; $i < 5 ;$i++)  
{  
    echo "<TR><TD>".$i."</TD>";  
    echo "<TD>".ayd($i)."</TD>";  
    echo "<TD>2 </TD></TR>";  
}
```

```

    }
    echo "</TABLE>";
?>

```



№	Аудитория	Корпус
1	201	2
2	202	2
3	203	2
4	204	2

Рис.7

В данном случае результат, который возвращает функция, мы используем в качестве значений ячеек таблицы. В тех случаях, когда возвращаемый результат необходимо использовать в программе, применяют следующую конструкцию:

```
Переменная = имя функции();
```

В этом случае мы сохраняем значение выдаваемое функцией в переменной, что дает возможность использовать ее многократно в скрипте. Что касается примера на листинге 2, то там мы поступили наоборот. Это связано с тем, что использовали значение выдаваемое функцией только один раз.

В PHP множество стандартных функций и часто программисту не нужно разрабатывать свои алгоритмы. Достаточно найти одну или несколько стандартных функций реализующих необходимый функционал.

## 8. Работа со строками

Наиболее часто при обработке данных приходится производить манипуляции со строками. В PHP имеется большой набор функции работы со строками.

Так, для подсчета числа символов строки имеется функция `strlen()`. Например, в результате выполнения оператора



```
echo strlen("ИДО");
```

в окне браузера будет выведено число 3.

Для удаления из строки лишних пробелов в начале и конце строки следует воспользоваться функцией `trim()`. При этом пробелами считаются: пробел, символ перевода строки (`\n`), символ возврата каретки (`\r`), символы горизонтальной (`\t`) и вертикальной (`\v`) табуляции и символ конца строки.

## 9. Использование кавычек

В языке PHP поддерживаются оба типа кавычек и одинарные и двойные. Это необходимо для того, чтобы применять одни кавычки для обрамления других. Например, в этом случае с помощью команды `echo` можно вывести на экран строку с фрагментом, который необходимо заключить в кавычки. Листинг 3 иллюстрирует сказанное.

**Листинг 3.** Пример использования кавычек

```
<HTML> <HEAD>
<TITLE> Пример использования кавычек</TITLE>
</HEAD>
<BODY>
<?php
    echo "Книга '1С:Предприятие 8. Учимся программировать на примерах.'";
?>
</BODY></HTML>
```

В PHP функциональность кавычек разнообразна. Так, если поместить в двойные кавычки переменную, ее значение будет подставлено в текст (листинг 4).

**Листинг 4.** Пример использования подстановки переменной

```
<HTML> <HEAD>
<TITLE> Пример использования кавычек</TITLE>
</HEAD>
```

```

<BODY>

<?php

    $a = "Книга ";

    echo " $a '1С:Предприятие 8. Учимся программировать на примерах.'";

?>

</BODY></HTML>

```

Иногда необходимо использовать экранирование для размещения двойных кавычек в строке, которая обрамляется также двойными кавычками. В этом случае следует применять обратный слэш (/). Листинг 5 демонстрирует вывод двойных кавычек в окно браузера, за счет того, что их предваряет символ обратного слеша.

**Листинг 5.** Пример использования специальных символов

```

<HTML> <HEAD>

<TITLE> Пример использования кавычек</TITLE>

</HEAD>

<BODY>

<?php

    $a = " Книга ";

    echo "$a \"Офисные решения с использованием Microsoft Excel и VBA.\"";

?>

</BODY></HTML>

```

## 10. Работа с файлами

Часто требуется включать в скрипты необходимые файлы. Две инструкции `include()` и `require()` позволяют это сделать. В функциональном плане они практически одинаковы. Поэтому мы рассмотрим несколько примеров связанных с одной из них, а затем поясним отличие функций друг от друга. На листинге 6 представлен файл `a.php`, в котором определяются значения массива `$mass`. В другом PHP-файле (`b.php` на листинге 7) производится его подключение. Результат запроса к файлу `a.php` в окне браузера представлен на рис. 7.

**Листинг 6.** Подключаемый файл `a.php` (его выполнять не надо!)

