# 9 Формы

## 9.0. Введение

Одной из самых сильных сторон PHP является органичная интеграция переменных форм в программы. Благодаря ей веб-программирование становится простым и элегантным, а цикл перехода от веб-форм к коду PHP и выводу HTML существенно ускоряется.

Впрочем, вместе с удобствами приходит ответственность: вы должны проследить за тем, чтобы предоставленная пользователем информация, которая так легко перетекает в вашу программу, содержала правильные данные. Внешнему вводу нельзя доверять ни при каких условиях, поэтому очень важно проверять все вводимые данные. Рецепты 9.2–9.9 показывают, как проверять типичные виды информации, а также содержат общие рекомендации по проверке произвольных данных форм. В Рецепте 9.10 рассматривается экранирование сущностей HTML для безопасного отображения данных, введенных пользователем. Рецепт 9.11 объясняет, как обрабатывать файлы, отправленные пользователем.

HTTP относится к категории протоколов *без состояния* — в нем нет встроенного механизма, позволяющего сохранить информацию с одной страницы, чтобы обратиться к ней из других страниц. В Рецептах 9.12, 9.13 и 9.14 представлены обходные решения этой фундаментальной проблемы для определения того, какой пользователь выдал тот или иной запрос к веб-серверу.

При обработке страницы PHP проверяет URL-адрес и переменные формы, отправленные файлы, действующие cookie, переменные веб-сервера и окружения. Полученные данные напрямую доступны в следующих массивах: `$_GET`, `$_POST`, `$_FILES`, `$_COOKIE`, `$_SERVER` и `$_ENV`. В них хранятся соответственно все переменные, заданные в строке запроса, в теле запроса, отправкой файлов, cookie, веб-сервером и окружением, в котором работает веб-сервер. Также существует `$_REQUEST` — один большой массив, в котором содержатся значения из шести массивов.

При размещении элементов в $_REQUEST, если два массива содержат одноименные ключи, PHP разрешает конфликт в соответствии с конфигурационной директивой variables_order. По умолчанию variables_order использует порядок EGPCS (или GPCS, если вы используете конфигурационный файл php.ini-recommended). Итак, PHP сначала добавляет в $_REQUEST переменные окружения, а затем добавляет в массив переменные строки запроса, отправленные значения, данные cookie и переменные веб-сервера в указанном порядке. Например, так как C в порядке по умолчанию следует после P, cookie с именем username заменяет отправленную переменную с именем username. Следует заметить, что со значением GPCS из файла php.ini-recommended массив $_ENV не заполняется переменными окружения.

Хотя массив $_REQUEST может быть удобен, обычно удобнее работать со специализированными массивами. В этом случае вы точно знаете, что получаете, и вам не нужно беспокоиться о том, что изменение variables_order повлияет на поведение программы.

Все эти массивы являются *автоглобальными*. Этот термин обозначает глобальность внутри функции или класса — они всегда находятся в области видимости.

В версиях PHP до 5.4.0 поддерживалась конфигурационная директива с именем register_globals. Если она была установлена, то все эти переменные также становились доступны в виде переменных глобального пространства имен. Итак, значение $_GET['password'] также доступно под именем $password. При всем удобстве такой подход создает серьезные проблемы с безопасностью, потому что злоумышленник может легко задать переменные снаружи и заменить доверенные внутренние переменные. Если вы используете старую версию PHP, проследите за тем, чтобы эта директива была отключена.

В листинге 9.1 приведена простая форма, на которой пользователю предлагается ввести имя. При отправке формы информация передается hello.php.

**Листинг 9.1.** Простая форма HTML

```
<form action="hello.php" method="post">
<p>What is your first name?</p>
<input type="text" name="first_name" />
<input type="submit" value="Say Hello" />
</form>
```

Текстовому полю на форме присвоено имя first_name, а форма использует метод post. Это означает, что при отправке формы в $_POST['first_name'] будет храниться строка, введенная пользователем. (Конечно, элемент может быть и пустым, если пользователь ничего не ввел.)

