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

Сброс переменных-ссылок »
« Передача по ссылке

- Руководство по PHP
- Справочник языка
- Объяснение ссылок

Change language: Russian

# Возврат по ссылке

Возврат по ссылке используется в тех случаях, когда вы хотите использовать функцию для выбора переменной, с которой должна быть связана данная ссылка. *Не* используйте возврат по ссылке для увеличения производительности. Ядро PHP само занимается оптимизацией. Применяйте возврат по ссылке только имея технические причины на это. Для возврата по ссылке используйте такой синтаксис:

```
<?php
class foo {
public $value = 42;

public function &getValue() {
return $this->value;
}
}

$obj = new foo;
$myValue = &$obj->getValue(); // $myValue указывает на $obj->value, равное 42.
$obj->value = 2;
echo $myValue; // отобразит новое значение $obj->value, то есть 2.
?>
```

В этом примере устанавливается свойство объекта, возвращённого функцией *getValue*, а не его копии, как было бы без использования ссылок.

> **Замечание**: В отличие от передачи параметров по ссылке, `&` здесь нужно использовать в обоих местах - для указания на то, что вы возвращаете ссылку, а не копию, как обычно, и для указания того, что происходит связывание по ссылке, а не обычное присвоение для *$myValue*.

> **Замечание**: Если вы возвращаете ссылку из функции, используя следующий синтаксис: `return ($this->value);`, это *не* будет работать, так как вы возвращаете по ссылке результат *выражения*, а не переменную. По ссылке можно возвращать только переменные и ничего больше.

Для использования возвращаемой ссылки вы должны применять присвоение по ссылке:

```
<?php
function &collector() {
static $collection = array();
return $collection;
}
$collection = &collector();
$collection[] = 'foo';
?>
```

Для передачи возвращаемой ссылки в другую функцию, принимающую ссылку, вы можете использовать следующий синтаксис:

```
<?php
function &collector() {
static $collection = array();
return $collection;
}
array_push(collector(), 'foo');
?>
```

> **Замечание**: Заметим, что `array_push(&collector(), 'foo');` *не* сработает, а приведёт к неисправимой ошибке.



## User Contributed Notes 19 notes


111
***Spad-XIII ¶***

**16 years ago**

a little addition to the example of pixel at minikomp dot com here below

```
<?php

function &func(){
static $static = 0;
$static++;
return $static;
}

$var1 =& func();
echo "var1:", $var1; // 1
func();
func();
echo "var1:", $var1; // 3
$var2 = func(); // assignment without the &
echo "var2:", $var2; // 4
func();
func();
echo "var1:", $var1; // 6
echo "var2:", $var2; // still 4

?>
```

up
down
19

***stanlemon at mac dot com ¶***

**16 years ago**

I haven't seen anyone note method chaining in PHP5. When an object is returned by a method in PHP5 it is returned by default as a reference, and the new Zend Engine 2 allows you to chain method calls from those returned objects. For example consider this code:

```
<?php

class Foo {

protected $bar;

public function __construct() {
$this->bar = new Bar();

print "Foo\n";
}

public function getBar() {
return $this->bar;
}
}

class Bar {

public function __construct() {
print "Bar\n";
}

public function helloWorld() {
print "Hello World\n";
}
}

function test() {
return new Foo();
```

```
}

test()->getBar()->helloWorld();

?>
```

Notice how we called test() which was not on an object, but returned an instance of Foo, followed by a method on Foo, getBar() which returned an instance of Bar and finally called one of its methods helloWorld(). Those familiar with other interpretive languages (Java to name one) will recognize this functionality. For whatever reason this change doesn't seem to be documented very well, so hopefully someone will find this helpful.

up
down
18

***szymoncofalik at gmail dot com ¶***

**13 years ago**

Sometimes, you would like to return NULL with a function returning reference, to indicate the end of chain of elements. However this generates E_NOTICE. Here is little tip, how to prevent that:

```
<?php
class Foo {
const $nullGuard = NULL;
// ... some declarations and definitions
public function &next() {
// ...
if (!$end) return $bar;
else return $this->nullGuard;
}
}
?>
```

by doing this you can do smth like this without notices:

```
<?php
$f = new Foo();
// ...
while (($item = $f->next()) != NULL) {
// ...
}
?>
```

you may also use global variable:

```
global $nullGuard;
return $nullGuard;
```

up
down
9

***sandaimespaceman at gmail dot com ¶***

**15 years ago**

The &b() function returns a reference of $a in the global scope.

```
<?php
$a = 0;
function &b()
{
global $a;
return $a;
}
$c = &b();
$c++;
echo
"
\$a: $a
```

```
\$b: $c
"
?>

It outputs:

$a: 1 $b: 1
```

up
down
12

***obscvresovl at NOSPAM dot hotmail dot com ¶***

**19 years ago**

An example of returning references:

```
<?

$var = 1;
$num = NULL;

function &blah()
{
$var =& $GLOBALS["var"]; # the same as global $var;
$var++;
return $var;
}

$num = &blah();

echo $num; # 2

blah();

echo $num; # 3

?>
```

