# Лабораторная № 9

# 10. Работа с формами

В веб-пространстве для передачи данных от посетителя (пользователя) веб-страницы на сервер используются HTML-формы В языке PHP предусмотрены многочисленные средства для работы с формами.

## Программа для обработки ввода пользователя

До некоторого времени мы будем разделять HTML-текст и текст PHP-программ. В листинге 10.1 приведен пример простой HTML-формы.

### Листинг 10.1. Простая HTML-форма

```
<html> <head>
<title> Листинг 10-1. Простая HTML-форма </title>
</head> <body>
<!-- Форма для задания цвета через переменную $bg.
Ее использование — в скрипте ls10-2.php -->
<p><b>Выберите цвет:</b>
<form action="ls10-2.php" method="GET">
<p><input type="radio" name="bg" value="red">red
<p><input type="radio" name="bg" value="green">green
<p><input type="radio" name="bg" value="blue">blue
<p><input type="submit" value="Выполнить">
</form>
</body> </html>
```

Мы создали форму, в которой есть набор кнопок-переключателей (radio button) с общим именем `"bg"` и кнопка передачи данных `"submit"`. Атрибут `action` тега `form` указывает на файл `ls10-2.php`, следовательно этот файл будет обрабатывать данные формы. Поскольку мы указали здесь только имя файла, а путь к нему опустили, то файл должен находиться в том же каталоге на сервере, что и файл с программой 10.1.

Обратите внимание, как передается значение переменной в методе **`GET`** (т.е. что пишется в поле «Адрес» браузера при нажатии кнопки «Выполнить») — это строка вида `ls10-2.php?bg=red`

т.е. после имени выполняемого скрипта идет знак вопроса, затем имя передаваемой переменной, знак «равно» и значение этой переменной.

Прежде чем перейти к листингу 10.2, содержащему php-скрипт, обрабатывающий данные формы из листинга 10.1, необходимо пояснить о системе безопасности в PHP средствами суперглобальных массивов.

## Безопасность средствами суперглобальных массивов в PHP

Для того, чтобы понять почему так важны суперглобальные массивы (Superglobals) в PHP нам необходимо разобраться с тем, как передаются данные от одной веб-страницы к другой. Существуют, по крайней мере, два способа передачи данных, известные как GET и POST.

Вот пример формы в HTML:

```
<form name="form1" method="post" action="process.php">
<p>Пожалуйста, введите Ваше имя:</p>
<p><input type="text" name="yourname"></p>
<p><input type="button" name="Submit" value="Submit">
</p>
</form>
```

Эта форма запрашивает некоторые сведения у пользователя и затем передает полученные данные скрипту `process.php`. Критическая точка здесь — это объявление метода, которым будут переданы данные.

## GET

Метод **GET**, передает имена переменных и их значения через заголовок URL адреса, добавляя их в конец URL. Такой метод передачи данных ограничен максимально возможной длиной строки URL, которая составляет 255 символов. Если объем данных превышает это значение, то все что находится за пределами верхней границы будет попросту потеряно. Более того, даже на SSL соединениях данные не шифруются, поскольку они являются частью веб-адреса.

Ниже приводится пример формы, которая могла бы быть размещена на веб-страничке:

```
<form name="form1" method="get" action="process.php">
<p>Пожалуйста, укажите Ваше имя и адрес электронной
почты:</p>
<p><input type="text" name="yourname"></p>
<p><input type="text" name="email"></p>
<p><input type="button" name="Submit" value="Отправить">
</p>
</form>
```

Когда пользователь щелкает по кнопке *Отправить*, то браузер забирает значения из полей формы и отправляет их по адресу:

```
http://www.my_site.ru/process.php?yourname=fred&email=
fred@mail.ru
```

Заметили как значения, введенные в форму, стали частью адреса? В этом и состоит суть метода **GET**.

