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

Предопределённые переменные »
« Сброс переменных-ссылок

- Руководство по PHP
- Справочник языка
- Объяснение ссылок

Change language: Russian

Неявное использование механизма ссылок

Многие синтаксические конструкции PHP реализованы через механизм ссылок, поэтому всё сказанное выше о ссылочном связывании применимо также и к этим конструкциям. Некоторые конструкции, вроде передающих и возвращающих по ссылке, рассмотрены ранее. Другие конструкции, использующие ссылки:

Ссылки global

Если вы объявляете переменную как **global $var**, вы фактически создаёте ссылку на глобальную переменную. Это означает то же самое, что и:

```
<?php
$var =& $GLOBALS["var"];
?>
```

Это значит, например, что сброс (unset) *$var* не приведёт к сбросу глобальной переменной.

User Contributed Notes 8 notes

23
BenBE at omorphia dot de ¶
17 years ago

```
Hi,

If you want to check if two variables are referencing each other (i.e. point to the same memory) you can use a function
like this:

<?php

function same_type(&$var1, &$var2){
return gettype($var1) === gettype($var2);
}

function is_ref(&$var1, &$var2) {
//If a reference exists, the type IS the same
if(!same_type($var1, $var2)) {
return false;
}

$same = false;

//We now only need to ask for var1 to be an array ;-)
if(is_array($var1)) {
//Look for an unused index in $var1
do {
$key = uniqid("is_ref_", true);
} while(array_key_exists($key, $var1));

//The two variables differ in content ... They can't be the same
if(array_key_exists($key, $var2)) {
return false;
}

//The arrays point to the same data if changes are reflected in $var2
$data = uniqid("is_ref_data_", true);
$var1[$key] =& $data;
//There seems to be a modification ...
```

```
if(array_key_exists($key, $var2)) {
if($var2[$key] === $data) {
$same = true;
}
}

//Undo our changes ...
unset($var1[$key]);
} elseif(is_object($var1)) {
//The same objects are required to have equal class names ;-)
if(get_class($var1) !== get_class($var2)) {
return false;
}

$obj1 = array_keys(get_object_vars($var1));
$obj2 = array_keys(get_object_vars($var2));

//Look for an unused index in $var1
do {
$key = uniqid("is_ref_", true);
} while(in_array($key, $obj1));

//The two variables differ in content ... They can't be the same
if(in_array($key, $obj2)) {
return false;
}

//The arrays point to the same data if changes are reflected in $var2
$data = uniqid("is_ref_data_", true);
$var1->$key =& $data;
//There seems to be a modification ...
if(isset($var2->$key)) {
if($var2[$key] === $data) {
$same = true;
}
}

//Undo our changes ...
unset($var1->$key);
} elseif (is_resource($var1)) {
if(get_resource_type($var1) !== get_resource_type($var2)) {
return false;
}

return ((string) $var1) === ((string) $var2);
} else {
//Simple variables ...
if($var1!==$var2) {
//Data mismatch ... They can't be the same ...
return false;
}

//To check for a reference of a variable with simple type
//simply store its old value and check against modifications of the second variable ;-)

do {
$key = uniqid("is_ref_", true);
} while($key === $var1);

$tmp = $var1; //WE NEED A COPY HERE!!!
$var1 = $key; //Set var1 to the value of $key (copy)
$same = $var1 === $var2; //Check if $var2 was modified too ...
```

```
$var1 = $tmp; //Undo our changes ...
}

return $same;
}

?>
```

Although this implementation is quite complete, it can't handle function references and some other minor stuff ATM.
This function is especially useful if you want to serialize a recursive array by hand.

The usage is something like:

```
<?php
$a = 5;
$b = 5;
var_dump(is_ref($a, $b)); //false

$a = 5;
$b = $a;
var_dump(is_ref($a, $b)); //false

$a = 5;
$b =& $a;
var_dump(is_ref($a, $b)); //true
echo "---\n";

$a = array();
var_dump(is_ref($a, $a)); //true

$a[] =& $a;
var_dump(is_ref($a, $a[0])); // true
echo "---\n";

$b = array(array());
var_dump(is_ref($b, $b)); //true
var_dump(is_ref($b, $b[0])); //false
echo "---\n";

$b = array();
$b[] = $b;
var_dump(is_ref($b, $b)); //true
var_dump(is_ref($b, $b[0])); //false
var_dump(is_ref($b[0], $b[0][0])); //true*
echo "---\n";

var_dump($a);
var_dump($b);

?>
```

* Please note the internal behaviour of PHP that seems to do the reference assignment BEFORE actually copying the variable!!! Thus you get an array containing a (different) recursive array for the last testcase, instead of an array containing an empty array as you could expect.

