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

Сравнение объектов »
« Ключевое слово final

- Руководство по PHP
- Справочник языка
- Классы и объекты

Change language: Russian

Клонирование объектов

Создание копии объекта с абсолютно идентичными свойствами не всегда является приемлемым вариантом. Хорошим примером необходимости копирования конструкторов может послужить ситуация, когда у вас есть объект, представляющий собой окно GTK и содержащий ресурс-идентификатор этого окна; когда вы создаёте копию этого объекта, вам может понадобиться, чтобы копия объекта содержала ресурс-идентификатор нового окна. Другим примером может послужить ситуация, когда ваш объект содержит ссылку на какой-либо другой используемый объект и, когда вы создаёте копию родительского объекта, вам нужно также создать новый экземпляр этого другого объекта, так, чтобы копия объекта-контейнера содержала собственный отдельный экземпляр содержащегося объекта.

Копия объекта создаётся с использованием ключевого слова `clone` (который вызывает метод __clone() объекта, если это возможно).

```
$copy_of_object = clone $object;
```

При клонировании объекта, PHP выполняет поверхностную копию всех свойств объекта. Любые свойства, являющиеся ссылками на другие переменные, останутся ссылками.

__clone(): void

После завершения клонирования, если метод __clone() определён, то будет вызван метод __clone() вновь созданного объекта для возможного изменения всех необходимых свойств.

Пример #1 Клонирование объекта

```
<?php
class SubObject
{
static $instances = 0;
public $instance;

public function __construct() {
$this->instance = ++self::$instances;
}

public function __clone() {
$this->instance = ++self::$instances;
}
}

class MyCloneable
{
public $object1;
public $object2;

function __clone()
{
// Принудительно клонируем this->object1, иначе
// он будет указывать на один и тот же объект.
$this->object1 = clone $this->object1;
}
}

$obj = new MyCloneable();

$obj->object1 = new SubObject();
$obj->object2 = new SubObject();

$obj2 = clone $obj;


print "Оригинальный объект:\n";
```

```php
print_r($obj);

print "Клонированный объект:\n";
print_r($obj2);

?>
```

Результат выполнения приведённого примера:

```
Оригинальный объект:
MyCloneable Object
(
    [object1] => SubObject Object
        (
            [instance] => 1
        )

    [object2] => SubObject Object
        (
            [instance] => 2
        )

)
Клонированный объект:
MyCloneable Object
(
    [object1] => SubObject Object
        (
            [instance] => 3
        )

    [object2] => SubObject Object
        (
            [instance] => 2
        )

)
```

Возможно обращаться к свойствам/методам только что склонированного объекта:

Пример #2 Доступ к только что склонированному объекту

```php
<?php
$dateTime = new DateTime();
echo (clone $dateTime)->format('Y');
?>
```

Вывод приведённого примера будет похож на:

```
2016
```


User Contributed Notes 14 notes

69
jorge dot villalobos at gmail dot com ¶
18 years ago
I think it's relevant to note that __clone is NOT an override. As the example shows, the normal cloning process always occurs, and it's the responsibility of the __clone method to "mend" any "wrong" action performed by it.

45
jojor at gmx dot net ¶
13 years ago
Here is test script i wrote to test the behaviour of clone when i have arrays with primitive values in my class - as an additonal test of the note below by jeffrey at whinger dot nl

```
<pre>
```

```
<?php

class MyClass {

private $myArray = array();
function pushSomethingToArray($var) {
array_push($this->myArray, $var);
}
function getArray() {
return $this->myArray;
}

}

//push some values to the myArray of Mainclass
$myObj = new MyClass();
$myObj->pushSomethingToArray('blue');
$myObj->pushSomethingToArray('orange');
$myObjClone = clone $myObj;
$myObj->pushSomethingToArray('pink');

//testing
print_r($myObj->getArray()); //Array([0] => blue,[1] => orange,[2] => pink)
print_r($myObjClone->getArray());//Array([0] => blue,[1] => orange)
//so array cloned

?>
</pre>
```

up
down
28

MakariVerslund at gmail dot com ¶

17 years ago

I ran into the same problem of an array of objects inside of an object that I wanted to clone all pointing to the same objects. However, I agreed that serializing the data was not the answer. It was relatively simple, really:

```
public function __clone() {
foreach ($this->varName as &$a) {
foreach ($a as &$b) {
$b = clone $b;
}
}
}
```

Note, that I was working with a multi-dimensional array and I was not using the Key=>Value pair system, but basically, the point is that if you use foreach, you need to specify that the copied data is to be accessed by reference.

up
down
13

emile at webflow dot nl ¶

13 years ago

Another gotcha I encountered: like __construct and __desctruct, you must call parent::__clone() yourself from inside a child's __clone() function. The manual kind of got me on the wrong foot here: "An object's __clone() method cannot be called directly."