## POST

Метод **POST,** передает имена переменных и их значения в теле запроса URL, а не в заголовке. Преимущества такого способа передачи данных состоят в том, что снимаются ограничения на объем передаваемых данных, поскольку они располагаются в теле запроса HTTP, а не в заголовке. Дополнительно, если вами используется SSL соединение, то данные проходят по сети в зашифрованном виде. Ниже приводится пример формы, использующей метод POST для передачи данных:

```
<form name="form1" method="post" action="process.php">
<p>Пожалуйста, укажите Ваше имя и адрес электронной
почты:</p>
<p><input type="text" name="yourname"></p>
<p><input type="text" name="email"></p>
<p><input type="button" name="Submit" value="Отправить">
</p>
</form>
```

Когда пользователь щелкает по кнопке *Отправить*, то браузер забирает значения из полей формы и отправляет их по адресу:

```
http://www.my_site.ru/process.php
```

Заметили, что теперь данные уже не являются частью адреса? В этом и состоит суть метода **POST**.

Кроме того вы должны знать, что кроме GET и POST существуют и другие методы передачи данных веб-страничкам:

## Суперглобальные массивы в PHP (Superglobals)

Представьте, к примеру, что вы работаете с переменной `$yourname`, а теперь скажите — можете ли вы быть абсолютно уверены в том, что в ходе исполнения скрипта эта переменная не была переопределена кем либо, пытающимся взломать сценарий? Например, вообразите себе, что некто сумел закачать на ваш веб-сервер скрипт со следующим содержанием:

```
<?php
setcookie("test","../../../../../../etc/passwd");
echo "cookie inserted";
?>
```

И взломщик получает ваши явки и пароли…

Было бы великолепно иметь возможность изолировать переменные, исходя из того, каким методом они могут быть изменены.

На рисунке представлены методы передачи данных веб-страницам:


Суперглобальные массивы позволят точно определить **каким способом переменная может получить свое значение**. Если пользователь заполнит форму (упоминавшуюся выше), то сервер запомнит введенные данные не в глобальных переменных `$yourname` или `$email`, а в ассоциативном массиве `$_POST`:

```
$_POST['yourname']
$_POST['email']
```

Следует сказать, что использование суперглобальных массивов стало необходимым, поскольку в файле настроек языка PHP **php.ini** директива `register_globals=Off`, т.е. отключена.

Если же у вас в Денвере установлен PHP, в котором `register_globals=On`, то советую сперва изменить эту настойку и только потом делать лабораторные работы.

Узнать о настройках PHP можно с помощью такого скрипта:

```
<?php  phpinfo();  ?>
```

**Основные суперглобальные массивы:**

1. `$_GET['variable']` — Переменные, переданные сценарию через HTTP GET. Аналогичен массиву `$HTTP_GET_VARS`
2. `$_POST['variable']` — Переменные, переданные сценарию через HTTP POST. Аналогичен массиву `$HTTP_POST_VARS`

**Другие суперглобальные массивы:**

1. `$_COOKIE['variable']` — Переменные, переданные сценарию через HTTP cookies. Аналогичен массиву `$HTTP_COOKIE_VARS`
2. `$_REQUEST['variable']` — Переменные, переданные сценарию в результате ответа пользователя на запрос (GET, POST, COOKIE) и которым, поэтому, не следует доверять.
3. `$_GLOBALS['variable']` — Содержит ссылки на все переменные, которые в настоящий момент доступны в глобальной области видимости сценария. Ключами этого массива являются имена глобальных переменных.
4. `$_SERVER['variable']` — Переменные, установленные веб-сервером или чем-то другим, напрямую связанным со средой исполнения сценария. Аналогичен массиву `$HTTP_SERVER_VARS`
5. `$_FILES['variable']` — Переменные, переданные сценарию путем передачи файла через HTTP. Аналогичен массиву `$HTTP_POST_FILES`
6. `$_ENV['variable']` — Переменные, переданные сценарию через окружение. Аналогичен массиву `$HTTP_ENV_VARS`
7. `$_SESSION['variable']` — Переменные, которые определены в текущей сессии сценария. Аналогичен массиву `$HTTP_SESSION_VARS`

Таким образом, вместо переменной `$name`, в которую записано, к примеру, имя "John", вы получите или `$_GET['name']="John"` или `$_POST['name']="John"`, в зависимости от того, каким способом была передана эта переменная. Преимущество состоит в том, что:

1. Переменная `$name` никогда не сможет быть изменена внешним пользователем; если сценарий записал туда некое значение, то оно там и останется!
2. Массивы `$_GET` и `$_POST` помогут вам отделить данные, пришедшие от формы методом POST (в теле запроса) от данных, добавленных кем-либо, пытающимся имитировать передачу данных методом GET, в конец адресной строки.
3. Суперглобальные массивы позволят вам не только изолировать значения переменных, но и разнести их по категориям, в зависимости от способа доставки на сервер. А кое-кому значительно усложнят взлом вашего сервера.

