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

Ключевое слово static »
« Наследование

Change language: Russian

Оператор разрешения области видимости (::)

Оператор разрешения области видимости (называемый также Paamayim Nekudotayim) или, проще говоря, «двойное двоеточие» — это лексема, разрешающая обращаться к константе, статическому свойству или статическому методу класса или одному из его родителей. Кроме этого, статические свойства или методы разрешено переопределять через позднее статическое связывание.

При обращении к этим элементам извне класса указывают имя этого класса.

Можно обращаться к классу через переменную. Значение переменной не должно быть ключевым словом (например, `self`, `parent` или `static`).

Paamayim Nekudotayim только вначале кажется странным словосочетанием для обозначения двойного двоеточия. Однако, пока писался движок Zend Engine версии 0.5 (который входил в PHP3), команда Zend решила так и назвать его. Вообще-то оно и означает «двойное двоеточие» — на иврите!

Пример #1 Использование :: вне объявления класса

```
<?php
class MyClass {
const CONST_VALUE = 'Значение константы';
}

$classname = 'MyClass';
echo $classname::CONST_VALUE;

echo MyClass::CONST_VALUE;
?>
```

К свойствам и методам внутри самого класса обращаются через ключевые слова *self*, *parent* и *static*.

Пример #2 Использование :: внутри объявления класса

```
<?php
class OtherClass extends MyClass
{
public static $my_static = 'статическая переменная';

public static function doubleColon() {
echo parent::CONST_VALUE . "\n";
echo self::$my_static . "\n";
}
}

$classname = 'OtherClass';
$classname::doubleColon();

OtherClass::doubleColon();
?>
```

Когда дочерний класс переопределяет методы родительского класса, PHP не вызывает методы родительского класса автоматически. Будет ли вызыван метод родительского класса, зависит от дочернего. Это правило также распространяется на конструкторы и деструкторы, перегруженные и «магические» методы.

Пример #3 Обращение к методу в родительском классе

```
<?php
class MyClass
{
protected function myFunc() {
echo "MyClass::myFunc()\n";
}
}
```

```
class OtherClass extends MyClass
{
// Переопределить родительское определение
public function myFunc()
{
// Но всё ещё вызываем родительскую функцию
parent::myFunc();
echo "OtherClass::myFunc()\n";
}
}

$class = new OtherClass();
$class->myFunc();
?>
```

Смотрите также некоторые примеры статических вызовов.

+add a note

User Contributed Notes 11 notes

up
down
224
Theriault ¶
14 years ago

```
A class constant, class property (static), and class function (static) can all share the same name and be accessed using the double-colon.

<?php

class A {

public static $B = '1'; # Static class variable.

const B = '2'; # Class constant.

public static function B() { # Static class function.
return '3';
}

}

echo A::$B . A::B . A::B(); # Outputs: 123
?>
```

up
down
103
1naveengiri at gmail dot com ¶
6 years ago

```
In PHP, you use the self keyword to access static properties and methods.

The problem is that you can replace $this->method() with self::method() anywhere, regardless if method() is declared static or not. So which one should you use?

Consider this code:

class ParentClass {
function test() {
self::who(); // will output 'parent'
$this->who(); // will output 'child'
}

function who() {
echo 'parent';
}
}

class ChildClass extends ParentClass {
function who() {
echo 'child';
}
}

$obj = new ChildClass();
$obj->test();
```
In this example, self::who() will always output ‘parent’, while $this->who() will depend on what class the object has.

Now we can see that self refers to the class in which it is called, while $this refers to the class of the current object.

So, you should use self only when $this is not available, or when you don’t want to allow descendant classes to overwrite the current method.

up
down
33
guy at syntheticwebapps dot com ¶
10 years ago

It seems as though you can use more than the class name to reference the static variables, constants, and static functions of a class definition from outside that class using the :: . The language appears to allow you to use the object itself.

For example:
```
class horse
{
static $props = {'order'=>'mammal'};
}
$animal = new horse();
echo $animal::$props['order'];

// yields 'mammal'
```
This does not appear to be documented but I see it as an important convenience in the language. I would like to see it documented and supported as valid.

If it weren't supported officially, the alternative would seem to be messy, something like this:
```
$animalClass = get_class($animal);
echo $animalClass::$props['order'];
```
up
down
22
jasverix at NOSPAM dot gmail dot com ¶
10 years ago

