Search

Getting Started
- Introduction
- A simple tutorial

Language Reference
- Basic syntax
- Types
- Variables
- Constants
- Expressions
- Operators
- Control Structures
- Functions
- Classes and Objects
- Namespaces
- Enumerations
- Errors
- Exceptions
- Fibers
- Generators
- Attributes
- References Explained
- Predefined Variables
- Predefined Exceptions
- Predefined Interfaces and Classes
- Predefined Attributes
- Context options and parameters
- Supported Protocols and Wrappers

Security
- Introduction
- General considerations
- Installed as CGI binary
- Installed as an Apache module
- Session Security
- Filesystem Security
- Database Security
- Error Reporting
- User Submitted Data
- Hiding PHP
- Keeping Current

Features
- HTTP authentication with PHP
- Cookies
- Sessions
- Dealing with XForms
- Handling file uploads
- Using remote files
- Connection handling
- Persistent Database Connections
- Command line usage

Garbage Collection
DTrace Dynamic Tracing

Function Reference
Affecting PHP's Behaviour
Audio Formats Manipulation
Authentication Services
Command Line Specific Extensions
Compression and Archive Extensions
Cryptography Extensions
Database Extensions
Date and Time Related Extensions
File System Related Extensions
Human Language and Character Encoding Support
Image Processing and Generation
Mail Related Extensions
Mathematical Extensions
Non-Text MIME Output
Process Control Extensions
Other Basic Extensions
Other Services
Search Engine Extensions
Server Specific Extensions
Session Extensions
Text Processing
Variable and Type Related Extensions
Web Services
Windows Only Extensions
XML Manipulation
GUI Extensions

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

Что такое ссылки »
« Объявление классов атрибутов

- Руководство по PHP
- Справочник языка

Change language: Russian

Submit a Pull Request Report a Bug

Объяснение ссылок

Содержание

- Что такое ссылки
- Что делают ссылки
- Чем ссылки не являются
- Передача по ссылке
- Возврат по ссылке
- Сброс переменных-ссылок
- Неявное использование механизма ссылок

User Contributed Notes 36 notes

80
Dave at SymmetricDesigns dot com ¶
15 years ago
Another example of something to watch out for when using references with arrays. It seems that even an ususued reference to an array cell modifies the *source* of the reference. Strange behavior for an assignment statement (is this why I've seen it written as an =& operator? - although this doesn't happen with regular variables).

```
<?php
$array1 = array(1,2);
$x = &$array1[1]; // Unused reference
$array2 = $array1; // reference now also applies to $array2 !
$array2[1]=22; // (changing [0] will not affect $array1)
print_r($array1);
?>
Produces:
Array
(
[0] => 1
[1] => 22 // var_dump() will show the & here
)
```

I fixed my bug by rewriting the code without references, but it can also be fixed with the unset() function:

```
<?php
$array1 = array(1,2);
$x = &$array1[1];
$array2 = $array1;
unset($x); // Array copy is now unaffected by above reference
$array2[1]=22;
print_r($array1);
?>
Produces:
Array
(
[0] => 1
[1] => 2
)
```


11
Carlos ¶
18 years ago
in the example below, you would get the same result if you change the function to something like:

```
function test_ref(&$arr) {
```

```
$time = time();
$size = sizeof($arr); // <--- this makes difference...
for($n=0; $n<$size; $n++) {
$x = 1;
}

echo "<br />The function using a reference took ".(time() - $time)." s";
}
```

up
down
32

ivan at mailinator dot com ¶

15 years ago

A little gotcha (be careful with references!):

```
<?php
$arr = array('a'=>'first', 'b'=>'second', 'c'=>'third');
foreach ($arr as &$a); // do nothing. maybe?
foreach ($arr as $a); // do nothing. maybe?
print_r($arr);
?>
```

Output:

```
Array
(
[a] => first
[b] => second
[c] => second
)
```

Add 'unset($a)' between the foreachs to obtain the 'correct' output:

```
Array
(
[a] => first
[b] => second
[c] => third
)
```

up
down
12

gnuffo1 at gmail dot com ¶

13 years ago

If you want to know the reference count of a particular variable, then here's a function that makes use of debug_zval_dump() to do so:

```
<?php
function refcount($var)
{
ob_start();
debug_zval_dump($var);
$dump = ob_get_clean();

$matches = array();
preg_match('/refcount\(([0-9]+)/', $dump, $matches);

$count = $matches[1];

//3 references are added, including when calling debug_zval_dump()
return $count - 3;
}
?>
```

debug_zval_dump() is a confusing function, as explained in its documentation, as among other things, it adds a reference count when being called as there is a reference within the function. refcount() takes account of these extra references by subtracting them for the return value.