В листинге 9.2 приведено содержимое программы hello.php, которая выводит информацию из формы.

**Листинг 9.2.** Простейшая обработка формы на PHP

```
echo 'Hello, ' . $_POST['first_name'] . '!';
```

Если ввести на форме из листинга 9.1 имя Twinkle, то листинг 9.2 выведет сообщение:

```
Hello, Twinkle!
```

Программа из листинга 9.2 предельно проста, и в ней пропущены два важных шага, которые должны присутствовать во всех приложениях обработки форм PHP: проверка данных (которая гарантирует, что данные, введенные на форме, являются допустимыми для вашей программы) и экранирование вывода (чтобы злоумышленники не могли использовать ваш сайт для атак). Проверка данных рассматривается в Рецептах 9.2–9.9, а экранирование вывода — в Рецепте 9.10.

# 9.1. Обработка ввода

## Задача

Требуется использовать одну страницу HTML для отображения формы и последующей обработки данных, введенных на этой форме. Иначе говоря, вы хотите избежать создания лишних страниц для обработки последовательных этапов операции.

## Решение

Используйте переменную `$_SERVER['REQUEST_METHOD']` для определения того, был ли запрос отправлен методом `get` или `post`. Если использовался метод `get`, выведите форму. Если использовался метод `post`, обработайте форму. В листинге 9.3 форма из листинга 9.1 объединяется с кодом из листинга 9.2 в одну программу, которая выбирает выполняемую операцию в зависимости от значения `$_SERVER['REQUEST_METHOD']`.

**Листинг 9.3.** Выбор операции в зависимости от метода запроса

```
<?php if ($_SERVER['REQUEST_METHOD'] == 'GET') { ?>
<form action="<?php echo htmlentities($_SERVER['SCRIPT_NAME']) ?>" method="post">
What is your first name?
<input type="text" name="first_name" />
<input type="submit" value="Say Hello" />
</form>
<?php } else {
    echo 'Hello, ' . $_POST['first_name'] . '!';
}
```

## Комментарий

Работа с формами упрощается в том случае, если все части находятся в одном файле (или используются одним файлом), а отображаемые секции определяются контекстом. Метод `get` (который используется вашим браузером при вводе

URL-адреса или щелчке на ссылке), по сути, означает: «Эй, сервер, передай мне то, что у тебя есть». Метод `post` (используемый браузером при отправке формы, у которой атрибут `method` имеет значение `post`) означает: «Эй, сервер, вот тебе данные, которые должны что-то изменить». Таким образом, типичным ответом на запрос `get` является форма HTML, а типичной реакцией на запрос `post` — результаты обработки этой формы. В листинге 9.3 «обработка» ограничивается простым выводом приветствия. В более типичных приложениях выполняются более сложные действия, например сохранение информации в базе данных или отправка сообщений электронной почты.

Хотя спецификация XHTML требует, чтобы атрибут `method` элемента `<form/>` записывался в нижнем регистре (`get` или `post`), по спецификации HTTP браузер должен использовать символы верхнего регистра (`GET` или `POST`) при отправке серверу метода запроса. В `$_SERVER['REQUEST_METHOD']` хранится значение, отправленное браузером, поэтому на практике оно всегда содержит символы верхнего регистра.

Также для упрощения сопровождения страниц не стоит жестко кодировать путь к странице в атрибуте `action` формы; это не позволит вам переименовать или переместить страницу без ее редактирования. Вместо этого в качестве значения `action` используется переменная `$_SERVER['SCRIPT_NAME']`. PHP для каждого запроса присваивает этой переменной имя файла текущего сценария (относительно корневого каталога документов).

