Search

Keyboard Shortcuts

?
: This help

j
: Next menu item

k
: Previous menu item

g p
: Previous man page

g n
: Next man page

G
: Scroll to bottom

g g
: Scroll to top

g h
: Goto homepage

g s
: Goto search
(current page)

/
: Focus search box

Чем ссылки не являются »
« Что такое ссылки

- Руководство по PHP
- Справочник языка
- Объяснение ссылок

Change language: Russian

Что делают ссылки

Есть три основных операции с использованием ссылок: присвоение по ссылке, передача по ссылке и возврат по ссылке. Данный раздел познакомит вас с этими операциями и предоставит ссылки для дальнейшего изучения.

Присвоение по ссылке

Первая из них - ссылки PHP позволяют создать две переменные указывающие на одно и то же значение. Таким образом, когда выполняется следующее:

```
<?php
$a =& $b;
?>
```

то *$a* указывает на то же значение что и *$b*.

> **Замечание**:
>
> *$a* и *$b* здесь абсолютно эквивалентны, но это не означает, что *$a* указывает на *$b* или наоборот. Это означает, что *$a* и *$b* указывают на одно и то же значение.

> **Замечание**:
>
> При присвоении, передаче или возврате неинициализированной переменной по ссылке, происходит её создание.

> **Пример #1 Использование ссылок с неинициализированными переменными**
>
> ```
> <?php
> function foo(&$var) { }
>
> foo($a); // $a создана и равна null
>
> $b = array();
> foo($b['b']);
> var_dump(array_key_exists('b', $b)); // bool(true)
>
> $c = new stdClass;
> foo($c->d);
> var_dump(property_exists($c, 'd')); // bool(true)
> ?>
> ```

Такой же синтаксис может использоваться в функциях, возвращающими ссылки, и с оператором `new`:

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

Использование того же синтаксиса с функцией, которая *не* возвращает по ссылке, приведёт к ошибке, так же как и её использование с результатом оператора new. Хотя объекты передаются как указатели, это не то же самое, что ссылки, как описано в разделе Объекты и ссылки.

Внимание

Если переменной, объявленной внутри функции как `global`, будет присвоена ссылка, она будет видна только в функции. Чтобы избежать этого, используйте массив *$GLOBALS*.

Пример #2 Присвоение ссылок глобальным переменным внутри функции

```
<?php
$var1 = "Пример переменной";
$var2 = "";

function global_references($use_globals)
{
```

```
global $var1, $var2;
if (!$use_globals) {
$var2 =& $var1; // только локально
} else {
$GLOBALS["var2"] =& $var1; // глобально
}
}

global_references(false);
echo "значение var2: '$var2'\n"; // значение var2: ''
global_references(true);
echo "значение var2: '$var2'\n"; // значение var2: 'Пример переменной'
?>
```

Думайте о `global $var;` как о сокращении от `$var =& $GLOBALS['var'];`. Таким образом, присвоение `$var` другой ссылки влияет лишь на локальную переменную.

> **Замечание**:
>
> При использовании переменной-ссылки в foreach, изменяется содержание, на которое она ссылается.
>
> **Пример #3 Ссылки и foreach**
>
> ```
> <?php
> $ref = 0;
> $row =& $ref;
> foreach (array(1, 2, 3) as $row) {
> // сделать что-нибудь
> }
> echo $ref; // 3 - последнее значение, используемое в цикле
> ?>
> ```

Хотя в выражениях, создаваемых с помощью конструкции `array()`, нет явного присвоения по ссылке, тем не менее они могут вести себя как таковые, если указать префикс `&` для элементов массива. Пример:

```
<?php
$a = 1;
$b = array(2, 3);
$arr = array(&$a, &$b[0], &$b[1]);
$arr[0]++; $arr[1]++; $arr[2]++;
/* $a == 2, $b == array(3, 4); */
?>
```

