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

Передача по ссылке »
« Что делают ссылки

- Руководство по PHP
- Справочник языка
- Объяснение ссылок

Change language: Russian

Чем ссылки не являются

Как уже было сказано, ссылки не являются указателями. Это означает, что следующая конструкция не будет делать то, что вы ожидаете:

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

Переменная *$var* в функции *foo* будет связана с *$bar* в вызывающем коде, но затем она будет перепривязана к *$GLOBALS["baz"]*. Нет способа связать *$bar* в области видимости вызывающем коде с чем-либо ещё путём использования механизма ссылок, поскольку *$bar* не доступна в функции *foo* (доступно лишь её значение через *$var*, но *$var* имеет только значение переменной и не имеет связи имя-значение в таблице имён переменных). Вы можете воспользоваться возвратом ссылок из функции для привязки внешней переменной к другому значению.

+ add a note

User Contributed Notes 13 notes

up
down
169
Andrew ¶
15 years ago

```
What References are not: References.

References are opaque things that are like pointers, except A) smarter and B) referring to HLL objects, rather than memory addresses. PHP doesn't have references. PHP has a syntax for creating *aliases* which are multiple names for the same object. PHP has a few pieces of syntax for calling and returning "by reference", which really just means inhibiting copying. At no point in this "references" section of the manual are there any references.
```

up
down
76
Anonymous ¶
15 years ago

```
The example given in the text:

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

illustrates (to my mind anyway) why the = and & should be written together as a new kind of replacement operator and not apart as in C, like $var = &$GLOBALS["baz"];

Using totally new terminology:

To me the result of this function is not surprising because the =& means 'change the "destination" of $var from wherever it was to the same as the destination of $GLOBALS["baz"]. Well it 'was' the actual parameter $bar, but now it will be the global at "baz".

If you simply remove the & in the the replacement, it will place the value of $GLOBALS["baz'] into the destination of $var, which is $bar (unless $bar was already a reference, then the value goes into that destination.)
```

To summarize, =, replaces the 'destination's value; =&, changes the destination.

up
down
2
bravo1romeo ¶
6 years ago

You can reference values of an array to the values of another array,
however if you change the array by reassigning it, the reference will no longer apply, for example:

```
<?php
$ref = [1,2,3];
$c = count($ref);
$foo = ['A'];

for($i=0;$i<$c;$i++)
$foo[] =& $ref[$i];

print_r($foo);
print_r($ref);

$ref = [4,5,6];

print_r($foo);
print_r($ref);
?>
```

Will output:

```
Array
(
[0] => A
[1] => 1
[2] => 2
[3] => 3
)
Array
(
[0] => 1
[1] => 2
[2] => 3
)
Array
(
[0] => A
[1] => 1
[2] => 2
[3] => 3
)
Array
(
[0] => 4
[1] => 5
[2] => 6
)
```

Therefore if you want the values to still reference you must set the array values individually to not reassign the array:

```
<?php
$ref = [1,2,3];
$c = count($ref);
$foo = ['A'];

for($i=0;$i<$c;$i++)
$foo[] =& $ref[$i];
```

```
print_r($foo);
print_r($ref);

$bar = [4,5,6];
foreach($bar as $i => $value)
$ref[$i] = $value;

print_r($foo);
print_r($ref);
?>

Results:
Array
(
[0] => A
[1] => 1
[2] => 2
[3] => 3
)
Array
(
[0] => 1
[1] => 2
[2] => 3
)
Array
(
[0] => A
[1] => 4
[2] => 5
[3] => 6
)
Array
(
[0] => 4
[1] => 5
[2] => 6
)
```

up
down
8
ansonyumo at email dot com ¶
19 years ago

The assertion, "references are not like pointers," is a bit confusing.

In the example, the author shows how assigning a reference to a formal parameter that is also a reference does not affect the value of the actual parameter. This is exactly how pointers behave in C. The only difference is that, in PHP, you don't have to dereference the pointer to get at the value.

```
-+-+-
int bar = 99;

void foo(int* a)
{
a = &bar;
}

int main()
{
int baz = 1;
foo(&baz);
printf("%d\n", baz);
return 0;
}
-+-+-
```

The output will be 1, because foo does not assign a value to the dereferenced formal parameter. Instead, it reassigns the formal parameter within foo's scope.