Note: if you take the & off from the function, the second echo will be 2, because without & the var $num contains its returning value and not its returning reference.

up
down
2

***rwruck ¶***

**17 years ago**

The note about using parentheses when returning references is only true if the variable you try to return does not already contain a reference.

```
<?php
// Will return a reference
function& getref1()
{
$ref =& $GLOBALS['somevar'];
return ($ref);
}

// Will return a value (and emit a notice)
function& getref2()
{
$ref = 42;
return ($ref);
}

// Will return a reference
```

```
function& getref3()
{
static $ref = 42;
return ($ref);
}
?>
```

up
down
1

***civilization28 at gmail dot com ¶***

**9 years ago**

Zayfod's example above is useful, but I feel that it needs more explanation. The point that should be made is that a parameter passed in by reference can be changed to reference something else, resulting in later changes to the local variable not affecting the passed in variable:

```
<?php

function & func_b ()
{
$some_var = 2;
return $some_var;
}

function func_a (& $param)
{
# $param is 1 here
$param = & func_b(); # Here the reference is changed and
# the "&" in "func_a (& $param)"
# is no longer in effect at all.
# $param is 2 here
$param++; # Has no effect on $var.
}

$var = 1;
func_a($var);
# $var is still 1 here!!! Because the reference was changed.

?>
```

up
down
1

***hawcue at yahoo dot com ¶***

**19 years ago**

Be careful when using tinary operation condition?value1:value2

See the following code:

```
$a=1;
function &foo()
{
global $a;
return isset($a)?$a:null;
}
$b=&foo();
echo $b; // shows 1
$b=2;
echo $a; // shows 1 (not 2! because $b got a copy of $a)
```

To let $b be a reference to $a, use "if..then.." in the function.

up
down
-1

***anisgazig at gmail dot com ¶***

**1 year ago**

```
<?php

$a = 9;
function &myF(){
global $a;
return $a;
}

//before modified the value
$func =& myF();
echo "$a and $func";
echo "\n";

//after modified the value
$func++;
echo "$a and $func";
```

up

down

0

***fabian dot picone at gmail dot com ¶***

**5 years ago**

This note seems not to apply with PHP 7:

"Note: If you try to return a reference from a function with the syntax: return ($this->value); this will not work as you are attempting to return the result of an expression, and not a variable, by reference. You can only return variables by reference from a function - nothing else. Since PHP 5.1.0, an E_NOTICE error is issued if the code tries to return a dynamic expression or a result of the new operator."

Bug following code works without error output. Same result as i would not have braces around $this-value.

```
<?php

class foo {
public $value = 42;

public function &getValue() {
return ($this->value);
}
}

$obj = new foo;
$myValue = &$obj->getValue();
$obj->value = 2;
echo $myValue;
```

up

down

0

***benjamin dot delespierre at gmail dot com ¶***

**12 years ago**

Keep in mind that returning by reference doesn't work with __callStatic:

```
<?php
class Test {
private static $_inst;
public static function & __callStatic ($name, $args) {
if (!isset(static::$_inst)){
echo "create";
static::$_inst = (object)"test";
}
return static::$_inst;
```

```
}

var_dump($a = &Test::abc()); // prints 'create'
$a = null;
var_dump(Test::abc()); // doesn't prints and the instance still exists in Test::$_inst
?>
```

up
down
-1

***spidgorny at gmail dot com ¶***

**13 years ago**

When returning reference to the object member which is instantiated inside the function, the object is destructed upon returning (which is a problem). It's easier to see the code:

```
<?php

class MemcacheArray {
public $data;

...

/**
* Super-clever one line cache reading AND WRITING!
* Usage $data = &MemcacheArray::getData(__METHOD__);
* Hopefully PHP will know that $this->data is still used
* and will call destructor after data changes.
* Ooops, it's not the case.
*
* @return unknown
*/
function &getData($file, $expire = 3600) {
$o = new MemcacheArray($file, $expire);
return $o->data;
}
?>
```

Here, destructor is called upon return() and the reference becomes a normal variable.