BenBE.

up
down
8

ludvig dot ericson at gmail dot com ¶

17 years ago

For the sake of clarity:

$this is a PSEUDO VARIABLE - thus not a real variable. ZE treats is in other ways then normal variables, and that means that some advanced variable-things won't work on it (for obvious reasons):

```
<?php
class Test {
var $monkeys = 0;

function doFoobar() {
$var = "this";
$$var->monkeys++; // Will fail on this line.
}
}

$obj = new Test;
$obj->doFoobar(); // Will fail in this call.
var_dump($obj->monkeys); // Will return int(0) if it even reaches here.
?>
```

up
down
3
ksamvel at gmail dot com ¶
18 years ago

One may check reference to any object by simple operator==(object). Example:

```
<?php
class A {}

$oA1 = new A();

$roA = & $oA1;

echo "roA and oA1 are " . ( $roA == $oA1 ? "same" : "not same") . "<br>";

$oA2 = new A();
$roA = & $roA2;

echo "roA and oA1 are " . ( $roA == $oA1 ? "same" : "not same") . "<br>";
?>
```

Output:

```
roA and oA1 are same
roA and oA1 are not same
```

Current technique might be useful for caching in objects when inheritance is used and only base part of extended class should be copied (analog of C++: oB = oA):

```
<?php
class A {
/* Any function changing state of A should set $bChanged to true */
public function isChanged() { return $this->bChanged; }
private $bChanged;
//...
}

class B extends A {
// ...
public function set( &$roObj) {
if( $roObj instanceof A) {
if( $this->roAObj == $roObj &&
$roObj->isChanged()) {
```

```
/* Object was not changed do not need to copy A part of B */
} else {
/* Copy A part of B */
$this->roAObj = &$roObj;
}
}
}

private $roAObj;
}
?>
```

up
down
2

Sergio Santana: ssantana at tlaloc dot imta dot mx ¶

18 years ago

Sometimes an object's method returning a reference to itself is required. Here is a way to code it:

```
<?php
class MyClass {
public $datum;
public $other;

function &MyRef($d) { // the method
$this->datum = $d;
return $this; // returns the reference
}
}

$a = new MyClass;
$b = $a->MyRef(25); // creates the reference

echo "This is object \$a: \n";
print_r($a);
echo "This is object \$b: \n";
print_r($b);

$b->other = 50;

echo "This is object \$a, modified" .
" indirectly by modifying ref \$b: \n";
print_r($a);
?>

This code outputs:
This is object $a:
MyClass Object
(
[datum] => 25
[other] =>
)
This is object $b:
MyClass Object
(
[datum] => 25
[other] =>
)
This is object $a, modified indirectly by modifying ref $b:
MyClass Object
(
[datum] => 25
[other] => 50
```

```
)
```
up
down
0
CodeWorX.ch ¶
12 years ago
here is an unconventional (and not very fast) way of detecting references within arrays:

```
<?php

function is_array_reference ($arr, $key) {
$isRef = false;
ob_start();
var_dump($arr);
if (strpos(preg_replace("/[ \n\r]*/i", "", preg_replace("/( ){4,}.*(\n\r)*/i", "", ob_get_contents())), "[" . $key .
"]=>&") !== false)
$isRef = true;
ob_end_clean();
return $isRef;
}

?>
```
up
down
0
Sergio Santana: ssantana at tlaloc dot imta dot mx ¶
18 years ago
```
*** WARNING about OBJECTS TRICKY REFERENCES ***
-----------------------------------------------
The use of references in the context of classes
and objects, though well defined in the documentation,
is somehow tricky, so one must be very careful when
using objects. Let's examine the following two
examples:

<?php
class y {
public $d;
}

$A = new y;
$A->d = 18;
echo "Object \$A before operation:\n";
var_dump($A);

$B = $A; // This is not an explicit (=&) reference assignment,
// however, $A and $B refer to the same instance
// though they are not equivalent names
$C =& $A; // Explicit reference assignment, $A and $C refer to
// the same instance and they have become equivalent
// names of the same instance

$B->d = 1234;

echo "\nObject \$B after operation:\n";
var_dump($B);
echo "\nObject \$A implicitly modified after operation:\n";
var_dump($A);
echo "\nObject \$C implicitly modified after operation:\n";
var_dump($C);

// Let's make $A refer to another instance
```

```
$A = new y;
$A->d = 25200;
echo "\nObject \$B after \$A modification:\n";
var_dump($B); // $B doesn't change
echo "\nObject \$A after \$A modification:\n";
var_dump($A);
echo "\nObject \$C implicitly modified after \$A modification:\n";
var_dump($C); // $C changes as $A changes
?>
```