It's also even more confusing when dealing with variables that have been assigned by reference (=&), either on the right or left side of the assignment, so for that reason, the above function doesn't really work for those sorts of variables. I'd use it more on object instances.

However, even taking into account that passing a variable to a function adds one to the reference count; which should mean that calling refcount() adds one, and then calling debug_zval_dump() adds another, refcount() seems to have aquired another reference from somewhere; hence subtracting 3 instead of 2 in the return line. Not quite sure where that comes from.

I've only tested this on 5.3; due to the nature of debug_zval_dump(), the results may be completely different on other versions.

up
down
10

alexander at gamerev dot org ¶

9 years ago

Simply put, here's an example of what referencing IS:

```
<?php
$foo = 5;
$bar = &$foo;
$bar++;

echo $foo;
?>
```

The above example will output the value 6, because $bar references the value of $foo, therefore, when changing $bar's value, you also change $foo's value too.

up
down
8

mpapec ¶

15 years ago

It's strange that function definition AND call to the same function must have "&" before them.

```
$arr = array();
$ref =& oras($arr['blah'], array());
$ref []= "via ref";
print_r($arr);

/* result
Array
(
[blah] => Array
(
[0] => via ref
)

)
*/

// perl like ||=
function &oras (&$v, $new) {
$v or $v = $new;
return $v;
}
```

up

down
19
midir ¶
14 years ago
Here is a good magazine article (PDF format) that explains the internals of PHP's reference mechanism in detail:
http://derickrethans.nl/files/phparch-php-variables-article.pdf

It should explain some of the odd behavior PHP sometimes seems to exhibit, as well as why you can't create "references to references" (unlike in C++), and why you should never attempt to use references to speed up passing of large strings or arrays (it will make no difference, or it will slow things down).

It was written for PHP 4 but it still applies. The only difference is in how PHP 5 handles objects: passing object variables by value only copies an internal pointer to the object. Objects in PHP 5 are only ever duplicated if you explicitly use the clone keyword.
up
down
13
nathan ¶
19 years ago
On the post that says php4 automagically makes references, this appears to *not* apply to objects:

http://www.php.net/manual/en/language.references.whatdo.php

"Note: Not using the & operator causes a copy of the object to be made. If you use $this in the class it will operate on the current instance of the class. The assignment without & will copy the instance (i.e. the object) and $this will operate on the copy, which is not always what is desired. Usually you want to have a single instance to work with, due to performance and memory consumption issues."
up
down
9
iryoku at terra dot es ¶
19 years ago
You should have in mind that php4 keep assigned variables "automagically" referenced until they are overwritten. So the variable copy is not executed on assignment, but on modification. Say you have this:

```
$var1 = 5;
$var2 = $var1; // In this point these two variables share the same memory location
$var1 = 3; // Here $var1 and $var2 have they own memory locations with values 3 and 5 respectively
```

Don't use references in function parameters to speed up aplications, because this is automatically done. I think that this should be in the manual, because it can lead to confusion.

More about this here:
http://www.zend.com/zend/art/ref-count.php
up
down
8
marco at greenlightsolutions dot nl ¶
16 years ago
I ran into a bit of a problem recently, with an array copy resulting in a reference copy of one of the elements instead of a clone. Sample code:

```
<?php
$a=array(1 => "A");
$b=&$a[1];
$c=$a; // should be a deep cloning
$c[1]="C";
var_dump($a[1]); // yields 'C' instead of 'A'
?>
```

After some searching, I found that it was a known bug which would be too costly to fix (see http://bugs.php.net/bug.php?id=20993). There was supposed to be some documentation on this behaviour on this page:

"Due to peculiarities of the internal workings of PHP, if a reference is made to a single element of an array and then the array is copied, whether by assignment or when passed by value in a function call, the reference is copied as part of the array. This means that changes to any such elements in either array will be duplicated in the other array (and in the other references), even if the arrays have different scopes (e.g. one is an argument inside a function and the other is global)! Elements that did not have references at the time of the copy, as well as references assigned to those other elements after the copy of the array, will behave normally (i.e. independent of the other array)."

