

Ссылки в PHP – как они работают и когда их использовать?

Доброго времени суток, уважаемые читатели блога SoftMaker.kz. На этом сайте уже есть статья о том [что такое PHP](#). А сегодня мы поговорим, о ссылках в языке программирование PHP, так как ссылки являются весьма полезной частью языка программирования PHP. Это утверждение касается так же и большинства других языков программирования. Тем не менее, тема ссылок может быть не вполне понятной при первом знакомстве с ней.



Эта статья поможет устранить всё, что может быть непонятно при первом знакомстве с темой **ссылок в PHP**. Вы узнаете, что такое ссылки в PHP и как они работают. Также в этой статье вы узнаете, как создавать и удалять ссылки, а также как передавать и возвращать переменную по ссылке в функцию и из неё. Вы также изучите некоторые другие виды ссылок, и узнаете о ситуациях, когда PHP создаёт ссылки автоматически.

- [Что же такое ссылки в PHP?](#)
- [Присвоение ссылки в PHP](#)
- [Удаление ссылки в PHP](#)

- [Передача переменных в функцию по ссылке](#)
- [Возвращение по ссылке из функций](#)
- [Изменение значений в предложении foreach с помощью ссылок в PHP](#)
- [Когда ссылки используются автоматически](#)
- [Краткий обзор статьи о ссылках в PHP](#)

Что же такое ссылки в PHP?

Ссылка – это способ обратиться к переменной с помощью другого имени. PHP-ссылки не похожи на указатели языка программирования C и не являются псевдонимами таблицы символов. Во многих отношениях они похожи на ярлык в Windows, файл псевдоним в [Mac OS X](#) и символические ссылки в [Linux](#).

Присвоение ссылки в PHP

```
<?php
```

```
$myVar = "Привет!";  
$anotherVar = $myVar;  
$anotherVar = "Увидимся позже";  
echo $myVar; // Выведет "Привет!"  
echo $anotherVar; // Выведет "Увидимся позже"
```

```
?>
```

Здесь мы создали переменную *\$myVar* со значением «Привет!». Затем мы присвоили значение другой переменной *\$anotherVar*. Это копия значения первой переменной во вторую. Затем мы изменим значение, сохраненное в *\$anotherVar* на «Увидимся позже».

Поскольку две переменные являются независимыми, *\$myVar* по-прежнему сохраняет свое первоначальное значение («Привет!»), которое будет выведено на странице. Пока всё идёт хорошо. А

теперь давайте изменим пример, чтобы присвоить переменной *\$myVar* значение *\$anotherVar*, используя *ссылку*, а не значение. Чтобы сделать это, мы просто напишем знак амперсанда («&») после знака равенства :

```
<?php  
  
$myVar = "Привет!";  
$AnotherVar =& $myVar;  
$AnotherVar= "Увидимся позже";  
echo $myVar; // Выведет "Увидимся позже"  
echo $anotherVar; // Выведет "Увидимся позже"  
  
?>
```

Теперь вы можете видеть, что *\$myVar* также изменен на «Увидимся позже»! Почему это произошло? Вместо того, чтобы присвоить значение переменной *\$myVar* переменной *\$anotherVar* – которые просто создают две независимых копии одного и того же значения – мы сделали переменную *\$anotherVar* *ссылкой* на значение *\$myVar*. Другими словами, *\$myVar* и *\$anotherVar* оба указывают на одно и то же значение. Таким образом, когда мы присвоили новое значение переменной, *\$anotherVar* значение переменной *\$myVar* также изменилось.

Обратите внимание на то, что мы могли бы изменить значение переменной *\$myVar* на «Увидимся позже» вместо изменения переменной *\$anotherVar* и результат был бы точно такой же. Две переменных, по сути, являются идентичными.

Удаление ссылки в PHP

Вы можете удалить ссылку с помощью функции PHP `unset()` также как вы удаляете обычную переменную. Когда вы удаляете ссылку, вы просто удаляете саму ссылку, а не значение ссылки:

```
<?php  
  
$myVar= "Привет!";
```

```
$AnotherVar =& $myVar;  
$AnotherVar = "Увидимся позже";  
unset($anotherVar);  
echo $myVar; // Выведет "Увидимся позже"
```

?>

Значение остается в памяти, пока вы не удалите *все* ссылки на него, в том числе в исходной переменной:

```
<?php
```

```
$myVar= "Привет!";  
$AnotherVar =& $myVar;  
$AnotherVar= "Увидимся позже";  
unset($anotherVar);  
unset( $myVar);  
echo $myVar; // Выведет "".
```

?>

Передача переменных в функцию по ссылке

Ссылки действительно работают, если вы передаёте их как аргументы функций. Как правило, когда вы передаете переменную в функцию, функция получает копию значения этой переменной. Передавая *ссылку* на переменную функция может сослаться на эту переменную, но что еще важнее, изменить исходную переменную.