Теперь, с учетом вышесказанного о безопасности работы с данными, приходящими в скрипт из формы, рассмотрим листинг 10.2, обрабатывающий данные формы из листинга 10.1.

**Листинг 10.2. Чтение данных формы из листинга 10.1**

```
<html> <head>
<title> Листинг 10-2. Чтение данных формы
        из листинга 10-1 </title> </head> <body>
<?php
// Из файла ls10-1.php предается массив $_GET['bg']
print "<body bgcolor=silver text=black link=white
        alink=maroon vlink=".$_GET['bg']."\n";
print "<p><table border=1 width=100 align=center>\n";
print "<tr> <td bgcolor=".$_GET['bg']." <br>
 </td></tr></table>\n";
print "<p align=center><a href='ls10-1.htm'>назад</a>";
?>
</body> </html>
```

Эта программа запускается не щелчком на ссылке и не указанием имени файла в строке адреса в окне браузера. Текст программы из листинга 10.2 нужно поместить в файл с именем `ls10-2.php`, и этот файл будет запускаться сервером тогда, когда пользователь нажмет кнопку «Выполнить» в форме из листинга 10.1. В программе мы обращаемся к элементу **`'bg'`** ассоциативного массива **`$_GET[]`**— в нем записано название цвета, которое выбрал пользователь. В результате на экране будет изображен прямоугольник и гиперссылка «назад» соответствующего цвета. Обратите внимание на знак конкатенации («точку») при объединении в операторе **`print`** строки (в кавычках) и элемента массива **`$_GET['bg']`**.

## Обработка элементов с многозначным выбором

В предыдущем примере мы рассмотрели обработку данных, при которой каждому элементу соответствует только одно значение. Но остается неясным, как обрабатывать информацию от элементов, которые позволяют выбирать несколько значений из списка. Создав несколько флажков с одним именем, вы позволяете пользователю ввести несколько значений под этим одним именем. Если в теге `input` мы дадим этому элементу простое имя, то по такому имени программа сможет обратиться только к одному значению, связанному с

этим элементом. Для того чтобы программа могла иметь доступ ко всем значениям, выбранным пользователем, нужно к имени данного элемента приписать пару пустых квадратных скобок (т.е. создать массив). Это и сделано в листинге 10.3.

### Листинг 10.3. HTML-форма с выбором из списка

```
<html> <head>
<title> Листинг 10-3. HTML-форма с выбором из списка
</title> </head>
<body>
<!--
Здесь в скрипт ls10-4.php передаются:
1) переменная $user
2) массив hobby[] со значениями, которые были выбраны
   в форме
-->
<form action="ls10-4.php" method="POST">
<p>Введите ваше имя
<p><textarea rows="1" cols="30" name="user">
</textarea>
<p>Что вы любите делать в свободное время <br>
   (можно выбрать несколько вариантов)
<p><input type="checkbox" name="hobby[]"
    value="слушать музыку">слушать музыку
<p><input type="checkbox" name="hobby[]"
    value="читать книгу">читать книгу
<p><input type="checkbox" name="hobby[]"
    value="смотреть телевизор">смотреть телевизор
<p><input type="checkbox" name="hobby[]"
    value="гулять на улице">гулять на улице
<p><input type="submit" value="Выбор сделан">
</form>
</body> </html>
```

Теперь в скрипте `ls10-4.php`, предназначенном для обработки этой формы, значения, выбранные пользователем в элементе `"hobby[]"`, будут доступны как элемент **`'hobby'`** массива **`$_POST['hobby']`**, причем этот элемент, в

свою очередь, является массивом! Кроме того, в скрипт ls10-4.php передается значение элемента **'user'** массива **$_POST['user']** — в котором записано то, что пользователь указал в текстовом поле "user". Пример вы видите в листинге 10.4.