```
<?php  
  
$mass[0]="Административный отдел";  
  
$mass[1]="Отдел сбыта";  
  
$mass[2]="Отдел закупок";  
  
$cols=3;  
  
?>
```

**Листинг 7.** Файл `b.php` (выполняемый файл)

```
<?php  
  
include ("a.php");  
  
for ( $i = 0 ; $i < $cols ; $i++)  
  
    echo $mass[$i]."<BR>";  
  
?>
```

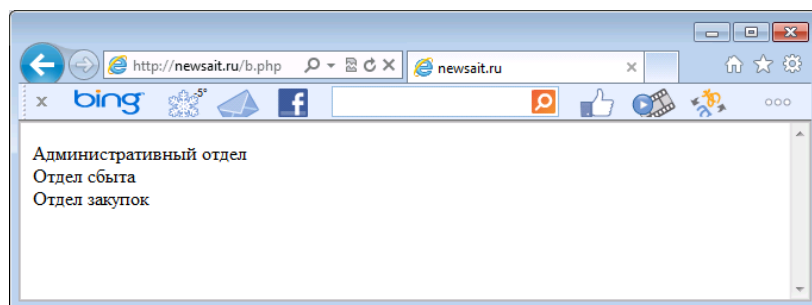


Рис.7

Включаемый функцией `include()` файл может быть не только PHP-скриптом, он может содержать простой текст.

Упомянутая выше функция `require()` отличается от `include()` реакцией на отсутствие подключаемого файла. Отсутствие файла указанного с помощью директивы `require()` приводит к остановке обрабатываемого скрипта. В аналогичной ситуации использование `include()` приводит к выводу на экран предупреждающего сообщения.

Существуют еще две функции аналогичные рассмотренным. Это

`include_once()` и `require_once()`. Они позволяют в документ подключить файл только один раз, даже если подключений будет несколько.

## Открытие файла

Любая работа, связанная с содержимым файла начинается с его открытия. Открытие файла производится с помощью функции `fopen()`, которая имеет следующий синтаксис:

```
int fopen(string filename, string mode)
```

Первый аргумент `filename` – относительный или абсолютный пункт к файлу. Второй аргумент задает режим работы (табл. 2) с открываемым файлом.

**Таблица 2.** Режимы работы с файлом

Значение	Комментарий
r	Открытие файла для чтения, после открытия указатель файла устанавливается в начало файла. Если файл не существует, то функция вернет <code>false</code> .
r+	Открытие файла для чтения и записи, после открытия указатель файла устанавливается в начало файла. Если файл не существует, то функция вернет <code>false</code> .
w	Создание нового пустого файла только для записи; если файл с таким именем уже есть, вся информация в нем уничтожается. После открытия файла указатель устанавливается на начало файла.
w+	Создание нового пустого файла только для чтения и записи; если файл с таким именем уже есть, вся информация в нем уничтожается.
a	Открытие файла для дополнения. Данные записываются в конец файла. Если файл не существует, то функция вернет <code>false</code> .
a+	Открытие файла для дополнения и чтения данных. Данные записываются в конец файла. Если файл не существует, то он будет создан.

После указания режима может следовать модификатор:

- ❑ `b` – файл будет открыт в бинарном режиме (по умолчанию);
- ❑ `t` – файл будет открыт в текстовом режиме.

В случае удачного открытия файла, функция `fopen()` возвращает дескриптор файла, а в случае неудачи – `false`.

#### ПРИМЕЧАНИЕ

Дескриптор файла представляет собой указатель на открытый файл, который используется операционной системой для поддержки операций с этим файлом и представляет уникальное число.

Возвращенный функцией дескриптор файла необходимо указывать во всех функциях, которые используются для работы с файлом.

Листинг 8 содержит пример создания файла `a.txt` в том каталоге из которого запущен скрипт.

#### Листинг 8. Создание файла `a.txt`

```
<?php
$fd = fopen("a.txt","w");
if (!$fd) exit("Ошибка открытия файла a.txt");
?>
```

Открытие двоичного файла, к примеру, рисунка, происходит таким же образом, только с флагом `b`.

Листинг 9 содержит пример открытия файла с рисунком `image.jpg` для чтения.

#### Листинг 9. Открытие двоичного файла `image.jpg` для чтения