Если вы используете инфраструктуры для разработки веб-приложений, у них имеются собственные правила объединения отображения форм с обработкой результатов. Хотя в этой книге мы не будем ориентироваться ни на какую конкретную инфраструктуру, четкое разделение между кодом представления (отображения информации для пользователя) и «бизнес-логикой» (обработкой данных, предоставленных пользователем) упростит сопровождение кода и его понимание. Для форм сколько-нибудь сложнее листинга 9.3 выделение логики отображения формы в шаблон принесет заметную пользу. Существует множество отличных языков для работы с шаблонами, но для простоты в этой книге в качестве шаблонного языка будет использоваться PHP.

При такой переработке листинг 9.3 преобразуется в три файла: один отображает форму по запросу `get`, другой обрабатывает результаты по запросу `post`, а третий решает, какую операцию следует выполнить.

Код отображения формы выглядит так:

```
<form action="<?= htmlentities($_SERVER['SCRIPT_NAME']) ?>" method="post">
What is your first name?
<input type="text" name="first_name" />
<input type="submit" value="Say Hello" />
</form>
```

Логика обработки формы:

```
Hello, <?= $_POST['first_name'] ?> !
```

И наконец, логика принятия решения:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {
    include __DIR__ . '/getpost-get.php';
}
else {
    include __DIR__ . '/getpost-post.php';
}
```

Логика принятия решений предполагает, что код отображения формы хранится в файле getpost-get.php, а код обработки формы — в файле getpost-post.php и что все три файла находятся в одном каталоге. Константа `__DIR__` сообщает программе, что искать следует в том же каталоге, в котором находится исполняемый код включаемых файлов.

Стратегия разбиения страниц на несколько файлов будет использоваться в других рецептах этой главы.

### См. также

Рецепт 9.12 — обработка многостраничных форм.

## 9.2. Проверка ввода на форме: обязательные поля

### Задача

Требуется убедиться в том, что для элемента формы было задано значение, например, что текстовое поле не осталось пустым.

### Решение

Воспользуйтесь функцией `filter_has_var()` для проверки присутствия элемента в соответствующем входном массиве, как показано в листинге 9.4.

**Листинг 9.4.** Проверка обязательного поля

```
if (! filter_has_var(INPUT_POST, 'flavor')) {
   print 'You must enter your favorite ice cream flavor.';
}
```

### Комментарий

Функция `filter_has_var()` проверяет входные данные, полученные PHP, перед внесением каких-либо модификаций в вашем коде. Последовательное использование различных функций-фильтров, рассматриваемых позднее в этой главе, обеспечивает необходимую проверку и защитную обработку данных, введенных

пользователем. Первый аргумент `filter_has_var()` указывает, где следует искать; значение `INPUT_POST` проверяет данные `POST` в теле запроса. Другие возможные значения — `INPUT_GET` (переменные из строки запроса), `INPUT_COOKIE` (cookie), `INPUT_SERVER` (серверная информация, которая сохраняется в `$_SERVER`) и `INPUT_ENV` (переменные окружения).

Разные типы элементов форм, которые не были заполнены пользователем, поразному ведут себя в данных `GET` и `POST`. Пустые текстовые поля, текстовые области и поля отправки файлов приводят к появлению элементов, значение которых представляет собой строку нулевой длины. Для флажков и переключателей, которые не были установлены, никакие элементы в данных `GET` или `POST` не создаются. Браузеры обычно заставляют пользователя выбрать один из вариантов в раскрывающихся меню, поддерживающих одиночный выбор, но раскрывающиеся меню с множественным выбором, в которых нет выбранных вариантов, работают аналогично флажкам — они не создают никаких элементов в данных `GET` или `POST`.

Что еще хуже, запросы могут поступать не только от браузеров. Ваши программы PHP могут получать запросы от других программ, любознательный хакер может построить запрос вручную, а злоумышленник может создавать запросы в попытке найти уязвимость в вашей системе. Чтобы ваш код был как можно более надежным, всегда проверяйте, что некоторый элемент существует в наборе входных данных, прежде чем применять к нему другие стратегии проверки. Кроме того, если стратегия проверки подразумевает, что элемент представляет собой массив значений (как в листинге 9.14), убедитесь в этом при помощи флага фильтрации `FILTER_REQUIRE_ARRAY`.

В листинге 9.5 функции `filter_has_var()`, `filter_input()` и `strlen()` используются для максимально жесткой проверки данных формы.

**Листинг 9.5.** Проверка данных формы