Alternatively,

```
-+-+-
int bar = 99;

void foo(int* a)
{
*a = bar;
}

int main()
{
int baz = 1;
foo(&baz);
printf("%d\n", baz);
return 0;
}
-+-+-
```

The output will be 9, because foo dereferenced the formal parameter before assignment.

So, while there are differences in syntax, PHP references really are very much like pointers in C.

I would agree that PHP references are very different from Java references, as Java does not have any mechanism to assign a value to a reference in such a way that it modifies the actual parameter's value.

up
down
1
leonardp122794 at gmail dot com ¶
6 years ago

I kinda agree with the user above who said that php references were actually quite a bit more like c pointers than JAVA or C++.

And here's why.

They act essentially exactly like pointers.

The difference between unset($x) and free(x) seem to be differences between the unset() and free() operators whether than differences between references and pointers themselves.

Free deallocates the memory on the heap.

Unset simple removes the variable.

Other than that, we're dealing with two very simple constructs here.

up
down
4
briank at kappacs dot com ¶
13 years ago

I think the terminology is tripping people up when it comes to assigning objects.

Try thinking of binding and references like this:

```
<?php
```

```
# Code:
$a = 5; $b =& $a; $c = new stdClass(); $d = $c;

# Behind the scenes symbol table and values:
$global_names = array(
'a' => array('binding' => 0),
'b' => array('binding' => 0),
'c' => array('binding' => 1),
'd' => array('binding' => 2),
);
$values = array(
0 => array('type' => 'scalar', 'value' => 5),
1 => array('type' => 'objId', 'value' => 0),
2 => array('type' => 'objId', 'value' => 0)
);
?>
```

$a is bound to (or references, or is a reference to) the value at index 0 (scalar 5).
$b is bound to the same thing as $a--the value at index 0 (scalar 5).
$c is bound to the value at index 1 (object ID 0).
$d is bound to the value at index 2 (a separate and distinct value also referring to object ID 0).

When the documentation states that you cannot [re-]bind $bar to something else from within the example function foo, it means you can't change what in my pseudo-engine would be $global_names['bar']['binding']. You can only change $values[$names['var']['binding']] (using "$var ="; the same value referenced/bound by $values[$global_names['bar']['binding']]) or $names['var']['binding'] (using "$var =&").

Also consider this code:

```
<?php
$a = 3; $b =& $a;
function foo (&$c) { $c = new stdClass(); }
function bar () { return new stdClass(); }
function &fum () { return new stdClass(); }
if (!is_object($a)) { echo "\$a does not initially refer to an object\n"; }
foo($b);
echo "\$b ", ($a === $b)? "has not": "has", " been re-bound by foo\n";
if (is_object($a)) { echo "\$a now contains an object identifier\n"; }
$b =& bar();
echo "\$b ", ($a === $b)? "has not": "has", " been re-bound by bar\n";
$b =& fum();
echo "\$b ", ($a === $b)? "has not": "has", " been re-bound by fum\n";
?>
```

which outputs:

```
$a does not initially refer to an object
$b has not been re-bound by foo
$a now contains an object identifier
$b has not been re-bound by bar
$b has been re-bound by fum
```

In other words, the value can be changed but the binding does not (except for returning a reference), exactly as stated.

Object identifiers do make object "values" work like pointers (but not to the extent of C/C++, and not like references).
up
down
4
christian at kno dot at ¶
22 years ago
As said above references are not pointers.

Following example shows a difference between pointers and references.

This Code

```
<?
$b = 1;
$a =& $b;

print("<pre>");
print("\$a === \$b: ".(($a === $b) ? "ok" : "failed")."\n");
print("unsetting \$a...\n");
unset($a);
print("now \$a is ".(isset($a) ? "set" : "unset")." and \$b is ".(isset($b) ? "set" : "unset")."\n");
print("</pre>");

$b = 1;
$a =& $b;

print("<pre>");
print("\$a === \$b: ".(($a === $b) ? "ok" : "failed")."\n");
print("unsetting \$b...\n");
unset($b);
print("now \$a is ".(isset($a) ? "set" : "unset")." and \$b is ".(isset($b) ? "set" : "unset")."\n");
print("</pre>");
?>
```

will produce this output:

```
---------
$a === $b: ok
unsetting $a...
now $a is unset and $b is set

$a === $b: ok
unsetting $b...
now $a is set and $b is unset
---------
```

So you see that $a and $b are identical ($a === $b -> true), but if one of both is unset, the other is not effected.