Однако следует отметить, что ссылки в массивах являются потенциально опасными. При обычном (не по ссылке) присвоении массива, ссылки внутри этого массива сохраняются. Это также относится и к вызовам функций, когда массив передаётся по значению. Пример:

```
<?php
/* Присвоение скалярных переменных */
$a = 1;
$b =& $a;
$c = $b;
$c = 7; //$c не ссылка и не изменяет значений $a и $b

/* Присвоение массивов */
$arr = array(1);
$a =& $arr[0]; // $a и $arr[0] ссылаются на одно значение
$arr2 = $arr; // присвоение не по ссылке!
$arr2[0]++;
/* $a == 2, $arr == array(2) */
/* Содержимое $arr изменилось, хотя было присвоено не по ссылке! */
?>
```

Иными словами, поведение отдельных элементов массива не зависит от типа присвоения этого массива.

Передача по ссылке

Второе, что делают ссылки - передача параметров по ссылке. При этом локальная переменная в функции и переменная в вызывающей области видимости ссылаются на одно и то же содержимое. Пример:

```
<?php
function foo(&$var) {
$var++;
}

$a = 5;
foo($a);
?>
```

Этот код присвоит *$a* значение 6. Это происходит, потому что в функции *foo* переменная *$var* ссылается на то же содержимое, что и переменная *$a*. Смотрите также детальное объяснение передачи по ссылке.

Возврат по ссылке

Третье, что могут делать ссылки - это возврат по ссылке.

User Contributed Notes 23 notes

64
ladoo at gmx dot at ¶
18 years ago
I ran into something when using an expanded version of the example of pbaltz at NO_SPAM dot cs dot NO_SPAM dot wisc dot edu below.
This could be somewhat confusing although it is perfectly clear if you have read the manual carfully. It makes the fact that references always point to the content of a variable perfectly clear (at least to me).

```
<?php
$a = 1;
$c = 2;
$b =& $a; // $b points to 1
$a =& $c; // $a points now to 2, but $b still to 1;
echo $a, " ", $b;
// Output: 2 1
?>
```


31
Hlavac ¶
16 years ago
Watch out for this:

```
foreach ($somearray as &$i) {
// update some $i...
}
...
foreach ($somearray as $i) {
// last element of $somearray is mysteriously overwritten!
}
```

Problem is $i contians reference to last element of $somearray after the first foreach, and the second foreach happily assigns to it!

23
elrah [] polyptych [dot] com ¶
13 years ago
It appears that references can have side-effects. Below are two examples. Both are simply copying one array to another. In

the second example, a reference is made to a value in the first array before the copy. In the first example the value at index 0 points to two separate memory locations. In the second example, the value at index 0 points to the same memory location.

I won't say this is a bug, because I don't know what the designed behavior of PHP is, but I don't think ANY developers would expect this behavior, so look out.

An example of where this could cause problems is if you do an array copy in a script and expect on type of behavior, but then later add a reference to a value in the array earlier in the script, and then find that the array copy behavior has unexpectedly changed.

```
<?php
// Example one
$arr1 = array(1);
echo "\nbefore:\n";
echo "\$arr1[0] == {$arr1[0]}\n";
$arr2 = $arr1;
$arr2[0]++;
echo "\nafter:\n";
echo "\$arr1[0] == {$arr1[0]}\n";
echo "\$arr2[0] == {$arr2[0]}\n";

// Example two
$arr3 = array(1);
$a =& $arr3[0];
echo "\nbefore:\n";
echo "\$a == $a\n";
echo "\$arr3[0] == {$arr3[0]}\n";
$arr4 = $arr3;
$arr4[0]++;
echo "\nafter:\n";
echo "\$a == $a\n";
echo "\$arr3[0] == {$arr3[0]}\n";
echo "\$arr4[0] == {$arr4[0]}\n";
?>
```

up
down
10
amp at gmx dot info ¶
16 years ago

Something that might not be obvious on the first look:
If you want to cycle through an array with references, you must not use a simple value assigning foreach control structure. You have to use an extended key-value assigning foreach or a for control structure.

