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

Позднее статическое связывание »
« Клонирование объектов

- Руководство по PHP
- Справочник языка
- Классы и объекты

Change language: Russian

Сравнение объектов

При использовании оператора сравнения (==), свойства объектов просто сравниваются друг с другом, а именно: два объекта равны, если они имеют одинаковые атрибуты и значения (значения сравниваются через ==) и являются экземплярами одного и того же класса.

С другой стороны, при использовании оператора идентичности (===), переменные, содержащие объекты, считаются идентичными только тогда, когда они ссылаются на один и тот же экземпляр одного и того же класса.

Следующий пример пояснит эти правила.

Пример #1 Пример сравнения объектов

```php
<?php
function bool2str($bool)
{
return (string) $bool;
}

function compareObjects(&$o1, &$o2)
{
echo 'o1 == o2 : ' . bool2str($o1 == $o2) . "\n";
echo 'o1 != o2 : ' . bool2str($o1 != $o2) . "\n";
echo 'o1 === o2 : ' . bool2str($o1 === $o2) . "\n";
echo 'o1 !== o2 : ' . bool2str($o1 !== $o2) . "\n";
}

class Flag
{
public $flag;

function __construct($flag = true) {
$this->flag = $flag;
}
}

class OtherFlag
{
public $flag;

function __construct($flag = true) {
$this->flag = $flag;
}
}

$o = new Flag();
$p = new Flag();
$q = $o;
$r = new OtherFlag();

echo "Два экземпляра одного и того же класса\n";
compareObjects($o, $p);

echo "\nДве ссылки на один и тот же экземпляр\n";
compareObjects($o, $q);

echo "\nЭкземпляры двух разных классов\n";
compareObjects($o, $r);
?>
```

Результат выполнения приведённого примера:

```
Два экземпляра одного и того же класса
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : FALSE
o1 !== o2 : TRUE

Две ссылки на один и тот же экземпляр
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : TRUE
o1 !== o2 : FALSE

Экземпляры двух разных классов
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

Замечание:

Модули могут определять собственные правила для сравнения своих объектов (==).

+ add a note

User Contributed Notes 12 notes

up
down
84
jazfresh at hotmail.com ¶
17 years ago

Note that when comparing object attributes, the comparison is recursive (at least, it is with PHP 5.2). That is, if $a->x contains an object then that will be compared with $b->x in the same manner. Be aware that this can lead to recursion errors:

```
<?php
class Foo {
public $x;
}
$a = new Foo();
$b = new Foo();
$a->x = $b;
$b->x = $a;

print_r($a == $b);
?>
Results in:
PHP Fatal error: Nesting level too deep - recursive dependency? in test.php on line 11
```

up
down
63
Anonymous ¶
13 years ago

Comparison using <> operators should be documented. Between two objects, at least in PHP5.3, the comparison operation stops and returns at the first unequal property found.

```
<?php

$o1 = new stdClass();
$o1->prop1 = 'c';
$o1->prop2 = 25;
$o1->prop3 = 201;
$o1->prop4 = 1000;

$o2 = new stdClass();
$o2->prop1 = 'c';
$o2->prop2 = 25;
```

```
$o2->prop3 = 200;
$o2->prop4 = 9999;

echo (int)($o1 < $o2); // 0
echo (int)($o1 > $o2); // 1

$o1->prop3 = 200;

echo (int)($o1 < $o2); // 1
echo (int)($o1 > $o2); // 0

?>
```
up
down
7
rnealxp at yahoo dot com ¶
6 years ago
These three functions call themselves recursively and handle any nesting levels of arrays/objects/values and do strict comparisons. The entry-point to this function set would be "valuesAreIdentical".