However, this paragraph appears to have been removed from this page at some point, presumably because it was a bit obscure. The comments section seem to be a proper place for this, though.

up
down
12

dnhuff at acm dot org ¶

15 years ago

This is discussed before (below) but bears repeating:

$a = null; ($a =& null; does not parse) is NOT the same as unset($a);

$a = null; replaces the value at the destination of $a with the null value;

If you chose to use a convention like $NULL = NULL;

THEN, you could say $a =& $NULL to break any previous reference assignment to $a (setting it of course to $NULL), which could still get you into trouble if you forgot and then said $a = '5'. Now $NULL would be '5'.

Moral: use unset when it is called for.

up
down
5

hkmaly at bigfoot dot com ¶

19 years ago

It seems like PHP has problems with references, like that it can't work properly with circular references or free properly structure with more references. See http://bugs.php.net/?id=30053.

I have big problem with this and I hope someone from PHP add proper warning with explanation IN manual, if they can't fix it.

up
down
7

php at REMOVEMEkennel17 dot co dot uk ¶

18 years ago

I found a very useful summary of how references work in PHP4 (and some of the common pitfalls) in this article:
http://www.obdev.at/developers/articles/00002.html

It deals with some subtle situations and I recommend it to anyone having difficulty with their references.

up
down
8

Someone ¶

8 years ago

Another example of something to watch out for when using references with arrays. It seems that even an usused reference to an array cell modifies the *source* of the reference. Strange behavior for an assignment statement (is this why I've seen it written as an =& operator? - although this doesn't happen with regular variables).

```
<?php
$array1 = array(1,2);
$x = &$array1[1]; // Unused reference
$array2 = $array1; // reference now also applies to $array2 !
$array2[1]=22; // (changing [0] will not affect $array1)
print_r($array1);
?>
```

Produces:

```
Array
(
[0] => 1
[1] => 22 // var_dump() will show the & here
)

//above was Noted By Dave at SymmetricDesign dot com//
//and below is my opinion to this simple problem. //

This is an normal referencing problem.

when you gain an reference to a memory at some variable.
this variable, means "memory itself". (in above example, this would be -> $x = &$array1[1]; // Unused reference)

and you've copied original one($array1) to another one($array2).
and the copy means "paste everything on itself". including references or pointers, etcs. so, when you copied $array1 to $array2, this $array2 has same referencers that original $array1 has. meaning that $x = &$array1[1] = &$array2[1];
and again i said above. this reference means "memory itself".
when you choose to inserting some values to $array2[1],
$x; reference is affected by $array2[1]'s value. because, in allocated memory, $array2[1] is a copy of $array1[1]. this means that $array2[1] = $array1[1], also means &$array2[1] = &$array1[1] as said above. this causes memory's value reallocation on $array1[1]. at this moment. the problem of this topic is cleared by '$x', the memory itself. and this problem was solved by unsetting the '$x'. unsetting this reference triggers memory reallocation of $array2[1]. this closes the reference link between the copied one($array1, which is the original) and copy($array2). this is where that bug(clearly, it's not a bug. it's just a missunderstanding) has triggered by. closing reference link makes two array object to be separated on memory. and this work was done through the unset() function. this topic was posted 7 years ago, but i just want to clarify that it's not a bug.

if there's some problems in my notes, plz, note that on above.
```

up
down
5

sneskid at hotmail dot com ¶
11 years ago

```
There is no built in method (yet) to check if two variables are references to the same piece of data, but you can do a "reference sniff" test. This is rarely needed, but can be very useful. The function bellow is a slightly modified version of this technique I saw in a forum regarding this comparison limitation.

<?php
function is_ref_to(&$a, &$b)
{
$t = $a;
if($r=($b===($a=1))){ $r = ($b===($a=0)); }
$a = $t;
return $r;
}

$varA = 1;
$varB = $varA;
$varC =&$varA;

var_dump( is_ref_to($varA, $varB) ); // bool(false)
var_dump( is_ref_to($varA, $varC) ); // bool(true)
?>

The test above uses a two step process to be 100% generic.
But if you are sure the variables being tested will not be a certain value, example null, then use that value to allow a one step check.

<?php
function is_ref_to_1step(&$a, &$b)
{
```

```
$t = $a;
$r=($b===($a=null));
$a = $t;
return $r;
}
?>
```

up
down
10

sneskid at hotmail dot com ¶
17 years ago

in addition to what 'jw at jwscripts dot com' wrote about unset; it can also be used to "detach" the variable alias so that it may work on a unique piece of memory again.