Чтобы передать аргумент в качестве ссылки установите знак амперсанда перед именем параметра функции:

```
<?php
```

```
function myFunc( &$myParam) {
```

```
// Здесь что-нибудь сделаем с myParam.  
}
```

```
?>
```

Теперь, каждый раз при вызове `myFunc()` и передаче переменной PHP передаёт *ссылку* на переменную, а не на значение переменной. Рассмотрим простой пример передачи по ссылке:

```
<?php
```

```
function goodbye( &$greeting) {  
    $greeting= "Увидимся позже";  
}
```

```
$myVar= «Привет!»;
```

```
goodbye( $myVar); echo $myVar; // Выведет «Увидимся позже»
```

```
?>
```

Здесь мы создали функцию `goodbye()`, которая принимает ссылку на переменную. Ссылка хранится в параметре `$greeting`. Функция присваивает новое значение («Увидимся позже») переменной `$greeting`, которая изменяет значение сохранённое в переменной, которая была передана в функцию.

Мы это проверим, путем создания переменной `$myVar`, с первоначальным значением «Привет!», вызвав функцию `goodbye()` с параметром `$myVar`, переданным по ссылке. Функция `goodbye()` изменяет значение, сохраненное в `$myVar` на «Увидимся позже».

Таким образом, используйте передачу по ссылке всякий раз, когда вы хотите изменять переменную, которая передаётся в качестве аргумента функции. Просто не правда ли?! Кстати, не поддавайтесь искушению записать знак амперсанда перед аргументом при вызове функции:

```
<?php
```

```
goodbye( &$myVar); // Не делайте так!
```

?>

Запись знака амперсанда перед параметром в определении функции является достаточным для того, чтобы передать переменную путем ссылки. Много функций, встроенных в PHP используют передачу аргументов по ссылке. Например, функция `sort()` принимает ссылку для сортировки массива, так что эта функция может изменить порядок элементов в массиве.

Возвращение по ссылке из функций

Если можно передавать переменные по ссылке в функцию, то так же можно возвращать ссылки из функции. Для этого нужно записать знак амперсанда перед названием функции в её определении. Вы также должны записать знак амперсанда (`=&`) при присвоении функции переменной, в противном случае вы просто присвоите значение, а не ссылку.

Пример:

```
<?php

$numWidgets= 10;
function &getNumWidgets()
{
    global numWidgets;
    return $numWidgets;
}
$numwidgetsref =& getNumWidgets( );
$numwidgetsref--;
echo " \ $numWidgets = $numWidgets<br> "; // Выведет " 9 ".
echo " \ $numWidgetsRef = $numWidgetsRef<br> "; // выводит " 9
".

?>
```

В этом примере функция `getNumWidgets()` находит глобальную переменную `$numWidgets` и возвращает ссылку на неё. Затем мы вызываем `getNumWidgets()`, которая сохраняет возвращенные

ссылки в `$numWidgetsRef`, и уменьшает значение в переменной, на которую указывает `$numWidgetsRef`. Это то же самое значение, на которое указывает переменная `$numWidgets`, как можно увидеть из результата работы функции `echo`.

Возможно, вы не так часто пользуетесь возвращением по ссылке, как передачей по ссылке, но этот метод может быть полезен в некоторых ситуациях, например, когда вы хотите создать функцию поиска (или метод класса), которая находит переменную (или свойство класса) и возвращает ссылку на переменную или свойство, так чтобы вызываемый код мог управлять переменной или свойством.