up
down
5

tolgakaragol at gmail dot com ¶

4 years ago

Here is a basic example about clone issue. If we use clone in getClassB method. Return value will be same as new B() result. But it we dont use clone we can effect B::$varA.

```
class A
{
protected $classB;

public function __construct(){
$this->classB = new B();
}

public function getClassB()
{
return clone $this->classB;
}
}

class B
{
protected $varA = 2;

public function getVarA()
{
return $this->varA;
}

public function setVarA()
{
$this->varA = 3;
}
}

$a = new A();

$classB = $a->getClassB();

$classB->setVarA();

echo $a->getClassB()->getVarA() . PHP_EOL;// with clone -> 2, without clone it returns -> 3

echo $classB->getVarA() . PHP_EOL; // returns always 3
```

up
down
12

ben at last dot fm ¶

14 years ago

Here are some cloning and reference gotchas we came up against at Last.fm.

1. PHP treats variables as either 'values types' or 'reference types', where the difference is supposed to be transparent. Object cloning is one of the few times when it can make a big difference. I know of no programmatic way to tell if a variable is intrinsically a value or reference type. There IS however a non-programmatic ways to tell if an object property is value or reference type:

```
<?php

class A { var $p; }

$a = new A;
$a->p = 'Hello'; // $a->p is a value type
var_dump($a);

/*
object(A)#1 (1) {
["p"]=>
string(5) "Hello" // <-- no &
```

```
}
*/

$ref =& $a->p; // note that this CONVERTS $a->p into a reference type!!
var_dump($a);

/*
object(A)#1 (1) {
["p"]=>
&string(5) "Hello" // <-- note the &, this indicates it's a reference.
}
*/

?>
```

2. unsetting all-but-one of the references will convert the remaining reference back into a value. Continuing from the previous example:

```
<?php

unset($ref);
var_dump($a);

/*
object(A)#1 (1) {
["p"]=>
string(5) "Hello"
}
*/

?>
```

I interpret this as the reference-count jumping from 2 straight to 0. However...

2. It IS possible to create a reference with a reference count of 1 - i.e. to convert an property from value type to reference type, without any extra references. All you have to do is declare that it refers to itself. This is HIGHLY idiosyncratic, but nevertheless it works. This leads to the observation that although the manual states that 'Any properties that are references to other variables, will remain references,' this is not strictly true. Any variables that are references, even to *themselves* (not necessarily to other variables), will be copied by reference rather than by value.

Here's an example to demonstrate:

```
<?php

class ByVal
{
var $prop;
}

class ByRef
{
var $prop;
function __construct() { $this->prop =& $this->prop; }
}

$a = new ByVal;
$a->prop = 1;
$b = clone $a;
$b->prop = 2; // $a->prop remains at 1

$a = new ByRef;
```

```
$a->prop = 1;
$b = clone $a;
$b->prop = 2; // $a->prop is now 2

?>
```

up
down
11

Hayley Watson ¶

16 years ago

It should go without saying that if you have circular references, where a property of object A refers to object B while a property of B refers to A (or more indirect loops than that), then you'll be glad that clone does NOT automatically make a deep copy!

```
<?php

class Foo
{
var $that;

function __clone()
{
$this->that = clone $this->that;
}

}

$a = new Foo;
$b = new Foo;
$a->that = $b;
$b->that = $a;

$c = clone $a;
echo 'What happened?';
var_dump($c);
```

up
down
3

stanislav dot eckert at vizson dot de ¶

9 years ago

This base class automatically clones attributes of type object or array values of type object recursively. Just inherit your own classes from this base class.

```
<?php
class clone_base
{
public function __clone()
{
$object_vars = get_object_vars($this);

foreach ($object_vars as $attr_name => $attr_value)
{
if (is_object($this->$attr_name))
{
$this->$attr_name = clone $this->$attr_name;
}
else if (is_array($this->$attr_name))
{
// Note: This copies only one dimension arrays
foreach ($this->$attr_name as &$attr_array_value)
{
if (is_object($attr_array_value))
```

```
{
$attr_array_value = clone $attr_array_value;
}
unset($attr_array_value);
}
}
}
}
}
?>

Example:
<?php
class foo extends clone_base
{
public $attr = "Hello";
public $b = null;
public $attr2 = array();

public function __construct()
{
$this->b = new bar("World");
$this->attr2[] = new bar("What's");
$this->attr2[] = new bar("up?");
}
}

class bar extends clone_base
{
public $attr;

public function __construct($attr_value)
{
$this->attr = $attr_value;
}
}

echo "<pre>";

$f1 = new foo();
$f2 = clone $f1;
$f2->attr = "James";
$f2->b->attr = "Bond";
$f2->attr2[0]->attr = "Agent";
$f2->attr2[1]->attr = "007";

echo "f1.attr = " . $f1->attr . "\n";
echo "f1.b.attr = " . $f1->b->attr . "\n";
echo "f1.attr2[0] = " . $f1->attr2[0]->attr . "\n";
echo "f1.attr2[1] = " . $f1->attr2[1]->attr . "\n";
echo "\n";
echo "f2.attr = " . $f2->attr . "\n";
echo "f2.b.attr = " . $f2->b->attr . "\n";
echo "f2.attr2[0] = " . $f2->attr2[0]->attr . "\n";
echo "f2.attr2[1] = " . $f2->attr2[1]->attr . "\n";
?>
```