```
<?php
$fd = fopen("image.jpg","rb");
if (!$fd) exit("Ошибка открытия файла");
?>
```

В рассмотренных примерах считается, что файлы создаются и открываются в том же каталоге, где расположен скрипт. Если же нужно открыть файл из произвольного каталога, то необходимо указать относительный путь.

При формировании относительных путей следует учитывать, что текущий каталог

обозначается одной точкой (.), а каталог, расположенный на один уровень выше, обозначается двумя точками (..). Например, если скрипт расположен в каталоге `C:/main/test/`, а файл необходимо создать в каталоге `C:/main/files/`, то скрипт должен выглядеть так, как показано на листинге 11. В этом случае мы поднимаемся на один каталог вверх, а затем опускаемся в каталог `files`.

**Листинг 11.** Открытие файла с указанием относительного пути

```
<?php
$fd = fopen("../files/file.txt","w");
if (!$fd) exit("Ошибка открытия файла a.txt");
?>
```

## Заккрытие файла

После того как работа с файлом завершена, его необходимо закрыть. Заккрытие файла осуществляется с помощью функции `fclose()`, которая имеет следующий синтаксис

```
int fclose(int fd);
```

Аргумент `fd` представляет собой дескриптор файла, который необходимо закрыть.

Если скрипт не закрывает файл, то его закрытие производится автоматически при завершении работы скрипта. По возможности следует закрывать файл сразу же, как с ним прекращена работа, т. к. в то время когда файл открыт, с ним не может работать другой скрипт. Листинг 12 содержит пример закрытия файла.

**Листинг 12.** Заккрытие файла

```
<?php
$fd = fopen("file.txt","w");
if (!$fd) exit("Ошибка открытия файла");
fclose($file);
?>
```

## Запись в файл

Рассмотрим теперь как происходит запись информации в файл. Для этого следует использовать практически идентичные функции:

```
fputs (дескриптор файла, строка, [длина строки]),  
fwrite (дескриптор файла, строка, [длина строки]).
```

Первый аргумент представляет собой дескриптор файла, который возвращается функцией `fopen()`. Второй аргумент является строкой, которая должна быть записана в файл. В этих функциях возможен третий аргумент, который является необязательным и задает количество символов в строке, которые должны быть записаны. Если третий аргумент не указан, записывается вся строка.

На листинге 13 приведен пример открытия файла, записи в него информации и последующего закрытия. Запустите этот скрипт и посмотрите на присутствие в том же каталоге файла `prim1.txt` и его содержимое.

#### Листинг 13. Запись в файл строки

```
<?php  
$file = fopen("prim1.txt","w");  
fputs($file,"Пример работы с файлами");  
fclose($file);  
?>
```

#### Чтение из файла

Чтение содержимого файла можно осуществить при помощи `fread()`, которая имеет следующий синтаксис:

```
string fread (дескриптор файла, длина в байтах).
```

Эта функция принимает в качестве первого аргумента дескриптор открытого файла, а в качестве второго длину строки, которую требуется прочитать из файла (в байтах).

Для чтения всего файла целиком, часто прибегают к функции `fsize()`, которая возвращает число байтов, содержащихся в файле, имя которого передается функции в качестве аргумента.

#### Листинг 14. Чтение из файла

```
<?php
$filename="prim1.txt";
// Открываем файл для чтения
$file = fopen($filename,"r");
// Читаем содержимое файла в переменную $buffer
$buffer =fread($file,filesize($filename));
// Закрываем файл
fclose($file);
echo $buffer;
?>
```

Следующий скрипт на листинге 15 позволяет открыть существующий файл и прочесть из него первые 7 символов.

#### Листинг 15. Чтение из файла 7-ми символов

```
<?php
$file = fopen("prim1.txt","r");
$inform=fread($file,7);
fclose($file);
echo $inform;
?>
```

Для извлечения информации из файла также используется функция

```
fgets (дескриптор) ,
```

которая позволяет считывать из файла по одной строке за раз. Считывание будет выполняться до тех пор, пока не встретится символ новой строки ( `/n`) или символ конца файла.

Иногда удобнее прочитать файл посимвольно. В этом случае необходимо контролировать – достигнут ли конец файла? Для этого предназначена функция

```
feof (дескриптор) .
```

В случае если достигнут конец файла, то функция возвращает `true`. Рассмотрим



пример посимвольного чтения информации из файла (листинг 16).

**Листинг 16.** Посимвольное чтение из файла