**Листинг 10.4. Обработка данных формы из листинга 10.3**

```
<html> <head>
<title> Листинг 10-4. Обработка данных формы
        из листинга 10-3 </title> </head> <body>
<?php
 print "<p>".$_POST['user']."", оказывается, вы
предпочитаете";
print "<ul>\n";
foreach ($_POST['hobby'] as $value)
        {print "<li>$value\n";
}
print "</ul>\n";
?>
 </body> </html>
```

Несколько флажков с одним именем — не единственный способ, который позволяет ввести несколько значений. В листинге 10.3 можно заменить последовательность флажков с одним именем на тег SELECT с атрибутом multiple, и работать это будет точно так же.

```
<select name= "hobby[]" multiple>
<option value="слушать музыку">слушать музыку
<option value="читать книгу">читать книгу
<option value="смотреть телевизор">смотреть телевизор
<option value="гулять на улице">гулять на улице
</select>
```

## Доступ ко всем полям формы через суперглобальный массив

Рассмотренная выше техника работает прекрасно, но только в том случае, если вы заранее знаете имена всех полей формы. Однако вам может понадобиться написать программу, которая умеет настраиваться на изменения, сделанные в форме, или даже обрабатывать несколько разных форм, при этом не заботясь о фактических именах полей. Эту проблему также можно решить с помощью

суперглобальных массивов **$_GET** и **$_POST**. По своей структуре это ассоциативные массивы, содержащие пары имя/значение для каждого элемента переданной формы. В листинге 10.5 показано, как с помощью этого способа можно вывести список всех полей и переданных значений при использовании метода передачи `POST`.

### Листинг 10.5. Чтение данных произвольной формы с помощью суперглобального массива

```
<?php
/* Заменить цикл foreach в листинге ls10-4.php
   Листинг 10-5. Чтение данных произвольной формы
            с помощью суперглобального массива
*/
foreach ($_POST as $key=>$value) {
         print "$key = $value<br>\n";
}
?>
```

Приведенная выше программа выводит имена и значения всех параметров, переданных ей. Конечно, не всегда разумно обрабатывать данные таким образом. Если бы мы передали этой программе для обработки форму из листинга 10.3, то получили бы следующее:

```
user = guest
hobby = Array
```

Дело в том, что данные `hobby` существуют как элементы массива `$_POST['hobby']`, а мы попытались обратиться к ним как к простой переменной. В листинге 10.6 эта ошибка исправлена и, перед тем как вывести те или иные данные, их тип проверяется.

### Листинг 10.6. Чтение данных произвольной формы с проверкой типов

```
<?php
/* Заменить цикл foreach в листинге ls10-4.php
   Листинг 10-6. Чтение данных произвольной формы
            с проверкой типов
*/
foreach ($_POST as $key=>$value)
{  #1
if (gettype($value) == "array")
  {  #2
print "$key = <br>\n";
     foreach ($value as $v ) {print "$v<br>";}
  }  #2
else   {
print "$key = $value<br>\n";
        }
}  #1
?>
```

Как и в предыдущих примерах, в цикле `foreach` мы просматриваем массив `$_POST[]` и с помощью функции `gettype()` проверяем, является ли очередной его элемент массивом. Если да, то создаем вложенный цикл, в котором выводим значения этого массива.

## Расположение HTML-текста и PHP-программы на одной странице

При некоторых обстоятельствах вам может понадобиться расположить PHP-программу, обрабатывающую данные, на той же странице, что и форму, передающую такие данные. Это может оказаться удобным, например, в том случае, когда вам нужно вывести форму несколько раз. В листинге 10.7 приведен пример такой формы: на экране браузера изображается небольшой прямоугольник — таблица, содержащая всего одну ячейку и предлагается задать цвет для этого прямоугольника. После выбора нужного цвета и нажатия кнопки "`закрасить`", можно видеть, что прямоугольник тут же меняет свой цвет.