here's an example

```
<?php
define('NL', "\r\n");

$v1 = 'shared';
$v2 = &$v1;
$v3 = &$v2;
$v4 = &$v3;

echo 'before:'.NL;
echo 'v1=' . $v1 . NL;
echo 'v2=' . $v2 . NL;
echo 'v3=' . $v3 . NL;
echo 'v4=' . $v4 . NL;

// detach messy
$detach = $v1;
unset($v1);
$v1 = $detach;

// detach pretty, but slower
eval(detach('$v2'));

$v1 .= '?';
$v2 .= ' no more';
$v3 .= ' sti';
$v4 .= 'll';

echo NL.'after:'.NL;
echo 'v1=' . $v1 . NL;
echo 'v2=' . $v2 . NL;
echo 'v3=' . $v3 . NL;
echo 'v4=' . $v4 . NL;

function detach($v) {
$e = '$detach = ' . $v . ';';
$e .= 'unset('.$v.');';
$e .= $v . ' = $detach;';
return $e;
}
?>

output {
before:
v1=shared
v2=shared
v3=shared
```

```
v4=shared

after:
v1=shared?
v2=shared no more
v3=shared still
v4=shared still
}
```

http://www.obdev.at/developers/articles/00002.html says there's no such thing as an "object reference" in PHP, but with detaching it becomes possible.

Hopefully detach, or something like it, will become a language construct in the future.

up
down
5

jw at jwscripts dot com ¶

19 years ago

Re-using variables which where references before, without unsetting them first, leads to unexpected behaviour.

The following code:

```
<?php

$numbers = array();

for ($i = 1; $i < 4; $i++) {
$numbers[] = null;
$num = count($numbers);
$index =& $numbers[$num ? $num - 1 : $num];
$index = $i;
}

foreach ($numbers as $index) {
print "$index\n";
}

?>
```

Does not produce:
1
2
3

But instead:
1
2
2

Applying unset($index) before re-using the variable fixes this and the expected list will be produced:
1
2
3

up
down
5

warnickr at gmail dot com ¶

16 years ago

I must say that it has been rather confusing following all of the explanations of PHP references, especially since I've worked a lot with C pointers. As far as I can tell PHP references are the same as C pointers for all practical purposes. I think a lot of the confusion comes from examples like the one shown below where people expect that a C pointer version of this would change what $bar references.

```
<?php
function foo(&$var)
{
$var =& $GLOBALS["baz"];
}
foo($bar);
?>
```

This is not the case. In fact, a C pointer version of this example (shown below) would behave exactly the same way (it would not modify what bar references) as the PHP reference version.

```
int baz = 5;
int* bar;
void foo(int* var)
{
var = &baz;
}
foo(bar);
```

In this case, just as in the case of PHP references, the call foo(bar) doesn't change what bar references. If you wanted to change what bar references, then you would need to work with a double pointer like so:

```
int baz = 5;
int* bar;
void foo(int** var)
{
*var = &baz;
}
foo(&bar);
```


5
trucex at gmail dot com ¶
16 years ago

In response to Xor and Slava:

I recommend you read up a bit more on the way PHP handles memory management. Take the following code for example:

```
<?php

$data = $_POST['lotsofdata'];
$data2 = $data;
$data3 = $data;
$data4 = $data;
$data5 = $data;

?>
```

Assuming we post 10MB of data to this PHP file, what will PHP do with the memory?

PHP uses a table of sorts that maps variable names to the data that variable refers to in memory. The $_POST superglobal will actually be the first instance of that data in the execution, so it will be the first variable referenced to that data in the memory. It will consume 10MB. Each $data var will simply point to the same data in memory. Until you change that data PHP will NOT duplicate it.

Passing a variable by value does just what I did with each $data var. There is no significant overhead to assigning a new name to the same data. It is only when you modify the data passed to the function that it must allocate memory for the data. Passing a variable by reference will do essentially the same thing when you pass the data to the function, only modifying it will modify the data that is in the memory already versus copying it to a new location in memory.