```
<?php
function valuesAreIdentical($v1, $v2): bool {
$type1 = gettype($v1);
$type2 = gettype($v2);

if($type1 !== $type2){
return false;
}

switch(true){
case ($type1==='boolean' || $type1==='integer' || $type1==='double' || $type1==='string'):
//Do strict comparison here.
if($v1 !== $v2){
return false;
}
break;

case ($type1==='array'):
$bool = arraysAreIdentical($v1, $v2);
if($bool===false){
return false;
}
break;

case 'object':
$bool = objectsAreIdentical($v1,$v2);
if($bool===false){
return false;
}
break;

case 'NULL':
//Since both types were of type NULL, consider their "values" equal.
break;

case 'resource':
//How to compare if at all?
break;

case 'unknown type':
//How to compare if at all?
break;
```

```
} //end switch

//All tests passed.
return true;
}

function objectsAreIdentical($o1, $o2): bool {
//See if loose comparison passes.
if($o1 != $o2){
return false;
}

//Now do strict(er) comparison.
$objReflection1 = new ReflectionObject($o1);
$objReflection2 = new ReflectionObject($o2);

$arrProperties1 = $objReflection1->getProperties(ReflectionProperty::IS_PUBLIC);
$arrProperties2 = $objReflection2->getProperties(ReflectionProperty::IS_PUBLIC);

$bool = arraysAreIdentical($arrProperties1, $arrProperties2);
if($bool===false){
return false;
}

foreach($arrProperties1 as $key=>$propName){
$bool = valuesAreIdentical($o1->$propName, $o2->$propName);
if($bool===false){
return false;
}
}

//All tests passed.
return true;
}

function arraysAreIdentical(array $arr1, array $arr2): bool {
$count = count($arr1);

//Require that they have the same size.
if(count($arr2) !== $count){
return false;
}

//Require that they have the same keys.
$arrKeysInCommon = array_intersect_key($arr1, $arr2);
if(count($arrKeysInCommon)!== $count){
return false;
}

//Require that their keys be in the same order.
$arrKeys1 = array_keys($arr1);
$arrKeys2 = array_keys($arr2);
foreach($arrKeys1 as $key=>$val){
if($arrKeys1[$key] !== $arrKeys2[$key]){
return false;
}
}

//They do have same keys and in same order.
foreach($arr1 as $key=>$val){
$bool = valuesAreIdentical($arr1[$key], $arr2[$key]);
if($bool===false){
```

```
return false;
}
}

//All tests passed.
return true;
}
?>
```
up
down
0
rnealxp at yahoo dot com ¶
3 years ago
Please use this corrected version of function "valuesAreIdentical" instead of that which I previously posted (dependencies found in previous post); if an Admin can just replace the fn snippet, awesome/thanks, otherwise, apologies.
```
<?php
public static function valuesAreIdentical($v1, $v2):bool{
$type1 = gettype($v1);
$type2 = gettype($v2);
switch(true){
case ($type1 !== $type2):
return false;
case ($type1==='boolean' || $type1==='integer' || $type1==='double' || $type1==='string'):
//Do strict comparison here.
return ($v1===$v2);
case ($type1==='array'):
return self::arraysAreIdentical($v1, $v2);
case ($type1==='object'):
return self::objectsAreIdentical($v1,$v2);
case ($type1==='NULL'):
//Since both types were of type NULL, consider their "values" equal.
return true;
case ($type1==='resource' || $type1==='unknown type'):
//How to compare if at all?
return true;
default:
return true; //Code-flow not intended to arrive here.
} //end switch
}
?>
```
up
down
-1
nhuhoai ¶
9 years ago
For comparison about two objects in a class, you can use an interface like this and customize your functions for each class:

```
<?php
interface EQU {
public static function compare( EQU $me, EQU $you );
public function equals( EQU $you );
}
?>
```

If you gotcha a super class, you can make generic functions (not safe but work with not complex class):

```
<?php
abstract class SuperClass {
public function __construct( ) {
// do what you need
}
public static function compare( $obj1, $obj2 ) {
return serialize( $obj1 ) == serialize( $obj2 );
}
public function equals( $obj ) {
return static::compare( $this, $obj );
}
}
?>
```

up
down
-2
rune at zedeler dot dk ¶
16 years ago

```
Whoops, apparently I hadn't checked the array-part of the below very well.
Forgot to test if the arrays had same length, and had some misaligned parenthesis.
This one should work better :+)