**Листинг 10.7. HTML-форма, вызывающая саму себя**

```
<html> <head>
<title> Листинг 10-7. HTML-форма, вызывающая саму себя
</title> </head> <body>
<?php
if ($_GET['bg']==null) {$bgc = "silver";}
else {$bgc = $_GET['bg'];}
print "<p>Выберите цвет фона прямоугольника";
print "<p><table border=1 width=100>";
print "<tr> <td bgcolor=$bgc> <br> 
</td></tr></table>";
?>
<form action="<?=$_SERVER['PHP_SELF'] ?>" method="get">
<p><input type="radio" name="bg" value="red">красный
<p><input type="radio" name="bg" value="green">зеленый
<p><input type="radio" name="bg" value="blue">синий
<p><input type="radio" name="bg" value="silver">исходный
<p><input type="submit" value="закрасить">
</form>
</body> </html>
```

Как бы мы ни назвали файл, содержащий эту форму, все равно форма будет обращаться к этому файлу, так как в ней в атрибуте `ACTION` есть элемент **`'PHP_SELF'`** суперглобального массива **`$_SERVER[]`**. Если мы первый раз вызываем файл с формой, то элемент **`$_GET['bg']`** еще не определен, т.е. имеет значение **`null`**.

## Значения по умолчанию в HTML-форме

В тегах, являющихся элементами HTML-формы возможно задать значение по умолчанию (в атрибуте `VALUE`) не только явно:

```
<input type="text" name="mycolor" value="red">
```

но и с помощью переменной:

```
<input type="text" name="mycolor" value="$color">
```

В частности, для тега `textarea` это делается так:

```
<textarea rows="1" cols="30" name="mycolor">$color
</textarea>
```

В этом случае на экране браузера будет отображаться текстовое поле, внутри которого находится текстовая строка из переменной `$color`. Если же значение этой переменной передано из другого скрипта методом `POST`, придется писать так:

```
print "<input type='text' name='mycolor'
value=".$_POST['color']."">";
```

## Перенаправление пользователя

Предположим, вы хотите дать возможность посетителю выбрать афишу мероприятий на один день недели и автоматически перенаправить его туда. В листинге 10.8 для этого используется функция `header()`. Когда программа на сервере общается с клиентом, она должна сначала послать некоторые заголовки, содержащие информацию о том документе, который за ними следует. Обычно PHP делает это автоматически, но вы можете послать свой собственный заголовок с помощью функции `header()`. Но прежде чем послать заголовок, вы должны убедиться в том, что никаких данных от вашей программы на браузер пока не поступало. К тому моменту, когда данные посылаются на браузер, заголовки уже отправлены и вам уже поздно посылать свой собственный заголовок. Даже простой перевод строки или единственный пробел приводят к тому, что программа посылает заголовок типа содержания. Если вы хотите послать свой собственный заголовок с помощью функции `header()`, то должны проверить программу и убедиться, что ничего пока не отправлено.

Послав браузеру заголовок `"Location"`, вы перенаправите пользователя на новую страницу:

```
header ("Location: http://www.my_new_site.ru");
```

### Листинг 10.8. Посылка заголовка с помощью функции `header()`

```
<?php
if ($_POST['day'] != "") {
    header("Location: ".$_POST['day']);
    exit;
}
else { // начало блока else
?>
<html>
```

```
<head>
<title> Листинг 10-8. Посылка заголовка с помощью
        функции header()
</title>
</head>
<body>
<form action="<?=$_SERVER['PHP_SELF'] ?>" method="post">
<select name="day">
<option value = "">Афиша:
<option value='d1.htm'>Понедельник
<option value='d2.htm'>Вторник
<option value='d3.htm'>Среда
<option value='d4.htm'>Четверг
<option value='d5.htm'>Пятница
<option value='d6.htm'>Суббота
<option value='d7.htm'>Воскресенье
</select>
<input type="submit" value="Перейти">
</form>
<?php
     } // конец блока else
?>
</body> </html>
```

---

Сначала мы проверяем, было ли присвоено значение элементу `'day'` массива `$_POST[]` (являющегося именем элемента `select` в форме). Если проверка дает положительный результат, вызывается функция `header()` с аргументом, в котором значение `$_POST['day']` присоединяется к строке `"Location: "`. При передаче этой команды функция `header()` перенаправляет браузер на соответствующий файл (`d1.htm`, `d2.htm` … `d7.htm`). Если значение `$_POST['day']`, не задано, форма выводится в браузере.