Изменение значений в предложении `foreach` с помощью ссылок в PHP

Ещё один полезный пример использования ссылок для изменения значений в массиве с помощью цикла `foreach`. С помощью обычного цикла `foreach`, вы работаете *скопией* значений массива, так что, если вы измените её значения вы не затронете исходного массива. Например, попробуйте перевести в верхний регистр названия музыкальных групп в массиве с помощью цикла `foreach`:

```
<?php

$bands= array( "The Who", "The Beatles", "The Rolling Stones"
);
foreach ( $bands, as$band ) {
    $Band = strtoupper( $band);
}
echo " <pre> ";
Print_r( $bands);
echo " < /pre> ";

?>
```

Будет выведено следующее:

```
<?php
```

```
Array(  
  [0] => The Who  
  [1] => The Beatles  
  [2] => The Rolling Stones)  
  
?>
```

Как вы можете видеть, исходный массив не был изменён в результате работы цикла **foreach**. Вместе с тем, если мы ставим знак амперсанда до *\$band* в операторе **foreach** *\$band* становится ссылкой на исходный элемент массива, а не на его копию. Затем мы можем преобразовать элементы массива в верхний регистр:

```
<?php
```

```
$bands= array( "The Who", "The Beatles", "The Rolling Stones"  
);  
foreach ( $bands as &$band ) {  
  $Band = strtoupper( $band);  
}  
echo " <pre> ";  
Print_r( $bands);  
echo " < /pre> ";  
  
?>
```

Наш код теперь работает как и предполагалось, следующим образом:

```
<?php
```

```
Array(  
  [0] => THE WHO  
  [1] => THE BEATLES  
  [2] => THE ROLLING STONES)  
  
?>
```

Другой способ изменить значения массива в цикле является использование цикла **for** вместо **foreach**.

Когда ссылки используются автоматически

Итак, вы узнали четыре пути создания ссылки напрямую:

- Присваивание ссылки
- Передача по ссылке
- Возвращение по ссылке
- Создание ссылки в цикле **foreach**

Кроме того, есть случаи, когда PHP автоматически создает ссылки. В большинстве случаев это вам не понадобится, но знать об этой возможности будет полезно!

Ссылки в PHP при использовании ключевого слова **global**

Когда вы используете ключевое слово **global** внутри функции, чтобы иметь доступ к глобальной переменной, тогда, на самом деле, вы создаёте ссылку в глобальной переменной массива `$GLOBALS`.

Например:

```
<?php

function myFunction() { global $globalVariable;}

?>
```

Не одно и то же, что следующий пример:

```
<?php

function myFunction() {
    $globalVariable =&
    $GLOBALS["globalVariable"];}

?>
```

Ссылки в PHP, когда используется ключевое слово *\$this*

При [написании объектно-ориентированного кода](#) часто используется ключевое слово *\$this*. При использовании *\$this* в пределах метода объекта, выполняется указание на текущий объект. Стоит запомнить, что *\$this* всегда ссылается на объект, а не на его копию.

Например:

```
<?php

class MyClass {
    var $aProperty = 123;
    function aMethod() {
        $this->aProperty = 456; // $это ссылка на объект
    }
}

$myObject = new MyClass();
$myObject->aMethod();
echo $myObject->aProperty; // Выведет "456"

?>
```

В примере приведенном выше *\$this* – это ссылка на объект. Метод может изменять свойство объекта на новое значение в пределах этого объекта.

При передаче объектов

В отличие от других типов переменной, всякий раз, когда вы присваиваете, передаете или возвращаете объект, вы возвращаете ссылку на объект, а не на его копию. Как правило, передавая функцию или метод вы работаете с самим объектом, а не с его копией.

Например:

```
<?php
```

```
class MyClass {  
    // (Здесь происходит определение класса)  
}  
$myObject = new MyClass();  
$Yourobject = $myObject; // $yourObject и $myObject указывают  
на тот же объект  
  
?>
```

В некоторых ситуациях, когда вы действительно хотите *сделать* копию объекта, вы можете использовать ключевое слово **clone**. В сущности, всё намного тоньше. При создании переменной объекта, она содержит *указатель* на объект в памяти, а не на сам объект. При присвоении или передаче переменной вы на самом деле *создаёте* копию переменной. Но копия, также является просто указателем на объект – обе копии по-прежнему указывают на тот же объект. Таким образом, в большинстве случаев вы создаёте ссылки.

Краткий обзор статьи о ссылках в PHP

В этой статье были объяснены основы работы со ссылками в PHP. Вы изучили присвоение, передачу по ссылке, и возвращение по ссылке; научились использовать ссылки для изменения элементов массива в цикле **foreach**; и увидели ситуации, когда PHP создает ссылки автоматически.

Источник: www.elated.com

Этот блог читают уже много людей
- читай и ТЫ!

Да, Я тоже хочу читать статьи!