<?
function deepCompare($a,$b) {
if(is_object($a) && is_object($b)) {
if(get_class($a)!=get_class($b))
return false;
foreach($a as $key => $val) {
if(!deepCompare($val,$b->$key))
return false;
}
return true;
}
else if(is_array($a) && is_array($b)) {
while(!is_null(key($a)) && !is_null(key($b))) {
if (key($a)!==key($b) || !deepCompare(current($a),current($b)))
return false;
next($a); next($b);
}
return is_null(key($a)) && is_null(key($b));
}
else
return $a===$b;
}
?>
```

up
down
-4
wbcarts at juno dot com ¶
15 years ago

```
COMPARING OBJECTS using PHP's usort() method.

PHP and MySQL both provide ways to sort your data already, and it is a good idea to use that if possible. However, since this section is on comparing your own PHP objects (and that you may need to alter the sorting method in PHP), here is an example of how you can do that using PHP's "user-defined" sort method, usort() and your own class compare() methods.

<?php

/*
* Employee.php
*
* This class defines a compare() method, which tells PHP the sorting rules
* for this object - which is to sort by emp_id.
*
*/
class Employee
{
```

```
public $first;
public $last;
public $emp_id; // the property we're interested in...

public function __construct($emp_first, $emp_last, $emp_ID)
{
$this->first = $emp_first;
$this->last = $emp_last;
$this->emp_id = $emp_ID;
}

/*
* define the rules for sorting this object - using emp_id.
* Make sure this function returns a -1, 0, or 1.
*/
public static function compare($a, $b)
{
if ($a->emp_id < $b->emp_id) return -1;
else if($a->emp_id == $b->emp_id) return 0;
else return 1;
}

public function __toString()
{
return "Employee[first=$this->first, last=$this->last, emp_id=$this->emp_id]";
}
}

# create a PHP array and initialize it with Employee objects.
$employees = array(
new Employee("John", "Smith", 345),
new Employee("Jane", "Doe", 231),
new Employee("Mike", "Barnes", 522),
new Employee("Vicky", "Jones", 107),
new Employee("John", "Doe", 2),
new Employee("Kevin", "Patterson", 89)
);

# sort the $employees array using Employee compare() method.
usort($employees, array("Employee", "compare"));

# print the results
foreach($employees as $employee)
{
echo $employee . '<br>';
}
?>
```

Results are now sorted by emp_id:

```
Employee[first=John, last=Doe, emp_id=2]
Employee[first=Kevin, last=Patterson, emp_id=89]
Employee[first=Vicky, last=Jones, emp_id=107]
Employee[first=Jane, last=Doe, emp_id=231]
Employee[first=John, last=Smith, emp_id=345]
Employee[first=Mike, last=Barnes, emp_id=522]
```

Important Note: Your PHP code will never directly call the Employee's compare() method, but PHP's usort() calls it many many times. Also, when defining the rules for sorting, make sure to get to a "primitive type" level... that is, down to a number or string, and that the function returns a -1, 0, or 1, for reliable and consistent results.

Also see: http://www.php.net/manual/en/function.usort.php for more examples of PHP's sorting facilities.

up
down
-3
cross+php at distal dot com ¶
15 years ago
In response to "rune at zedeler dot dk"s comment about class contents being equal, I have a similar issue. I want to sort an array of objects using sort().

I know I can do it with usort(), but I'm used to C++ where you can define operators that allow comparison. I see in the zend source code that it calls a compare_objects function, but I don't see any way to implement that function for an object. Would it have to be an extension to provide that interface?

If so, I'd like to suggest that you allow equivalence and/or comparison operations to be defined in a class definition in PHP. Then, the sorts of things rune and I want to do would be much easier.
up
down
-1
cpmjr1 at gmail dot com ¶
11 months ago
It is not immediately obvious based on the docs, but the equality comparison operator does also check protected and private properties.

Example:
```
<?php
class A { public $a = 0; private $b = 1; public function __construct($test) {$this->b = $test;}}
echo "A(1) == A(2) " . var_export((new A(1)) == (new A(2)), true) . "\n";
echo "A(1) == A(1) " . var_export((new A(1)) == (new A(1)), true) . "\n";
?>
```
Output:
```
A(1) == A(2) false
A(1) == A(1) true
```
up
down
-3
rune at zedeler dot dk ¶
16 years ago
I haven't found a build-in function to check whether two obects are identical - that is, all their fields are identical.
In other words,