A simple value assigning foreach control structure produces a copy of an object or value. The following code

```
$v1=0;
$arrV=array(&$v1,&$v1);
foreach ($arrV as $v)
{
$v1++;
echo $v."\n";
}
```

yields

```
0
1
```

which means $v in foreach is not a reference to $v1 but a copy of the object the actual element in the array was referencing to.

The codes

```
$v1=0;
$arrV=array(&$v1,&$v1);
foreach ($arrV as $k=>$v)
{
$v1++;
echo $arrV[$k]."\n";
}
```

and

```
$v1=0;
$arrV=array(&$v1,&$v1);
$c=count($arrV);
for ($i=0; $i<$c;$i++)
{
$v1++;
echo $arrV[$i]."\n";
}
```

both yield

```
1
2
```

and therefor cycle through the original objects (both $v1), which is, in terms of our aim, what we have been looking for.

(tested with php 4.1.3)

up
down
3

Anonymous ¶

8 years ago

to reply to ' elrah [] polyptych [dot] com ', one thing to keep in mind is that array (or similar large data holders) are by default passed by reference. So the behaviour is not side effect. And for array copy and passing array inside function always done by 'pass by reference'...

up
down
6

nay at woodcraftsrus dot com ¶

12 years ago

in PHP you don't really need pointer anymore if you want to share an object across your program

```
<?php
class foo{
protected $name;
function __construct($str){
$this->name = $str;
}
function __toString(){
return 'my name is "'. $this->name .'" and I live in "' . __CLASS__ . '".' . "\n";
}
function setName($str){
$this->name = $str;
}
}

class MasterOne{
protected $foo;
function __construct($f){
$this->foo = $f;
```

```
}
function __toString(){
return 'Master: ' . __CLASS__ . ' | foo: ' . $this->foo . "\n";
}
function setFooName($str){
$this->foo->setName( $str );
}
}

class MasterTwo{
protected $foo;
function __construct($f){
$this->foo = $f;
}
function __toString(){
return 'Master: ' . __CLASS__ . ' | foo: ' . $this->foo . "\n";
}
function setFooName($str){
$this->foo->setName( $str );
}
}

$bar = new foo('bar');

print("\n");
print("Only Created \$bar and printing \$bar\n");
print( $bar );

print("\n");
print("Now \$baz is referenced to \$bar and printing \$bar and \$baz\n");
$baz =& $bar;
print( $bar );

print("\n");
print("Now Creating MasterOne and Two and passing \$bar to both constructors\n");
$m1 = new MasterOne( $bar );
$m2 = new MasterTwo( $bar );
print( $m1 );
print( $m2 );

print("\n");
print("Now changing value of \$bar and printing \$bar and \$baz\n");
$bar->setName('baz');
print( $bar );
print( $baz );

print("\n");
print("Now printing again MasterOne and Two\n");
print( $m1 );
print( $m2 );

print("\n");
print("Now changing MasterTwo's foo name and printing again MasterOne and Two\n");
$m2->setFooName( 'MasterTwo\'s Foo' );
print( $m1 );
print( $m2 );

print("Also printing \$bar and \$baz\n");
print( $bar );
print( $baz );
?>
```

up

down
4
Oddant ¶
10 years ago
About the example on array references.
I think this should be written in the array chapter as well.
Indeed if you are new to programming language in some way, you should beware that arrays are pointers to a vector of Byte(s).

<?php $arr = array(1); ?>
$arr here contains a reference to which the array is located.
Writing :
<?php echo $arr[0]; ?>
dereferences the array to access its very first element.

Now something that you should also be aware of (even you are not new to programming languages) is that PHP use references to contains the different values of an array. And that makes sense because the type of the elements of a PHP array can be different.