up

down

7

shuimuqingshu at gmail dot com ¶

11 years ago

i think the code below can tell the difference between PHP reference and C pointer:

In PHP:

```
<?php
$a = 0;
$b = &a;
echo $a; //0
unset($b); // unset $b
echo $a; //0 it's OK
?>
```

In C:

```
#include <stdio.h>
int main(int argc, char const *argv[]) {
int a = 0;
int* b = &a;

printf("%i\n", a); //0
free(b); // free b
printf("%i\n", a); //get error: *** error for object 0x7fff6350da08: pointer being freed was not allocated
}
```
up
down
1
Anonymous ¶
13 years ago

```
I understand this like that:
  The reference in PHP is like creating single pointer at own variable in C/C++ and point at variable ( without pointers arithmetic and we can't get number of variable address in memory).

For example
<?php
$a = 4;
$b = &$a;
$c = &$b;
echo "$a - $b - $c<br>";
// 3 pointers ( a , b , c) point at memory location where stored value of number is 4.
$c = 5;
echo "$a - $b - $c<br>";
// all variables equals 5;
unset($a);
$c = 6;
echo "$a - $b - $c<br>";
//$a is not exist but it was only pointer ( not real part of memory) so we have to way to get value or change it
?>
----
When we want create some "pointer of pointer" in PHP i can't do that because it's impossible in PHP. We need pointer to another pointer to change the place that the pointer refers to. In your exaple you just change value of variable in function. ( no operation of pointers )
```
up
down
-2
minhtrung2606 at gmail dot com ¶
4 years ago

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

Let's re-write the above snippet of code in another way

<?php
$bar = &$var; // At this time, $bar refers to the content to which $var refers which is NULL or empty
$var = & $GLOBALS["baz"]; // At this time, $var changes to refer to another content to which global $baz variable refers. However $bar is still refers to the content of NULL

Expected Result: $bar will hold the content of $var (which is now the content of $baz)
Actual Result: $bar holds a content of NULL or empty. This is what's PHP References does
```
up
down
0
Anonymous ¶
8 years ago

```
It is possible to do something a bit similar to pointers (like in C), where something like array(&$a) is a pointer to a variable called $a; this value can then be passed around as a value; it is not a variable or an alias or whatever but is an actual value.

You can then use codes such as these to read/write through pointers:
<?php
```

```
function get($x) { return $x[0]; }
function put($x,$y) { $x[0]=$y; }
?>
```
up
down
0
***ArticIce(Juice)* ¶**
8 years ago
```
Consider this block of code:

<?php
$arr = ['1', '2', '3', '4'];

foreach ($arr as &$i) {}
echo implode($arr, ', ')."\n";

foreach ($arr as $i) {}
echo implode($arr, ', ')."\n";
?>

which will output
1, 2, 3, 4
1, 2, 3, 3

although it seems no changes were made to the array.

The last item in the array gets overwritten, because reference is replaced by a copy in the second iteration. In more
detail, it gets overwritten first by a 1, then by 2, by 3 and again by a 3.

Make sure to do an unset($i) when you run an iteration by copy after an iteration by reference using same variable names!
```
up
down
-2
***schultz   at   widescreen   dot   ch* ¶**
20 years ago
```
A not so simple Workaround...but still doable...have fun

class My{
var $value;

function get1(&$ref){
$ref[] =& $this;
}

function get2(&$ref){
$ref =& $this;
}

function get3(&$ref){
$ref = $this;
}
}

$m = new My();

$m->value = 'foo';
$m->get1($ref=array());
$m1 =& $ref[0];
$m1->value = 'bar';
echo "\n".'Works but is ugly...';
echo "\n".' m:'. get_class($m) . '->value = '. $m->value;
echo "\n".' m1:'. get_class($m1) . '->value = '. $m1->value;
echo "\n".'Does not work because references are not pointers...';
$m->value = 'foo';
$m->get2($m2);
$m2->value = 'bar';
echo "\n".' m:'. get_class($m) . '->value = '. $m->value;
echo "\n".' m2:'. get_class($m2) . '->value = '. $m2->value;

$m->value = 'foo';
$m->get3($m3);
$m3->value = 'bar';
echo "\n".'Does not work becuase it is set to a copy';
echo "\n".' m:'. get_class($m) . '->value = '.$m->value;
echo "\n".' m3:'. get_class($m3) . '->value = '. $m3->value;
```
+ add a note