```
<?
class A {
var $x;
function __construct($x) { $this->x = $x; }

}
$identical1 = new A(42);
$identical2 = new A(42);
$different = new A('42');
?>
```

Comparing the objects with "==" will claim that all three of them are equal. Comparing with "===" will claim that all are un-equal.
I have found no build-in function to check that the two identicals are
identical, but not identical to the different.

The following function does that:

```
<?
function deepCompare($a,$b) {
if(is_object($a) && is_object($b)) {
if(get_class($a)!=get_class($b))
return false;
foreach($a as $key => $val) {
if(!deepCompare($val,$b->$key))
return false;
}
return true;
}
else if(is_array($a) && is_array($b)) {
while(!is_null(key($a) && !is_null(key($b)))) {
if (key($a)!==key($b) || !deepCompare(current($a),current($b)))
return false;
next($a); next($b);
}
return true;
}
else
return $a===$b;
}
?>
```

up
down
-4
Hayley Watson ¶
15 years ago

This has already been mentioned (see jazfresh at hotmail.com's note), but here it is again in more detail because for objects the difference between == and === is significant.

Loose equality (==) over objects is recursive: if the properties of the two objects being compared are themselves objects, then those properties will also be compared using ==.

```
<?php
class Link
{
public $link; function __construct($link) { $this->link = $link; }
}
class Leaf
{
public $leaf; function __construct($leaf) { $this->leaf = $leaf; }
}

$leaf1 = new Leaf(42);
$leaf2 = new Leaf(42);

$link1 = new Link($leaf1);
$link2 = new Link($leaf2);

echo "Comparing Leaf object equivalence: is \$leaf1==\$leaf2? ", ($leaf1 == $leaf2 ? "Yes" : "No"), "\n";
echo "Comparing Leaf object identity: is \$leaf1===\$leaf2? ", ($leaf1 === $leaf2 ? "Yes" : "No"), "\n";
echo "\n";
echo "Comparing Link object equivalence: is \$link1==\$link2? ",($link1 == $link2 ? "Yes" : "No"), "\n";
echo "Comparing Link object identity: is \$link1===\$link2? ", ($link1 === $link2 ? "Yes" : "No"), "\n";
?>
```

Even though $link1 and $link2 contain different Leaf objects, they are still equivalent because the Leaf objects are themselves equivalent.

The practical upshot is that using "==" when "===" would be more appropriate can result in a severe performance penalty, especially if the objects are large and/or complex. In fact, if there are any circular relationships involved between the objects or (recursively) any of their properties, then a fatal error can result because of the implied infinite loop.

```
<?php
class Foo { public $foo; }
$t = new Foo; $t->foo = $t;
```

```
$g = new Foo; $g->foo = $g;

echo "Strict identity: ", ($t===$g ? "True" : "False"),"\n";
echo "Loose equivalence: ", ($t==$g ? "True" : "False"), "\n";
?>
```

So preference should be given to comparing objects with "===" rather than "=="; if two distinct objects are to be compared for equivalence, try to do so by examining suitable individual properties. (Maybe PHP could get a magic "__equals" method that gets used to evaluate "=="? :))

up
down
-7
***Oddant* ¶**
10 years ago

This example is way too much confusing, if you new to php comparison motor, you should think (after reading this example) that '==' is actually comparing the type of the objects. that's not true, it actually compares the type of the objects AND the properties of them.

```
<?php

class A {
private $value;
function __construct ($value)
{
$this->value = $value;
}
}
class B {
private $value;
function __construct ($value)
{
$this->value = $value;
}
}

$a1 = new A (1);
$a2 = new A (2);
$b1 = new B (1);

var_dump( $a1 == $a2 );
var_dump( $a1 == $b1 );
?>
```

+add a note