Thus, note the difference between assignments $X = $Y and $X =& $Y.
When $Y is anything but an object instance, the first assignment means
that $X will hold an independent copy of $Y, and the second, means that
$X and $Y will refer to the same thing, so they are tight together until
either $X or $Y is forced to refer to another thing. However, when $Y
happens to be an object instance, the semantic of $X = $Y changes and
becomes only slightly different to that of $X =& $Y, since in both
cases $X and $Y become references to the same object. See what this
example outputs:

```
Object $A before operation:
object(y)#1 (1) {
["d"]=>
int(18)
}

Object $B after operation:
object(y)#1 (1) {
["d"]=>
int(1234)
}

Object $A implicitly modified after operation:
object(y)#1 (1) {
["d"]=>
int(1234)
}

Object $C implicitly modified after operation:
object(y)#1 (1) {
["d"]=>
int(1234)
}

Object $B after $A modification:
object(y)#1 (1) {
["d"]=>
int(1234)
}

Object $A after $A modification:
object(y)#2 (1) {
["d"]=>
int(25200)
}

Object $C implicitly modified after $A modification:
object(y)#2 (1) {
["d"]=>
int(25200)
}
```

```
Let's review a SECOND EXAMPLE:
<?php
class yy {
public $d;
function yy($x) {
$this->d = $x;
}
}

function modify($v)
{
$v->d = 1225;
}

$A = new yy(3);
var_dump($A);
modify($A);
var_dump($A);
?>

Although, in general, a formal argument declared
as $v in the function 'modify' shown above, implies
that the actual argument $A, passed when calling
the function, is not modified, this is not the
case when $A is an object instance. See what the
example code outputs when executed:

object(yy)#3 (1) {
["d"]=>
int(3)
}
object(yy)#3 (1) {
["d"]=>
int(1225)
}
```

up
down
-2

Abimael Rodrguez Coln ¶

12 years ago

```
This is one way to check if is a reference
<?php
$a = 1;
$b =& $a;
$c = 2;
$d = 3;
$e = array($a);
function is_reference($var){
$val = $GLOBALS[$var];
$tmpArray = array();
/**
* Add keys/values without reference
*/
foreach($GLOBALS as $k => $v){
if(!is_array($v)){
$tmpArray[$k] = $v;
}
}

/**
* Change value of rest variables
*/
```

```
foreach($GLOBALS as $k => $v){
if($k != 'GLOBALS'
&& $k != '_POST'
&& $k != '_GET'
&& $k != '_COOKIE'
&& $k != '_FILES'
&& $k != $var
&& !is_array($v)
){
usleep(1);
$GLOBALS[$k] = md5(microtime());
}
}

$bool = $val != $GLOBALS[$var];

/**
* Restore defaults values
*/
foreach($tmpArray as $k => $v){
$GLOBALS[$k] = $v;
}

return $bool;
}
var_dump(is_reference('a'));
var_dump(is_reference('b'));
var_dump(is_reference('c'));
var_dump(is_reference('d'));
?>
```

This won't check if reference is inside a array.

up
down
-5

lutondatta at gmail dot com ¶
10 years ago

This is very useful in passing large arrays. If you do not pass the array by reference, you are creating another copy of the array in memory.

```
<?php
//Part 1
$x = 40;
$y = $x;
$z =& $x;
echo '$x is ' . $x . '<br />'; //Show 40
echo '$y is ' . $y . '<br />'; //Show 40
echo '$z is ' . $z . '<br />'; //Show 40
//Part 2
$x = 50;
echo '$x is ' . $x . '<br />'; //Show 50
echo '$y is ' . $y . '<br />'; //Show 40
echo '$z is ' . $z . '<br />'; //Show 50
?>
```

In part 2 value of $z will changes according to $x.

＋add a note

- Объяснение ссылок
  - Что такое ссылки
  - Что делают ссылки
  - Чем ссылки не являются
  - Передача по ссылке
  - Возврат по ссылке