```
// Убедиться в том, что значение $_POST['flavor'] существует,
// прежде чем проверять его длину
if (! (filter_has_var(INPUT_POST, 'flavor') &&
       (strlen(filter_input(INPUT_POST, 'flavor')) > 0))) {
    print 'You must enter your favorite ice cream flavor.';
}

// Значение $_POST['color'] не является обязательным, но если оно
// задано, то после защитной обработки оно должно содержать
// более 5 символов
if (filter_has_var(INPUT_POST, 'color') &&
    (strlen(filter_input(INPUT_POST, 'color', FILTER_SANITIZE_STRING)) <= 5)) {
    print 'Color must be more than 5 characters.';
}
// Убедиться в том, что $_POST['choices'] существует
// и представляет собой массив
if (! (filter_has_var(INPUT_POST, 'choices') &&
       filter_input(INPUT_POST, 'choices', FILTER_DEFAULT,
                    FILTER_REQUIRE_ARRAY))) {
    print 'You must select some choices.';
}
```

Вызов `filter_input()` с двумя аргументами применяет фильтр по умолчанию, который не изменяет входные данные. В листинге 9.5 отправленное значение `flavor` никак не преобразуется. Фильтр `FILTER_SANITIZE_STRING`, примененный к отправленному значению `color`, удаляет теги HTML, удаляет двоичные символы, не входящие в набор ASCII, и кодирует амперсанды (`&`). Фильтр `FILTER_DEFAULT` применяется к `choices` для явного задания фильтра по умолчанию. Это существенно в последней части листинга 9.5, потому что флаг фильтра `FILTER_REQUIRE_ARRAY` должен передаваться в четвертом аргументе `filter_input()`.

Возможно, в минуту слабости вам захочется использовать `empty()` вместо `strlen()` для проверки того, что в текстовом поле было введено значение. Не поддавайтесь искушению; вы создадите себе проблемы, потому что строка из одного символа `0` интерпретируется как `false` по правилам логических вычислений PHP. Это может привести к ошибкам в проверке форм, если, например, кто-нибудь введет 0 в текстовом поле children, в результате чего элемент `$_POST['children']` будет содержать 0. Тогда вызов `empty($_POST['children'])` вернет `true` — что с точки зрения проверки формы свидетельствует об ошибке.

### См. также

Документация по функциям `filter_has_var()` и `filter_input()`, список фильтров защитной обработки, список флагов фильтров; Рецепт 9.5 — информация о проверке раскрывающихся меню, Рецепт 9.6 — информация о проверке переключателей, Рецепт 9.7 — информация о проверке флажков.

## 9.3. Проверка ввода на форме: числа

### Задача

Требуется убедиться в том, что в поле на форме было введено число. Например, вы хотите проверить, что в поле «Возраст пользователя» введено конкретное значение (13, 56 и т. д.), а не текст «В расцвете лет» или что-нибудь в этом роде.

### Решение

Если вас интересуют целые числа, воспользуйтесь фильтром `FILTER_VALIDATE_INT`, как показано в листинге 9.6.

**Листинг 9.6.** Проверка чисел с фильтром FILTER_VALIDATE_INT

```
$age = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
if ($age === false) {
   print "Submitted age is invalid.";
}
```

Если вас интересуют дробные числа, используйте фильтр FILTER_VALIDATE_FLOAT, как показано в листинге 9.7.

**Листинг 9.7.** Проверка чисел с фильтром FILTER_VALIDATE_FLOAT