Consider the following example :

```
<?php

$arr = array(1, 'test');

$point_to_test =& $arr[1];

$new_ref = 'new';

$arr[1] =& $new_ref;

echo $arr[1]; // echo 'new';
echo $point_to_test; // echo 'test' ! (still pointed somewhere in the memory)

?>
```

up
down
3
dovbysh at gmail dot com ¶
16 years ago
Solution to post "php at hood dot id dot au 04-Mar-2007 10:56":

```
<?php
$a1 = array('a'=>'a');
$a2 = array('a'=>'b');

foreach ($a1 as $k=>&$v)
$v = 'x';

echo $a1['a']; // will echo x

unset($GLOBALS['v']);

foreach ($a2 as $k=>$v)
{}

echo $a1['a']; // will echo x

?>
```

up
down
3

charles at org oo dot com ¶

16 years ago

points to post below me.
When you're doing the references with loops, you need to unset($var).

for example

```
<?php
foreach($var as &$value)
{
...
}
unset($value);
?>
```

up
down
2

php.devel at homelinkcs dot com ¶

19 years ago

In reply to lars at riisgaardribe dot dk,

When a variable is copied, a reference is used internally until the copy is modified. Therefore you shouldn't use references at all in your situation as it doesn't save any memory usage and increases the chance of logic bugs, as you discoved.

up
down
3

Amaroq ¶

14 years ago

I think a correction to my last post is in order.

When there is a constructor, the strange behavior mentioned in my last post doesn't occur. My guess is that php was treating reftest() as a constructor (maybe because it was the first function?) and running it upon instantiation.

```
<?php
class reftest
{
public $a = 1;
public $c = 1;

public function __construct()
{
return 0;
}

public function reftest()
{
$b =& $this->a;
$b++;
}

public function reftest2()
{
$d =& $this->c;
$d++;
}
}

$reference = new reftest();

$reference->reftest();
$reference->reftest2();
```

```
echo $reference->a; //Echoes 2.
echo $reference->c; //Echoes 2.
?>
```

up
down
1

akinaslan at gmail dot com ¶

13 years ago

In this example class name is different from its first function and however there is no construction function. In the end as you guess "a" and "c" are equal. So if there is no construction function at same time class and its first function names are the same, "a" and "c" doesn't equal forever. In my opinion php doesn't seek any function for the construction as long as their names differ from each others.

```
<?php
class reftest_new
{
public $a = 1;
public $c = 1;

public function reftest()
{
$b =& $this->a;
$b++;
}

public function reftest2()
{
$d =& $this->c;
$d++;
}
}

$reference = new reftest_new();

$reference->reftest();
$reference->reftest2();

echo $reference->a; //Echoes 2.
echo $reference->c; //Echoes 2.
?>
```

up
down
2

dexant9t at gmail dot com ¶

2 years ago

It matters if you are playing with a reference or with a value

Here we are working with values so working on a reference updates original variable too;

```
$a = 1;
$c = 22;

$b = & $a;
echo "$a, $b"; //Output: 1, 1

$b++;
echo "$a, $b";//Output: 2, 2 both values are updated

$b = 10;
echo "$a, $b";//Output: 10, 10 both values are updated

$b =$c; //This assigns value 2 to $b which also updates $a
```

```
echo "$a, $b";//Output: 22, 22

But, if instead of $b=$c you do
$b = &$c; //Only value of $b is updated, $a still points to 10, $b serves now reference to variable $c

echo "$a, $b"//Output: 10, 22
```

up
down
1

Amaroq ¶

15 years ago

```
The order in which you reference your variables matters.

<?php
$a1 = "One";
$a2 = "Two";
$b1 = "Three";
$b2 = "Four";

$b1 =& $a1;
$a2 =& $b2;

echo $a1; //Echoes "One"
echo $b1; //Echoes "One"

echo $a2; //Echoes "Four"
echo $b2; //Echoes "Four"
?>
```

up
down
1

Drewseph ¶

15 years ago

```
If you set a variable before passing it to a function that takes a variable as a reference, it is much harder (if not
impossible) to edit the variable within the function.

Example:
<?php
function foo(&$bar) {
$bar = "hello\n";
}

foo($unset);
echo($unset);
foo($set = "set\n");
echo($set);

?>

Output:
hello
set

It baffles me, but there you have it.
```