If for learning purposes you choose to disregard the obvious pointlessness in benchmarking the difference between these

two methods of passing arguments, you will need to modify the data when it is passed to the function in order to obtain more accurate results.

up
down
5

sneskid at hotmail dot com ¶

16 years ago

(v5.1.4)
One cool thing about var_dump is it shows which variables are references (when dumping arrays), symbolized by '∫' for int/null, and by '&' for boolean/double/string/array/object. I don't know why the difference in symmmmbolism.
After playing around I found a better way to implement detaching (twas by accident). var_dump can show what's going on.

```
<?php
function &detach($v=null){return $v;}

$A=array('x' => 123, 'y' => 321);
$A['x'] = &$A['x'];
var_dump($A);
/* x became it's own reference...
array(2) {
["x"]=> ∫(123)
["y"]=> int(321)
}*/

$A['y']=&$A['x'];
var_dump($A);
/* now both are references
array(2) {
["x"]=> ∫(123)
["y"]=> ∫(123)
}*/

$z = 'hi';
$A['y']=&detach(&$z);
var_dump($A);
/* x is still a reference, y and z share
array(2) {
["x"]=> ∫(123)
["y"]=> &string(2) "hi"
}*/

$A['x'] = $A['x'];
$A['y']=&detach();
var_dump($A,$z);
/* x returned to normal, y is on its own, z is still "hi"
array(2) {
["x"]=> int(123)
["y"]=> NULL
}*/
?>
```

For detach to work you need to use '&' in the function declaration, and every time you call it.

Use this when you know a variable is a reference, and you want to assign a new value without effecting other vars referencing that piece of memory. You can initialize it with a new constant value, or variable, or new reference all in once step.

up
down
3

gunter dot sammet at gmail dot com ¶

17 years ago

I tried to create an array with n depth using a recursive function passing array references around. So far I haven't had

much luck and I couldn't find anything on the web. So I ended up using eval() and it seems to work well:

```
<?php
foreach(array_keys($this->quantity_array) AS $key){
if($this->quantity_array[$key] > 0){
$combinations = explode('-', $key);
$eval_string = '$eval_array';
foreach(array_keys($combinations) AS $key2){
$option_key_value = explode('_', $combinations[$key2]);
$eval_string .= '['.$option_key_value[0].']['.$option_key_value[1].']';
}
$eval_string .= ' = '.$this->quantity_array[$key].';';
eval($eval_string);
}
}
?>
```

This produces an n dimensional array that will be available in the $eval_array variable. Hope it helps somebody!

up
down
4

cesoid at yahoo dot com ¶

18 years ago

Responding to post from nathan (who was responding to iryoku).

It is important to note the difference between what php is doing from the programmer's point of view and what it is doing internally. The note that nathan refers to, about how (for example) $something = $this makes a "copy" of the current object, is talking about making a "copy" from the programmer's perspective. That is, for the programmer, for all practical purposes, $something is a copy, even if internally nothing has been copied yet. For example, changing the data in member $something->somethingVar will not change your current object's data (i.e. it will not change $this->somethingVar).

What it does internally is a totally different story. I've tested "copying" an object which contains a 200,000 element array, it takes almost no time at all until you finally change something in one of the copies, because internally it only makes the copy when it becomes necessary. The original assignment takes less than a millisecond, but when I alter one of the copies, it takes something like a quarter of a second. But this only happens if I alter the 200,000 element array, if I alter a single integer of the object, it takes less than a microsecond again, so the interpretter seems to be smart enough to make copies of some of the objects variables and not others.

The result is that when you change a function to pass by reference, it will only become more efficient if, inside the function, the passed variable is having its data altered, in which case passing by reference causes your code to alter the data of the original copy. If you are passing an object and calling a function in that object, that function may alter the object without you even knowing, which means that you should pass an object by reference as long as it is ok for the original copy to be effected by what you do with the object inside the function.

I think the real moral of the story is this:
1) Pass by reference anything that should refer to and affect the original copy.
2) Pass not by reference things that will definitely not be altered in the function (for an object, it may be impossible to know whether it alters itself upon calling one of its functions).
3) If something needs to be altered inside a function without effecting the original copy, pass it not by reference, and pass the smallest practical part that needs to change, rather than passing, for example, a huge array of which one little integer will be altered.

Or a shorter version: Only pass things by reference when you need to refer to the original copy! (And don't pass huge arrays or long strings when you need to change just a small part of them!)

up
down
3

shooo dot xz at gmail dot com ¶

10 years ago

Hi, i've worked a abit on reference stuff.
Here is what i've noticed.