```
<?php
$file = fopen("prim1.txt","r");
$s="";
while (!feof($file))
{
    $s=$s.fread($file,1);
}
echo $s;
?>
```

Рассмотрим пример построчного чтения информации из файла (листинг 17).

**Листинг 17.** Построчное чтение из файла

```
<?php
$file = fopen("prim1.txt","r");
while (!feof($file))
{
    echo fgets($file);
}
fclose($file);
?>
```

Здесь для чтения информации из файла нам пришлось использовать несколько строк кода. Вместо этого можно воспользоваться функцией `file()`, которая читает файл в массив. Каждый элемент массива — это прочитанная строка из файла. Пример организации построчного чтения из файла:

```
$s=file('file.txt');
```

## 11. Формы

Путешествуя по Web-сайтам вы неоднократно сталкивались с формами. Любая

авторизация доступа или внесение сведений в электронную анкету требует заполнения формы. Основное назначение форм *заключается в организации передачи информации от браузера пользовательского компьютера на Web-сервер.*

#### ПРИМЕЧАНИЕ

Включение формы делает страничку интерактивной. Фактически в этом случае пользователю обеспечивается диалог с Web-сервером. Посетитель вводит запрашиваемую информацию. Это могут быть - фамилия, адрес, банковские реквизиты.

Важно заметить, что форма является элементом языка *HTML*, а не *PHP*. Технология *PHP* предоставляет только механизм работы. В дальнейшем тексте мы практически в каждой разработке будем использовать формы.

Форма в HTML-документе реализуется *тегом-контейнером* `<FORM>`, в котором задаются необходимые управляющие элементы - поля ввода, кнопки и т.д.

Если управляющие элементы указаны *вне содержимого тега* `FORM`, то они не создают форму, а используются для построения пользовательского интерфейса на HTML-странице.

Обработка элементов формы производится с помощью скриптов (PHP-сценариев). Имена элементам формы присваиваются через их атрибут `name`.

На листинге 19 представлен файл формы (*19.html*). Это обыкновенный HTML-файл без скриптов и в окне браузера он приводит к отображению формы (рис. 10). Здесь перед пользователем открывается поле ввода и кнопка с помощью которой информация от элементов формы отправляется на сервер.

**Листинг 19.** Файл 19.html, обеспечивающий Web-страницу на рис. 10

```
<HTML> <HEAD><TITLE> Форма </TITLE>
</HEAD><BODY>

<FORM action = "20.php">

<P>Ваша фамилия:

    <INPUT type = "text" name = "fio" value="" title="Поле для фамилии">
```

```

</P>

<INPUT type="submit" value="Передать данные на сервер"

        name="OK" title="Отправка информации на сервер">

</FORM>

</BODY></HTML>

```

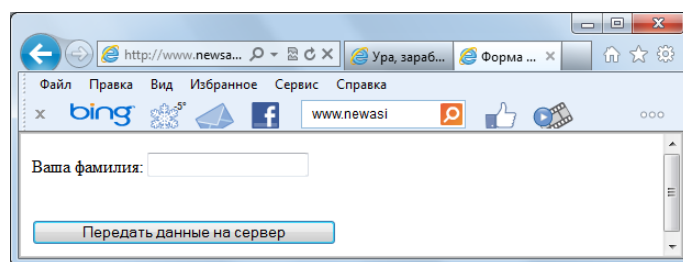


Рис. 10.

Начнем рассмотрение листинга 19 с ключевого тега – `<FORM>`. Фактически он объявляет форму. Для окончания объявленной формы существует соответствующий закрывающий тег `</FORM>`. У тега `<FORM>` имеется несколько параметров из которых на листинге 19 мы использовали только один. Отметим только, что задать имя формы можно с помощью еще одного параметра – `name`. В большинстве случаев этот параметр устанавливать не требуется, и в наших примерах мы так и будем поступать.

В примере на листинге 19 указано значение параметра `action`