My solution is to store objects in a pool until the final exit(), but I don't like it. Any other ideas?

```
<?php
protected static $instances = array();

function &getData($file, $expire = 3600) {
$o = new MemcacheArray($file, $expire);
self::$instances[$file] = $o; // keep object from destructing too early
return $o->data;
}
?>
```

up
down
-1

***zayfod at yahoo dot com ¶***

**20 years ago**

There is a small exception to the note on this page of the documentation. You do not have to use & to indicate that reference binding should be done when you assign to a value passed by reference the result of a function which returns by reference.

Consider the following two exaples:

```
<?php
```

```
function & func_b ()
{
$some_var = 2;
return $some_var;
}

function func_a (& $param)
{
# $param is 1 here
$param = & func_b();
# $param is 2 here
}

$var = 1;
func_a($var);
# $var is still 1 here!!!

?>
```

The second example works as intended:

```
<?php

function & func_b ()
{
$some_var = 2;
return $some_var;
}

function func_a (& $param)
{
# $param is 1 here
$param = func_b();
# $param is 2 here
}

$var = 1;
func_a($var);
# $var is 2 here as intended

?>
```

(Experienced with PHP 4.3.0)

up
down
-1

***php at thunder-2000 dot com ¶***

**17 years ago**

If you want to get a part of an array to manipulate, you can use this function

```
function &getArrayField(&$array,$path) {
if (!empty($path)) {
if (empty($array[$path[0]])) return NULL;
else return getArrayField($array[$path[0]], array_slice($path, 1));
} else {
return $array;
}
}
```

Use it like this:

```
$partArray =& getArrayField($GLOBALS,array("config","modul1"));
```

You can manipulate $partArray and the changes are also made with $GLOBALS.

up
down
-2

***Anonymous ¶***

**9 years ago**

I learned a painful lesson working with a class method that would pass by reference.

In short, if you have a method in a class that is initialed with ampersand during declaration, do not use another ampersand when using the method as in &$this->method();

For example

```
<?php
class A {
public function &hello(){
static $a='';
return $a;
}
public function bello(){
$b=&$this->hello(); // incorrect. Do not use ampersand.
$b=$this->hello(); // $b is a reference to the static variable.
}
?>
```

up
down
-2

***contact at infopol dot fr ¶***

**20 years ago**

A note about returning references embedded in non-reference arrays :

```
<?
$foo;

function bar () {
global $foo;
$return = array();
$return[] =& $foo;
return $return;
}

$foo = 1;
$foobar = bar();
$foobar[0] = 2;
echo $foo;
?>
```

results in "2" because the reference is copied (pretty neat).

up
down
-3

***willem at designhulp dot nl ¶***

**18 years ago**

There is an important difference between php5 and php4 with references.

Lets say you have a class with a method called 'get_instance' to get a reference to an exsisting class and it's properties.

```
<?php
class mysql {
function get_instance(){
// check if object exsists
```

```
if(empty($_ENV['instances']['mysql'])){
// no object yet, create an object
$_ENV['instances']['mysql'] = new mysql;
}
// return reference to object
$ref = &$_ENV['instances']['mysql'];
return $ref;
}
}
?>
```

Now to get the exsisting object you can use
`mysql::get_instance();`

Though this works in php4 and in php5, but in php4 all data will be lost as if it is a new object while in php5 all properties in the object remain.

up
down
-2

***jpenna ¶***

**3 years ago**

You can set the value of the variable returned by reference, be it a `static` function variable or a `private` property of an object (which is quite dangerous o.o).

Static function variable:

```
<?php
function &func(){
static $static = 0;
return $static;
}

$var1 =& func();
echo "var1:", $var1, "\n"; // 0
func();

$var1 = 90;
echo "var1:", $var1, "\n"; // 90
echo "static:", func(), "\n"; // 90
?>
```

Private property

```
<?php
class foo {
private $value = 1;

public function &getValue() {
return $this->value;
}

public function setValue($val) {
$this->value = $val;
}
}

$obj = new foo;
$myValue = &$obj->getValue(); // $myValue is a reference to $obj->value, which is 1.
echo $obj->getValue(); // 1
echo $myValue; // 1
$obj->setValue(5);
echo $obj->getValue(); // 5
```

```
echo $myValue; // 5
$myValue = 1000;
echo $obj->getValue(); // 1000
echo $myValue; // 1000
?>
```

up
down
-11
***pixel at minikomp dot com ¶***
**16 years ago**

```
<?php

function &func(){
static $static = 0;
$static++;
return $static;
}

$var =& func();
echo $var; // 1
func();
func();
func();
func();
echo $var; // 5

?>
```

＋add a note