The problem: Sort mysql result through columns

```
$rewrite_self = gt::recombine(array('lang', 'id'), 'value_column', $sql_result);

public static function recombine($keys, $value, &$arr) {
$ref = array();
$main = &$ref;
foreach($arr as $data) {
foreach($keys as $key) {
if (!is_array($ref[$data[$key]])) $ref[$data[$key]] = array();
$ref = &$ref[$data[$key]];
}
$ref = $data[$value];
$ref = &$main;
}
return $main;
}

array(2) {
["pl"]=>
array(2) {
[2]=>
string(4) "value_column_str"
[3]=>
string(4) "value_column_str2"
}
["en"]=>
array(1) {
[13]=>
string(7) "value_column_str3"
}
}

Enjoy!
```
up
down
4
Ed ¶
18 years ago
Responding to Slava Kudinov. The only reason why your script takes longer when you pass by reference is that you do not at all modify the array that your passing to your functions. If you do that the diffrences in execution time will be a lot smaller. In fact passing by reference will be faster if just by a little bit.
up
down
5
Francis dot a at gmx dot net ¶
19 years ago
I don't know if this is a bug (I'm using PHP 5.01) but you should be careful when using references on arrays.
I had a for-loop that was incredibly slow and it took me some time to find out that most of the time was wasted with the function sizeof() at every loop, and even more time I spent finding out that this problem it must be somehow related to the fact, that I used a reference of the array. Take a look at the following example:

```
function test_ref(&$arr) {
$time = time();
for($n=0; $n<sizeof($arr); $n++) {
$x = 1;
}
echo "<br />The function using a reference took ".(time() - $time)." s";
}

function test_val($arr) {
$time = time();
for($n=0; $n<sizeof($arr); $n++) {
```

```
$x = 1;
}
echo "<br />The funktion using a value took: ".(time() - $time)." s";
}

// fill array
for($n=0; $n<2000; $n++) {
$ar[] = "test".$n;
}

test_ref($ar);
test_val($ar);
echo "<br />Done";
```

When I tested it, the first function was done after 9 seconds, while the second (although the array must be copied) was done in not even one.

The difference is inproportional smaller when the array size is reduced:
When using 1000 loops the first function was running for 1 second, when using 4000 it wasn't even done after 30 Seconds.

up
down
4

zoranbankovic at gmail dot com ¶

13 years ago

If someone wants to add slashes to multidimensional array directly, can use recursive (pass-by-reference) function like this:

```
<?php
function slashit(&$aray, $db_link)
{
foreach ($aray as $key => &$value)
if(is_array($value)) slashit($value, $link);
else $aray[$key] = mysql_real_escape_string($value, $db_link);
}

// Test:
$fruits = array (
"fruits" => array("a" => "or'ange", "b" => "ban'ana", "c" => "apple'"),
"numbers" => array(1, 2, 3, 4, 5, 6),
"holes" => array("fir'st", 5 => "sec'ond", "thir'd"),
"odma" => "jugo'slavija"
);

// You have to make link to the database or can use addslashes instead of mysql_real_escape_string and remove $link from function definition

slashit($fruits, $dbLink);
echo "<pre>"; print_r($fruits); echo "</pre>";
?>

// Output:
Array
(
[fruits] => Array
(
[a] => or\'ange
[b] => ban\'ana
[c] => apple\'
)

[numbers] => Array
(
[0] => 1
```

```
[1] => 2
[2] => 3
[3] => 4
[4] => 5
[5] => 6
)

[holes] => Array
(
[0] => fir\'st
[5] => sec\'ond
[6] => thir\'d
)

[odma] => jugo\'slavija
)
```

up
down
2

Youssef Omar ¶

14 years ago

This is to show the affect of changing property of object A through another object B when you pass object A as a property of another object B.

```
<?php
// data class to be passed to another class as an object
class A{
public $info;
function __construct(){
$this->info = "eeee";
}
}

// B class to change the info in A obj
class B_class{
public $A_obj;

function __construct($A_obj){
$this->A_obj = $A_obj;
}
public function change($newVal){
$this->A_obj->info = $newVal;
}
}

// create data object from the A
$A_obj = new A();
// print the info property
echo 'A_obj info: ' . $A_obj->info . '<br/>';

// create the B object and pass the A_obj we created above
$B_obj = new B_class($A_obj);
// print the info property through the B object to make sure it has the same value 'eeee'
echo 'B_obj info: ' . $B_obj->A_obj->info . '<br/>';

// chage the info property
$B_obj->change('xxxxx');
// print the info property through the B object to make sure it changed the value to 'xxxxxx'
echo 'B_obj info after change: ' . $B_obj->A_obj->info . '<br/>';
// print the info property from the A_obj to see if the change through B_obj has affected it
echo 'A_obj info: ' . $A_obj->info . '<br/>';
```

```
?>

The result:

A_obj info: eeee
B_obj info: eeee
B_obj info after change: xxxxx
A_obj info: xxxxx
```

up
down
2
grayson at uiuc dot edu ¶
18 years ago

I found a subtle feature of references that caused a bug in one of my PHP applications. In short, if an object passes one of its members to an external function that takes a reference as an argument, the external function can turn that member into a reference to an anonymous point in memory.