```
action = "20.php",
```

которое представляет имя программы (скрипта). Эта программа будет обрабатывать данные на Web-сервере полученные от элементов формы.

#### ПРИМЕЧАНИЕ

Отправка формы – это обычный запрос к серверу, только вместе с запросом передается еще и дополнительная информация – данные от формы.

Если параметр `action` не указать, то данные будут переданы текущей программе (самой себе). Она будет выполнена на сервера повторно, но уже с принятыми значениями от элементов формы. Такая ситуация достаточно распространена.

Еще один параметр – `method`, и он определяет способ отправки формы на сервер. Этих способов два – `GET` и `POST`, различия между ними мы рассмотрим позже. Заметим только, что если способ отправки не указан, то предполагается использование `GET`.

Вернемся к рис. 10. Если пользователь внес информацию в текстовое поле и нажал на кнопку `Передать данные сервер`, то к указанному в разделе `action` скрипту `20.php` произойдет обращение на сервере.

Как уже сказано, основное назначение формы заключается в том, чтобы быть контейнером для различных элементов. Наиболее распространенным элементом является *тестовое поле* (его также часто называют – *поле для ввода*). На нашей форме этот элемент объявлен следующим образом:

```
<INPUT type = "text" name = "fio" value=""  
title="Поле для фамилии">.
```

Значение параметра `type` равно `text` как раз и определяет тип элемента – *поле для ввода*. Параметр `name` – *имя элемента*, и когда информация будет отправляться на сервер, то данные от этого элемента передаются в виде:

Значение `name` = Значение `value`.

Указанный здесь параметр `value` определяет текст, который будет отображаться в поле ввода. Еще один параметр `title` – это всплывающая подсказка, которая будет отображаться при наведении мыши на поле для ввода.

Имеется еще один параметр, который в приведенном примере не указан:

`size` - определяет длину поля для ввода (в качестве значения данного параметра необходимо указать число).

В качестве альтернативы объявлению поля ввода на листинге 6.10 можно использовать более информативную запись:

```
<INPUT type = "text" name = "fio"  
title="Поле для фамилии" value="Здесь необходимо ввести  
свою фамилию" size=40>
```

На листинге 19 расположен еще один элемент управления – *кнопка* с помощью которой данные отправляются на сервер. Программно *кнопка* создается также с использованием тега `<INPUT>`, но в качестве значения параметра `type` следует указать `submit`. В рассматриваемом примере мы использовали полный набор параметров тега определяющего на форме кнопку:

```
<INPUT type="submit" value="Передать данные на сервер"
       name="OK" title="Отправка информации на сервер">
```

Рассмотрим, что представляет собой передача данных от элементов формы скрипту на сервере - скрипту `20.php`.

Для того, чтобы другая программа могла получить данные формы можно использовать один двух методов:

- ❑ `GET` – в этом случае данные будут передаваться присоединенными к имени программы указанной в параметре `action`;
- ❑ `POST` – данные в этом случае будут отправляться незаметно для пользователя.

В рассмотренном примере мы не указывали метод передачи данных скрипту на сервере:

```
<FORM action = "20.php">
```

В этом случае вступает правило – по умолчанию считается, что данные передаются методом `GET`.

На листинге 20 приведен пример скрипта, который обрабатывает данные полученные от формы на рис. 10.

**Листинг 20.** Файл `20.php` для обработки файла с формой

```
<html><body>
<?php
echo "Добрый день,   ".$_GET['fio']."!";
?>
</body></html>
```

В данном тексте учтено, что для имени поля ввода было использовано `fio`. В связи с этим значение, которое ввел пользователь на сервере можно получить с помощью:

```
$_GET['fio']
```

Результат выполнения данного скрипта показан на рис. 11. Если взглянуть на адресную строку, то мы увидим, что к имени скрипта добавляется строка символов с передаваемой информацией. Если вы в Интернете увидите подобную строку (содержащую знак вопроса), то перед вами передача параметров методом `GET`. Как раз знак вопроса (?) после имени переменной означает начало передачи данных методом `GET`. После него следуют пары `имя=значение`, разделенные символом амперсанда (&). В обрабатывающем скрипте доступ к `GET`-параметрам можно получить через суперглобальный массив `$_GET`, указав в квадратных скобках после него имя параметра.

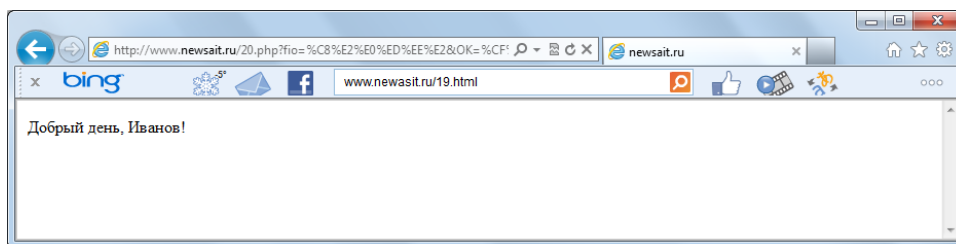


Рис. 11.

Передача методом `GET` часто не удобна по ряду причин:

- ❑ пользователь видит в адресной строке параметры и при желании может изменить их значения;
- ❑ объем передаваемой информации ограничен.

Чтобы сервер мог данные отличить, они присоединяются по специальным правилам. Сначала после имени программы-обработчика ставится знак вопроса (?), что указывает передачу данных. После этого указывается имя первого объявленного элемента на форме (в нашем случае это текстовое окно), далее знак равенства и значение. Если элемент один, то формирование запроса считается оконченным. После этого он отправляется на сервер. Если же элементов несколько, то информация о следующем отделяется от предыдущего знаком &. Формат записи очередного

элемента строится по описанной схеме: имя элемента, знак равно, значение элемента.

## ПРИМЕЧАНИЕ

Стоит отметить, что русские буквы передаются в закодированном виде, так что вы не увидите русских букв – они будут заменены кодами.

Поясним теперь в чем отличительная особенность метода `POST`. Для того, чтобы использовать данный метод передачи необходимо указать у формы в качестве значения параметра `action` вариант `POST`. В этом случае для извлечения значений параметров формы следует использовать суперглобальный массив `$_POST`.

Для иллюстрации данного метода передачи изменим файл с формой (новый вариант на листинге 21) и скрипт обработки формы на сервере (листинг 22).

### Листинг 21. Файл `21.html` для формы

```
<HTML> <HEAD><TITLE> Форма </TITLE>
</HEAD><BODY>

<FORM method="POST" action = "22.php">

Ваша фамилия:

<INPUT type = "text" name = "fio" value="" title="Поле для фамилии">

<BR> <BR> <BR>

<INPUT type="submit" value="Передать данные на сервер"
        name="OK" title="Отправка информации на сервер">

</FORM>

</BODY></HTML>
```

### Листинг 22. Файл `22.php` обработки данных от формы

```
<html><body>

<?php

echo "Добрый день,    ".$_POST['fio']. "!";

?>

</body></html>
```

Как видно (рис. 12) теперь в адресной строке нет никаких параметров - при использовании метода `POST` данные отправляются незаметно для пользователя в теле запроса.

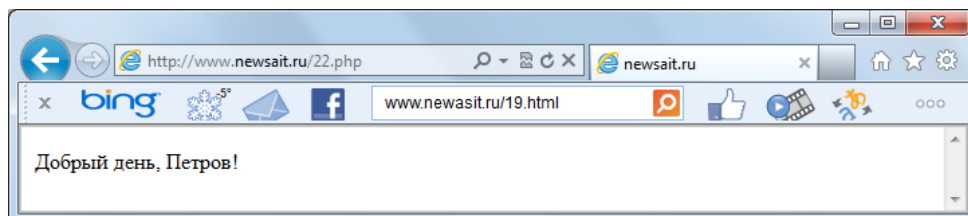


Рис. 12.

## 12. Элемент формы для пароля

В плане работы с формой мы использовали текстовое окно и кнопку, но ими ассортимент элементов не исчерпывается. Здесь мы приведем примеры использования других элементов в программе. Начнем с поля для ввода пароля, которое объявляется точно также как и поле ввода. Только в параметре `type` необходимо указать значение `password`. Например, следующая запись:

```
<INPUT type="password"
      name="pass" title="Поле для ввода имени">
```

создает на форме поле для ввода пароля (все вводимые символы будут выглядеть звездочками). Продемонстрируем сказанное на примере. На листинге 23 представлен файл, который позволяет создать на странице форму с необходимостью указания пароля (рис. 13).

**Листинг 23.** Файл 23.html, обеспечивающий форму с указанием пароля

```
<HTML> <BODY>

<FORM ACTION = "24.php">

Введите пароль:

<INPUT type = "password" name = "pass"
      value="" title="Необходимо ввести пароль">
```