up
down
0
fabio at naoimporta dot com ¶
7 years ago

It's possible to know how many clones have been created of a object. I'm think that is correct:

```
<?php

class Classe {

public static $howManyClones = 0;

public function __clone() {
++static::$howManyClones;
}

public static function howManyClones() {
return static::$howManyClones;
}

public function __destruct() {
--static::$howManyClones;
}
}

$a = new Classe;

$b = clone $a;
$c = clone $b;
$d = clone $c;

echo 'Clones:' . Classe::howManyClones() . PHP_EOL;

unset($d);

echo 'Clones:' . Classe::howManyClones() . PHP_EOL;
```

up
down
-1
flaviu dot chelaru at gmail dot com ¶
5 years ago

```
<?php

class Foo
{
private $bar = 1;

public function get()
{
$x = clone $this;
return $x->bar;
}
}
// will NOT throw exception.
// Foo::$bar property is visible internally even if called as external on the clone
print (new Foo)->get();
```

up
down
-3
yinzw at chuchujie dot com ¶
7 years ago

It's clearly depicted in the manual, about the mechanism of clone process:
- First, shallow copy: properties of references will keep references (refer to the same target/variable)
- Then, change content/property as requested (calling __clone method which is defined by user).

To illustrate this process, the following example codes seems better, comparing the the original version:

```
class SubObject
{
static $num_cons = 0;
static $num_clone = 0;

public $construct_value;
public $clone_value;

public function __construct() {
$this->construct_value = ++self::$num_cons;
}

public function __clone() {
$this->clone_value = ++self::$num_clone;
}
}

class MyCloneable
{
public $object1;
public $object2;

function __clone()
{
// 制制一this->object, 否然指向同一个象
$this->object1 = clone $this->object1;
}
}

$obj = new MyCloneable();

$obj->object1 = new SubObject();
$obj->object2 = new SubObject();

$obj2 = clone $obj;

print("Original Object:\n");
print_r($obj);
echo '<br>';
print("Cloned Object:\n");
print_r($obj2);

==================

the output is as below

Original Object:
MyCloneable Object
(
[object1] => SubObject Object
(
[construct_value] => 1
[clone_value] =>
)

[object2] => SubObject Object
(
[construct_value] => 2
[clone_value] =>
)

)
```

```
<br>Cloned Object:
MyCloneable Object
(
[object1] => SubObject Object
(
[construct_value] => 1
[clone_value] => 1
)

[object2] => SubObject Object
(
[construct_value] => 2
[clone_value] =>
)

)
```

up
down
-4
crrodriguez at suse dot de ¶
15 years ago

Keep in mind that since PHP 5.2.5, trying to clone a non-object correctly results in a fatal error, this differs from previous versions where only a Warning was thrown.

up
down
-7
cheetah at tanabi dot org ¶
15 years ago

Want deep cloning without too much hassle?

```
<?php
function __clone() {
foreach($this as $key => $val) {
if(is_object($val)||(is_array($val))){
$this->{$key} = unserialize(serialize($val));
}
}
}
?>
```

That will insure any object, or array that may potentially contain objects, will get cloned without using recursion or other support methods.

[EDIT BY danbrown AT php DOT net: An almost exact function was contributed on 02-DEC-2008-10:18 by (david ashe AT metabin):

```
<?php
function __clone(){
foreach($this as $name => $value){
if(gettype($value)=='object'){
$this->$name= clone($this->$name);
}
}
}
?>
```

Giving credit where it's due. ~DPB]
[EDIT BY cmb AT php DOT net: the latter function fails to make deep copies of object arrays, and might end up with infinite recursion.]

up

down
-8
jason at jewelrysupply dot com ¶
8 years ago
@DPB

I believe the two functions are not quite the same. The serialize followed by deserialize method is the way I've done deep cloning in other languages (bypasses any weird clone function behavior and ensures you have a no-strings-attached copy of the object).