up
down
0

dnhuff at acm dot org ¶

15 years ago

```
In reply to Drewseph using foo($a = 'set'); where $a is a reference formal parameter.

$a = 'set' is an expression. Expressions cannot be passed by reference, don't you just hate that, I do. If you turn on
```

```
error reporting for E_NOTICE, you will be told about it.

Resolution: $a = 'set'; foo($a); this does what you want.
```

up
down
0

firespade at gmail dot com ¶

16 years ago

```
Here's a good little example of referencing. It was the best way for me to understand, hopefully it can help others.

$b = 2;
$a =& $b;
$c = $a;
echo $c;

// Then... $c = 2
```

up
down
-1

Amaroq ¶

14 years ago

```
When using references in a class, you can reference $this-> variables.

<?php
class reftest
{
public $a = 1;
public $c = 1;

public function reftest()
{
$b =& $this->a;
$b = 2;
}

public function reftest2()
{
$d =& $this->c;
$d++;
}
}

$reference = new reftest();

$reference->reftest();
$reference->reftest2();

echo $reference->a; //Echoes 2.
echo $reference->c; //Echoes 2.
?>

However, this doesn't appear to be completely trustworthy. In some cases, it can act strangely.

<?php
class reftest
{
public $a = 1;
public $c = 1;

public function reftest()
{
$b =& $this->a;
```

```
$b++;
}

public function reftest2()
{
$d =& $this->c;
$d++;
}
}

$reference = new reftest();

$reference->reftest();
$reference->reftest2();

echo $reference->a; //Echoes 3.
echo $reference->c; //Echoes 2.
?>
```

In this second code block, I've changed reftest() so that $b increments instead of just gets changed to 2. Somehow, it winds up equaling 3 instead of 2 as it should.

up
down
-1

php at hood dot id dot au ¶

16 years ago

I discovered something today using references in a foreach

```
<?php
$a1 = array('a'=>'a');
$a2 = array('a'=>'b');

foreach ($a1 as $k=>&$v)
$v = 'x';

echo $a1['a']; // will echo x

foreach ($a2 as $k=>$v)
{}

echo $a1['a']; // will echo b (!)
?>
```

After reading the manual this looks like it is meant to happen. But it confused me for a few days!

(The solution I used was to turn the second foreach into a reference too)

up
down
-3

strata_ranger at hotmail dot com ¶

14 years ago

An interesting if offbeat use for references: Creating an array with an arbitrary number of dimensions.

For example, a function that takes the result set from a database and produces a multidimensional array keyed according to one (or more) columns, which might be useful if you want your result set to be accessible in a hierarchial manner, or even if you just want your results keyed by the values of each row's primary/unique key fields.

```
<?php
function array_key_by($data, $keys, $dupl = false)
/*
* $data - Multidimensional array to be keyed
* $keys - List containing the index/key(s) to use.
```

```
* $dupl - How to handle rows containing the same values. TRUE stores it as an Array, FALSE overwrites the previous row.
*
* Returns a multidimensional array indexed by $keys, or NULL if error.
* The number of dimensions is equal to the number of $keys provided (+1 if $dupl=TRUE).
*/
{
// Sanity check
if (!is_array($data)) return null;

// Allow passing single key as a scalar
if (is_string($keys) or is_integer($keys)) $keys = Array($keys);
elseif (!is_array($keys)) return null;

// Our output array
$out = Array();

// Loop through each row of our input $data
foreach($data as $cx => $row) if (is_array($row))
{

// Loop through our $keys
foreach($keys as $key)
{
$value = $row[$key];

if (!isset($last)) // First $key only
{
if (!isset($out[$value])) $out[$value] = Array();
$last =& $out; // Bind $last to $out
}
else // Second and subsequent $key....
{
if (!isset($last[$value])) $last[$value] = Array();
}

// Bind $last to one dimension 'deeper'.
// First lap: was &$out, now &$out[...]
// Second lap: was &$out[...], now &$out[...][...]
// Third lap: was &$out[...][...], now &$out[...][...][...]
// (etc.)
$last =& $last[$value];
}

if (isset($last))
{
// At this point, copy the $row into our output array
if ($dupl) $last[$cx] = $row; // Keep previous
else $last = $row; // Overwrite previous
}
unset($last); // Break the reference
}
else return NULL;

// Done
return $out;
}

// A sample result set to test the function with
$data = Array(Array('name' => 'row 1', 'foo' => 'foo_a', 'bar' => 'bar_a', 'baz' => 'baz_a'),
Array('name' => 'row 2', 'foo' => 'foo_a', 'bar' => 'bar_a', 'baz' => 'baz_b'),
Array('name' => 'row 3', 'foo' => 'foo_a', 'bar' => 'bar_b', 'baz' => 'baz_c'),
Array('name' => 'row 4', 'foo' => 'foo_b', 'bar' => 'bar_c', 'baz' => 'baz_d')
```

```
);

// First, let's key it by one column (result: two-dimensional array)
print_r(array_key_by($data, 'baz'));

// Or, key it by two columns (result: 3-dimensional array)
print_r(array_key_by($data, Array('baz', 'bar')));

// We could also key it by three columns (result: 4-dimensional array)
print_r(array_key_by($data, Array('baz', 'bar', 'foo')));

?>
```