```

<INPUT TYPE ="submit" value="Отправка значения пароля на сервер"
      name="OK">
</FORM> </BODY></HTML>

```

В этом случае вводимый пользователем текст будет скрыт от взгляда посторонних, а отправляемый серверу введенный пароль проверяется на соответствие истинному (условно в качестве правильного пароля принята комбинация **123**). Файл, обрабатывающий данный запрос представлен на листинге 24.

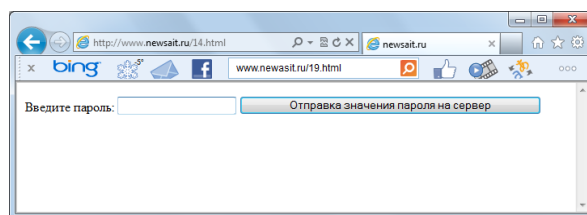


Рис. 13.

**Листинг 24.** Скрипт 24.php, обрабатывающий форму на рис. 13

```

<HTML><BODY>

<?php

if ($_GET['pass'] == "123")

    echo "Привет! Вы прошли авторизацию.";

else

    echo "Ошибка при указании пароля.";

?>

</BODY></HTML>

```

Результат работы скрипта при правильном указании пароля продемонстрирован на рис. 14.

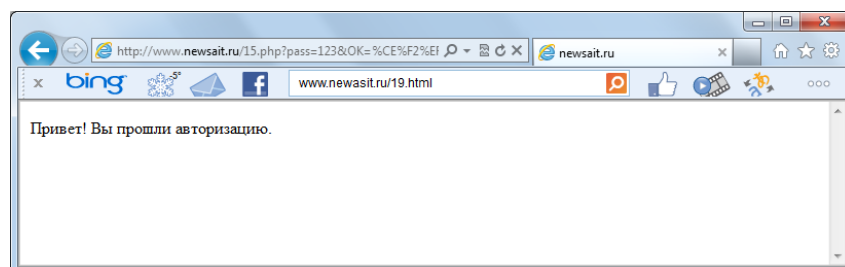


Рис. 14.

## Скрытое поле

Важный элемент управления – *скрытое поле*. Данный элемент нельзя увидеть и его назначение заключается в передаче данных, которые не должны изменяться пользователем.

Скрытое поле создается следующим образом:

```
<input name="имя" type="hidden" value="значение">
```

## 13. Форма для регистрации вопросов на сайте

В этом разделе мы рассмотрим создание формы, которая позволяет посетителям сайта оставить вопрос администратору, либо другому руководящему сотруднику. Разумеется все это можно сделать и с помощью средств электронной почты, однако рассматриваемый механизм интереснее и кроме того он поможет нам попрактиковаться в PHP.

На рис. 15 представлена форма, которую нам следует реализовать. Здесь расположены три поля ввода и кнопка для отправки сведений на сервер. Для формирования данных элементов на странице формы нам потребуется разработать HTML-файл (листинг 25).

Подобная форма регистрации посетителя представляет механизм общения посетителя сайта с его группой администрирования.

**Листинг 25.** Файл для регистрации посетителя 25.html

```
<form action="26.php" method="post">
```

```

Ваша фамилия и имя <br> <input type="text" name="fam"><br>
Адрес электронной почты или
контактный телефон <br> <input type="text" name="adr">
<br> Сообщение:<br> <br>
<textarea name="info" cols=10 rows=5>
</textarea> <br>
<input type="submit" name="ok" value="OK">
</form>

```

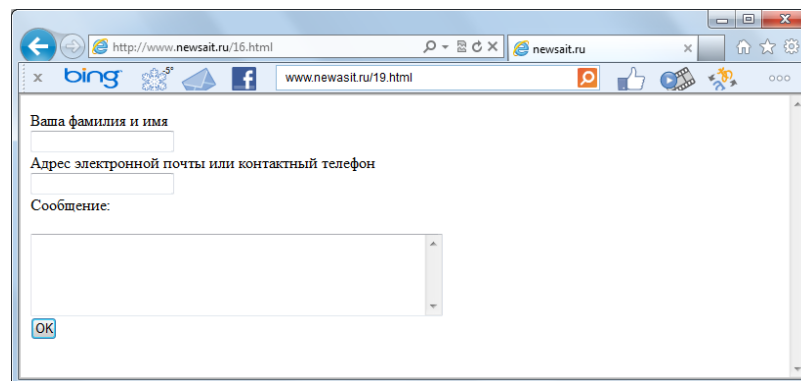


Рис. 15.

В заголовке формы