Why is this a problem? Later, when you copy the object with $a = $b, the copy and the original share memory.

Solution: If you want to have a function that uses references to modify a member of your object, your object should never pass the member to the function directly. It should first make a copy of the member. Then give that copy to the function. Then copy the new value of that copy in to your original object member.

Below is some code that can reporoduce the this feature and demonstrate the workaround.

```
function modify1 ( &$pointer_obj ){
$pointer_obj->property = 'Original Value';
}

function modify2 ( &$pointer_obj ){
$newObj->property = 'Original Value';
$pointer_obj = $newObj;
}

class a {
var $i; # an object with properties

function corrupt1(){
modify1 ($this->i);
}

function doNotCorrupt1(){
$tmpi = $this->i;
modify1 ($tmpi);
$this->i = $tmpi;
}

function corrupt2(){
modify2 ($this->i);
}

function doNotCorrupt2(){
$tmpi = $this->i;
modify2 ($tmpi);
$this->i = $tmpi;
}

}

$functions = array ('corrupt1', 'corrupt2', 'doNotCorrupt1', 'doNotCorrupt2');

foreach ($functions as $func){
```

```
$original = new a;

### Load some data in to the orginal with one of the four $functions
$original->$func();

$copy = $original;

$copy->i->property = "Changed after the copy was made.";

echo "\n{$func}: \$original->i->property = '" . $original->i->property . "'";
}

The script generates output:

corrupt1: $original->i->property = 'Changed after the copy was made.'
corrupt2: $original->i->property = 'Changed after the copy was made.'
doNotCorrupt1: $original->i->property = 'Original Value'
doNotCorrupt2: $original->i->property = 'Original Value'
```

up
down
2

jlaing at gmail dot com ¶

19 years ago

```
While trying to do object references with the special $this variable I found that this will not work:
class foo {
function bar() {
...
$this =& $some_other_foo_obj;
}
}

If you want to emulate this functionality you must iterate through the vars of the class and assign references like this:

$vars = get_class_vars('foo');
foreach (array_keys($vars) as $field) {
$this->$field =& $some_other_foo_obj->$field;
}

Now if you modify values within $this they will be modified within $some_other_foo_obj and vice versa.

Hope that helps some people!

p.s.
developer at sirspot dot com's note about object references doesn't seem correct to me.

$temp =& $object;
$object =& $temp->getNext();

Does the same exact thing as:

$object =& $object->getNext();

when you refernce $temp to $object all it does is make $temp an alias to the same memory as $object, so doing $temp-
>getNext(); and $object->getNext(); are calling the same function on the same object. Try it out if you don't believe me.
```

up
down
2

zzo38 ¶

15 years ago

```
You can make references like pointers. Example:
<?php
```

```
$a=6;
$b=array(&$a); // $b is a pointer to $a
$c=array(&$b); // $c is a pointer to $b
$d=7;
$c[0][0]=9; // $a is 9
$c[0]=array(&$d); // $b is a pointer to $d
$c[0][0]=4; // $d is 4
$b=array(&$a); // $b is a pointer to $a again
echo $a.$b[0].$c[0][0].$d; // outputs 9994
?>
```

These kind of pointers may even be passed to functions or returned from functions, copied and stored in multiple arrays/variables/objects, etc.

up
down
2

dallgoot ¶

5 years ago

After some headaches, here is a function to check if, between 2 variables $a,$b check if one is a reference to the other.
It means they "point" to the same value.
Tested on :
PHP 7.2.2 (cli) (built: Jan 31 2018 19:31:17) (ZTS MSVC15 (Visual C++ 2017) x64)
Hope that helps...

```
<?php

function are_references(&$a, &$b){
$mem = $a; //memorize
$a = uniqid ("REFERENCE???", true ); //change $a
$same = $a === $b; //compare
$a = $mem; //restore $a
return $same;
}
echo "***distinct vars AND distinct values\n";
$a = "toto";
$b = "tata";

var_dump($a, $b, are_references($a, $b));
echo "verify original values: $a, $b\n";

echo "***distinct vars BUT SAME values\n";
$a = "toto";
$b = "toto";

var_dump($a, $b, are_references($a, $b));
echo "verify original values: $a, $b\n";

echo '*** $b is a reference of $a'."\n";
$a = "titi";
$b = &$a;

var_dump($a, $b, are_references($a, $b));
echo "verify original values: $a, $b\n";

echo '*** $a is a reference of $b'."\n";
$b = "titi";
$a = &$b;

var_dump($a, $b, are_references($a, $b));
echo "verify original values: $a, $b\n";
?>
```

