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

Что делают ссылки »
« Объяснение ссылок

- Руководство по PHP
- Справочник языка
- Объяснение ссылок

Change language: Russian

Что такое ссылки

Ссылки в PHP - это средство доступа к содержимому одной переменной под разными именами. Они не похожи на указатели C; например, вы не можете выполнять над ними адресную арифметику, они не являются реальными адресами в памяти и т.д. Для получения дополнительной информации смотрите Чем ссылки не являются. Вместо этого указатели в PHP - это псевдонимы в таблице имён переменных. В PHP имя переменной и её содержимое - это разные вещи, поэтому одно содержимое может иметь разные имена. Можно провести аналогию с именами файлов и файлами в Unix: имена переменных - записи в каталоге, а содержимое переменной - это сам файл. Ссылки в PHP - аналог жёстких ссылок в файловых системах Unix.

User Contributed Notes 4 notes

193
273118949 at qq dot com ¶
6 years ago
```
it just likes a person who has two different names.
```

40
Anonymous ¶
5 years ago
```
Unlike in C, PHP references are not treated as pre-dereferenced pointers, but as complete aliases.

The data that they are aliasing ("referencing") will not become available for garbage collection until all references to
it have been removed.

"Regular" variables are themselves considered references, and are not treated differently from variables assigned using =&
for the purposes of garbage collection.

The following examples are provided for clarification.

1) When treated as a variable containing a value, references behave as expected. However, they are in fact objects that
*reference* the original data.

<?php
var = "foo";
$ref1 =& $var; // new object that references $var
$ref2 =& $ref1; // references $var directly, not $ref1!!!!!

echo $ref; // >foo

unset($ref);

echo $ref1; // >Notice: Undefined variable: ref1
echo $ref2; // >foo
echo $var; // >foo
?>

2) When accessed via reference, the original data will not be removed until *all* references to it have been removed. This
includes both references and "regular" variables assigned without the & operator, and there are no distinctions made
between the two for the purpose of garbage collection.

<?php
$var = "foo";
$ref =& $var;
```

```
unset($var);

echo $var; // >Notice: Undefined variable: var
echo $ref; // >foo
?>

3) To remove the original data without removing all references to it, simply set it to null.

<?php
$var = "foo";
$ref =& $var;

$ref = NULL;

echo $var; // Value is NULL, so nothing prints
echo $ref; // Value is NULL, so nothing prints
?>

4) Placing data in an array also counts as adding one more reference to it, for the purposes of garbage collection.

For more info, see http://php.net/manual/en/features.gc.refcounting-basics.php
```

up
down
15

aldo dot caruso at argencasas dot com ¶

4 years ago

```
The following three code snippets show the effect of using references in scalar variables, arrays and objects under
different circumstances.

In any case the result is the expected one if you stick to the concept that a reference is an alias to a variable. After
assigning by reference ( no matter if $a =& $b or $b =& $a ) both variable names refer to the same variable.

References with scalars

<?php
/*
References are aliases for the same variable
*/

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

$b = 2;
echo "$a,$b\n"; // 2,2

$a = 3;
echo "$a,$b\n"; // 3,3

// Variables can be bound before being assigned
$c =& $d;
$c = 4;
echo "$c,$d\n"; // 4,4
?>

References with arrays

<?php
/*
Array elements referencing scalar variables
*/

$a = 1;
```

```
$b = 2;
$c = array(&$a, &$b);

$a = 3;
$b = 4;
echo "c: $c[0],$c[1]\n"; // 3,4

$c[0] = 5;
$c[1] = 6;
echo "a,b: $a,$b\n"; // 5,6

/*
Reference between arrays
*/

$d = array(1,2);
$e =& $d;

$d[0] = 3;
$d[1] = 4;
echo "e: $e[0],$e[1]\n"; // 3,4

$e[0] = 5;
$e[1] = 6;
echo "d: $d[0],$d[1]\n"; // 5,6

$e = 7;
echo "d: $d\n"; // 7 ( $d is no more an array, but an integer )

/*
Iterating an array of references using foreach construct
*/

$a = 1;
$b = 2;
$f = array(&$a,&$b);

foreach($f as $x) // If $x is assigned by value it doesn't change referred variables.
$x = 3;
echo "a,b: $a,$b\n"; // 1,2

foreach($f as &$x) // If $x is assigned by reference it changes referred variables.
$x = 3;
echo "a,b: $a,$b\n"; // 3,3

// Be aware that, after the loop, $x still references $f[1] and so $b
$x = 4;
echo "a,b: $a,$b\n"; // 3,4 ( $b affected )

// To avoid previous side effects it is advisable to unset x, unlinking it from $f[1] and $b
unset($x);
$x = 5;
echo "a,b: $a,$b\n"; // 3,4 ( $b not affected )
?>
```

References with objects

```
<?php
/*
Object property referencing a scalar variable
*/
```

```
$a = 1;
$b = new stdClass();
$b->x =& $a;

$a = 2;
echo "b->x: $b->x\n"; // 2

$b->x = 3;
echo "a: $a\n"; // 3

/* Reference between objects */
$c = new stdClass();
$c->x = 1;
$d =& $c;

$d->x = 2;
echo "c->x: $c->x\n"; // 2

$d = new stdClass();
$d->y = 3;
echo "c->y: $c->y\n"; // 3
echo "c->x: $c->x\n"; // Undefined property: stdClass::$x
?>
```

up
down
-12
anon ¶
7 years ago

In summary, "&$reference" means "do-not-copy-on-write the value here, in perpetuity". Assigning by reference is not assignment, it's "make &$variable a reference and its value do-not-copy-on-write, in perpetuity, and make the variable I'm assigning to use that do-not-copy-on-write value as well".

To "unreference/unalias" you have to either unset or make an explicit copy into a new variable.

Object properties that are references will survive cloning and remain references. Generally the same is true with references in arrays and PHP's array functions (combine, intersect, call_user_func, func_get_args, etc).

Calling a function that uses a reference parameter will *make* the supplied variable a reference. This is also true when using variadic array expansion for arguments; the supplier's array element will become a reference.

Generally, don't use them unless you're dealing with low-level calls, or need an accumulator, etc. For poorly designed functions that use them, give them a copy to mangle.

＋add a note