```
<form action="26.php" method="post">
```

указано название скрипта, который должен обрабатывать данные формы. Этот скрипт приведен на листинге 26. В нем в первую очередь проверяется наличие в полученных сведений информации от элемента с именем `ok` – запущен ли скрипт в результате щелчка по кнопке с указанным именем. Только при положительном ответе на данный вопрос производятся действия по открытию файла и внесения в него информации.

В результате открытия файла

```
$a=fopen("info.txt","at")
```

в переменную `$a` записывается дескриптор открытого файла. В дальнейшем вся

работа происходит через данную переменную и нам не требуется больше указывать имя файла и режим работы с ним. В следующей строке производится блокировка файла

```
flock($a,2)
```

Благодаря этому другой посетитель не сможет внести изменения в данный файл. Далее в файл с помощью функции `fputs()` последовательно записывается информация о фамилии, адресе и самом сообщении. В результате в файл `info.txt` вносится информация (рис. 16) от очередного посетителя. После этого осуществляется разблокировка файла:

```
flock($a,3).
```

**Листинг 26.** Файл `26.php` для обработки данных формы на рис. 15

```
<?php
if (isset($_POST['ok']))
{
    $a=fopen("info.txt","at");
    flock($a,2);
    fputs($a,$_POST['fam']."\n");
    fputs($a,$_POST['adr']."\n");
    fputs($a,$_POST['info']."\n");
    flock($a,3);
    fclose($a);
    echo "Информация внесена в файл";
}
?>
```

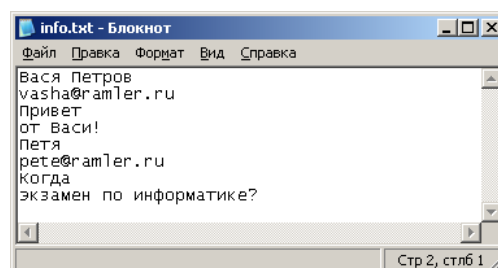


Рис. 16.

## 14. Установка паролей

Один из наиболее важных моментов связан с авторизацией – проверки прав доступа. Рассмотрим следующую форму (рис. 17), где располагается два поля ввода и кнопка для отправки сведений на сервер. Для ее реализации потребуется файл, приведенный на листинге 27.

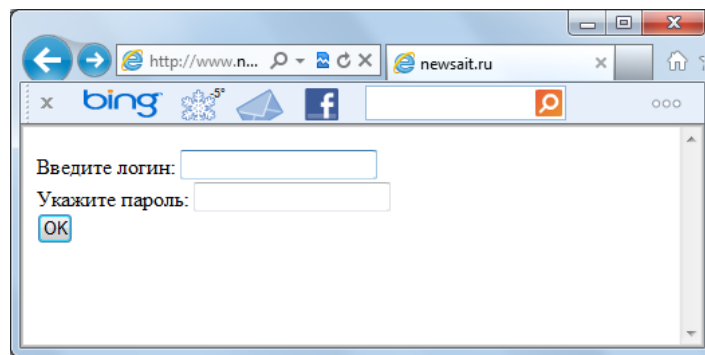


Рис. 17.

**Листинг 27.** Файл, обеспечивающий форму, показанную на рис. 17

```
<FORM method="GET" ACTION = "28.php">  
  
Введите логин:  
  
<INPUT type = "text" name = "login" value=""><BR>  
  
Укажите пароль:  
  
<INPUT type = "password" name = "pass"  
value="" ><BR>  
  
<INPUT TYPE = "submit" name="OK" value="OK">  
  
</FORM> </BODY></HTML>
```

Скрипт 28.php, обрабатывающий сведения от формы представлен на листинге 28.

**Листинг 28.** Скрипт 28.php, обрабатывающий сведения формы на рис. 17

```
<HTML><BODY>
```

```
<?php

if (($_GET['pass'] == "1") && ($_GET['login'] == "5"))

    echo "Привет! Вы прошли авторизацию.";

else

    echo "Ошибка при указании пароля или логина";

?>

</BODY></HTML>
```