up
down
-3

joachim at lous dot org ¶

20 years ago

So to make a by-reference setter function, you need to specify reference semantics _both_ in the parameter list _and_ the assignment, like this:

```
class foo{
var $bar;
function setBar(&$newBar){
$this->bar =& newBar;
}
}
```

Forget any of the two '&'s, and $foo->bar will end up being a copy after the call to setBar.

up
down
-4

butshuti at smartrwanda dot org ¶

10 years ago

This appears to be the hidden behavior: When a class function has the same name as the class, it seems to be implicitly called when an object of the class is created.
For instance, you may take a look at the naming of the function "reftest()": it is in the class "reftest". The behavior can be tested as follows:

```
<?php
class reftest
{
public $a = 1;
public $c = 1;

public function reftest1()
{
$b =& $this->a;
$b++;
}

public function reftest2()
{
$d =& $this->c;
$d++;
}

public function reftest()
{
echo "REFTEST() called here!\n";
}
}
```

```
$reference = new reftest();
/*You must notice the above will also implicitly call reference->reftest()*/

$reference->reftest1();
$reference->reftest2();

echo $reference->a."\n"; //Echoes 2, not 3 as previously noticed.
echo $reference->c."\n"; //Echoes 2.
?>

The above outputs:

REFTEST() called here!
2
2

Notice that reftest() appears to be called (though no explicit call to it was made)!
```

up
down
-6

admin at torntech dot com ¶
10 years ago

```
Something that has not been discussed so far is reference of a reference.
I needed a quick and dirty method of aliasing incorrect naming until a proper rewrite could be done.
Hope this saves someone else the time of testing since it was not covered in the Does/Are/Are Not pages.
Far from best practice, but it worked.

<?php
$a = 0;

$b =& $a;
$a =& $b;

$a = 5;
echo $a . ', ' . $b;
//ouputs: 5,5

echo ' | ';

$b = 6;
echo $a . ',' . $b;
//outputs: 6,6

echo ' | ';
unset( $a );
echo $a . ', ' . $b;

//outputs: , 6

class Product {

public $id;
private $productid;

public function __construct( $id = null ) {
$this->id =& $this->productid;
$this->productid =& $this->id;
$this->id = $id;
}

public function getProductId() {
return $this->productid;
```

```
}

}

echo ' | ';

$Product = new Product( 1 );
echo $Product->id . ', ' . $Product->getProductId();
//outputs 1, 1
$Product->id = 2;
echo ' | ';
echo $Product->id . ', ' . $Product->getProductId();
//outputs 2, 2
$Product->id = null;
echo ' | ';
echo $Product->id . ', ' . $Product->getProductId();
//outouts ,
```