```
$price = filter_input(INPUT_POST, 'price', FILTER_VALIDATE_FLOAT);
if ($price === false) {
   print "Submitted price is invalid.";
}
```

## Комментарий

С фильтрами FILTER_VALIDATE_INT и FILTER_VALIDATE_FLOAT функция filter_input() возвращает число указанного типа (целое или с плавающей точкой), если входная строка представляет число, соответствующее фильтру, или false в противном случае.

Существует несколько флагов, управляющих поведением числовых фильтров. Флаг FILTER_FLAG_ALLOW_OCTAL приказывает FILTER_VALIDATE_INT поддерживать восьмеричную запись. Другими словами, для отправленной строки 017 возвращается целое число 15. С похожим флагом FILTER_FLAG_ALLOW_HEX отправленная строка 0x2f возвращается в виде целого числа 47.

Флаг FILTER_FLAG_ALLOW_THOUSAND изменяет поведение фильтра FILTER_VALIDATE_FLOAT, разрешая использование запятых для разделения групп разрядов. Без него строка 5,252 будет сочтена недействительной, а с ним она правильно проходит проверку как число с плавающей точкой 5252.

В некоторых видах проверки могут использоваться регулярные выражения. В листинге 9.8 приведены регулярные выражения для проверки целого и дробного числа.

**Листинг 9.8.** Проверка чисел по регулярному выражению

```
// Шаблон совпадает с необязательным знаком -,
// за которым следует минимум одна цифра
if (! preg_match('/^-?\d+$/',$_POST['rating'])) {
    print 'Your rating must be an integer.';
}

// Шаблон совпадает с необязательным знаком -, за которым следует
// необязательная последовательность цифр (целая часть),
// затем необязательная точка и как минимум одна цифра.
if (! preg_match('/^-?\d*\.?\d+$/',$_POST['temperature'])) {
   print 'Your temperature must be a number.';
}
```

Многие фанатичные сторонники оптимизации считают, что регулярных выражений следует избегать, потому что они работают относительно медленно. Однако в данном случае простые регулярные выражения практически не уступают функциям фильтров по эффективности. Если вы увереннее чувствуете себя

с регулярными выражениями или используете их в других контекстах, этот вариант может быть удобен. Регулярное выражение также позволяет рассматривать действительные числа (например, 782364.238723123), которые не могут храниться в формате с плавающей точкой PHP без потери точности. Например, такая возможность может пригодиться для хранения значений широты и долготы, которые вы планируете хранить в строковом виде.

### См. также

Рецепт 9.2 — информация о проверке обязательных полей; список фильтров проверки; список флагов фильтров.

## 9.4. Проверка ввода на форме: адреса электронной почты

### Задача

Требуется проверить адрес электронной почты, введенный пользователем.

### Решение

Используйте фильтр `FILTER_VALIDATE_EMAIL`, как показано в листинге 9.9. Этот фильтр проверяет действительность адреса электронной почты по правилам RFC 5321 (в основном).

**Листинг 9.9.** Проверка адреса электронной почты

```
$email = filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL);
if ($email === false) {
    print "Submitted email address is invalid.";
}
```

### Комментарий

RFC 5321 объединяет несколько документов RFC, относящихся к работе с электронной почтой, и определяет стандарты действительных адресов электронной почты. Фильтр `FILTER_VALIDATE_EMAIL` использует регулярное выражение, базирующееся на этих правилах, хотя он и не поддерживает комментарии или свертку пробелов.

Фильтр проверяет только синтаксическую правильность конкретного адреса. Он поможет предотвратить ввод адреса вида *bingolover2261@example* вместо *bingolover2261@example.com*, но ничего не скажет о том, что произойдет при отправке сообщения по этому адресу. Более того, фильтр не скажет, действительно