Result:

```
***distinct vars AND distinct values
string(4) "toto"
```

```
string(4) "tata"
bool(false)
verify original values: toto, tata
***distinct vars BUT SAME values
string(4) "toto"
string(4) "toto"
bool(false)
verify original values: toto, toto
*** $b is a reference of $a
string(4) "titi"
string(4) "titi"
bool(true)
verify original values: titi, titi
*** $a is a reference of $b
string(4) "titi"
string(4) "titi"
bool(true)
verify original values: titi, titi
```

up
down
1
mramirez (at) star (minus) dev (dot) com ¶
18 years ago

```
For php programmers that come from pascal,
in object pascal (delphi),
variable references are used with the "absolute" keyword.

PHP example:

<?php

global $myglobal;

$myglobal = 5;

function test()
{
global $myglobal;

/*local*/ $mylocal =& $myglobal;

echo "local: " . $mylocal . "\n";
echo "gloal: " . $myglobal . "\n";
}

test();

?>

Pascal example:

program dontcare;

var myglobal: integer;

procedure test;
var mylocal ABSOLUTE myglobal;
begin
write("local: ", mylocal);
write("global: ", myglobal);
end;
```

```
begin
myglobal := 5;
test;
end.

By the way, a "local" keyword in php for local variables,
could be welcome :-)
```

up
down
1
nslater at gmail dot com ¶
18 years ago

In addition to the note made by "Francis dot a at gmx dot net" you should not normally be using a function such as sizeof() or count() in a control structure such as FOR because the same value is being calculated repeatedly for each iteration. This can slow things down immensely, regardless of whether you pass by value or reference.

It is generally much better to calculate the static values before the defining the looping control structure.

Example:

```
<?php

$intSize = sizeof($arrData);

for($i = 0; $i < $intSize; $n++) {
// Do stuff
}

?>
```

up
down
0
jasonpvp at gmail dot com ¶
15 years ago

To change values in a multi-dimensional array while looping through:

```
$var=array('a'=>array(1,2,3),'b'=>array(4,5,6));

foreach ($var as &$sub) {
foreach ($sub as &$element) {
$element=$element+1;
}
}

var_dump($var);

------------------------------
produces:
------------------------------
array(2) {
["a"]=>
array(3) {
[0]=>
int(2)
[1]=>
int(3)
[2]=>
int(4)
}
["b"]=>
array(3) {
[0]=>
```

```
int(5)
[1]=>
int(6)
[2]=>
int(7)
}
}
```

up
down
-2

jszoja at gmail dot com ¶

7 years ago

```
Be aware of a reference to another reference. It is probably bad practice to even do that.

<?php
class Obj1 { public $name = 'Obj1'; }
class Obj2 { public $name = 'Obj2'; }

$objects = [];
$toLoad = [ 'Obj1', 'Obj2' ];

foreach( $toLoad as $i => $obj )
{
$ref = new $obj();
$objects[$i] =& $ref; // a reference to a reference
// $objects[$i] = $ref; // that would work
}

echo $objects[0]->name;
// outputs 'Obj2' !
?>
```

up
down
-1

info at fedushin dot ru ¶

2 years ago

```
Arrays of references are supported:

<?php
$x = 1;
$y = 2;
$z = 3;

$arr = [&$x, &$y, &$z];

foreach($arr as &$item)
$item = 10;

var_dump($x, $y, $z);
?>

Output:

int(10) int(10) int(10)
```

＋add a note

- Операторы
- Управляющие конструкции
- Функции
- Классы и объекты
- Пространства имён
- Перечисления
- Ошибки
- Исключения
- Fibers
- Генераторы
- Атрибуты
- Объяснение ссылок
- Предопределённые переменные
- Предопределённые исключения
- Встроенные интерфейсы и классы
- Предопределённые атрибуты
- Контекстные опции и параметры
- Поддерживаемые протоколы и обёртки