Just found out that using the class name may also work to call similar function of anchestor class.
```
<?php

class Anchestor {

public $Prefix = '';

private $_string = 'Bar';
public function Foo() {
return $this->Prefix.$this->_string;
}
}

class MyParent extends Anchestor {
public function Foo() {
$this->Prefix = null;
return parent::Foo().'Baz';
}
}

class Child extends MyParent {
public function Foo() {
$this->Prefix = 'Foo';
return Anchestor::Foo();
}
}

$c = new Child();
echo $c->Foo(); //return FooBar, because Prefix, as in Anchestor::Foo()

?>

The Child class calls at Anchestor::Foo(), and therefore MyParent::Foo() is never run.
```

up
down
12
giovanni at gargani dot it ¶
14 years ago

Well, a "swiss knife" couple of code lines to call parent method. The only limit is you can't use it with "by reference" parameters.
Main advantage you dont need to know the "actual" signature of your super class, you just need to know which arguments do you need

```
<?php
class someclass extends some superclass {
// usable for constructors
function __construct($ineedthisone) {
$args=func_get_args();
/* $args will contain any argument passed to __construct.
* Your formal argument doesnt influence the way func_get_args() works
*/
call_user_func_array(array('parent',__FUNCTION__),$args);
}
// but this is not for __construct only
function anyMethod() {
$args=func_get_args();
call_user_func_array(array('parent',__FUNCTION__),$args);
}
// Note: php 5.3.0 will even let you do
function anyMethod() {
//Needs php >=5.3.x
call_user_func_array(array('parent',__FUNCTION__),func_get_args());
}

}
?>
```

up
down
10
remy dot damour at ----no-spam---laposte dot net ¶
13 years ago

As of php 5.3.0, you can use 'static' as scope value as in below example (add flexibility to inheritance mechanism

```
compared to 'self' keyword...)

<?php

class A {
const C = 'constA';
public function m() {
echo static::C;
}
}

class B extends A {
const C = 'constB';
}

$b = new B();
$b->m();

// output: constB
?>
```
up
down
-1
gazianis2200 at gmail dot com ¶
29 days ago
```
<?php
/**
*access a constant from outside a class
*/
class Foo{
public const A = "Constant A";
}
echo Foo::A;
echo "\n";

/**
*access a constant within its own class
*/

class Bar{
public const A = "Constant A";
public function abc(){
echo self::A;
echo "\n";
}
}

$obj = new Bar;
$obj->abc();

/**
*access a constant within her child class
*/

class Baz extends Bar{
public function abc(){
echo parent::A;
}
}
$obj = new Baz;
$obj->abc();
```

//Static property and static method also follows this principle.
up
down
2
wouter at interpotential dot com ¶
14 years ago
It's worth noting, that the mentioned variable can also be an object instance. This appears to be the easiest way to refer to a static function as high in the inheritance hierarchy as possible, as seen from the instance. I've encountered some odd behavior while using static::something() inside a non-static method.

See the following example code:

```
<?php
class FooClass {
public function testSelf() {
return self::t();
}

public function testThis() {
return $this::t();
}

public static function t() {
return 'FooClass';
}

function __toString() {
return 'FooClass';
}
}

class BarClass extends FooClass {
public static function t() {
return 'BarClass';
}

}

$obj = new BarClass();
print_r(Array(
$obj->testSelf(), $obj->testThis(),
));
?>
```

which outputs:

```
<pre>
Array
(
[0] => FooClass
[1] => BarClass
)
</pre>
```

As you can see, __toString has no effect on any of this. Just in case you were wondering if perhaps this was the way it's done.
up
down
2
luka8088 at gmail dot com ¶
15 years ago
Little static trick to go around php strict standards ...

Function caller founds an object from which it was called, so that static method can alter it, replacement for $this in static function but without strict warnings :)

```
<?php

error_reporting(E_ALL + E_STRICT);

function caller () {
$backtrace = debug_backtrace();
$object = isset($backtrace[0]['object']) ? $backtrace[0]['object'] : null;
$k = 1;

while (isset($backtrace[$k]) && (!isset($backtrace[$k]['object']) || $object === $backtrace[$k]['object']))
$k++;

return isset($backtrace[$k]['object']) ? $backtrace[$k]['object'] : null;
}

class a {

public $data = 'Empty';

function set_data () {
b::set();
}

}

class b {

static function set () {
// $this->data = 'Data from B !';
// using this in static function throws a warning ...
caller()->data = 'Data from B !';
}

}

$a = new a();
$a->set_data();
echo $a->data;

?>
```

Outputs: Data from B !

No warnings or errors !

up

down

0

csaba dot dobai at php-sparcle dot com ¶

15 years ago

For the 'late static binding' topic I published a code below, that demonstrates a trick for how to setting variable value in the late class, and print that in the parent (or the parent's parent, etc.) class.

```
<?php

class cA
{
/**
* Test property for using direct default value
*/
protected static $item = 'Foo';

/**
* Test property for using indirect default value
*/
protected static $other = 'cA';

public static function method()
{
print self::$item."\r\n"; // It prints 'Foo' on everyway... :(
print self::$other."\r\n"; // We just think that, this one prints 'cA' only, but... :)
}

public static function setOther($val)
{
self::$other = $val; // Set a value in this scope.
}
}

class cB extends cA
{
/**
* Test property with redefined default value
*/
protected static $item = 'Bar';

public static function setOther($val)
{
self::$other = $val;
}
}

class cC extends cA
{
/**
* Test property with redefined default value
*/
protected static $item = 'Tango';

public static function method()
{
print self::$item."\r\n"; // It prints 'Foo' on everyway... :(
print self::$other."\r\n"; // We just think that, this one prints 'cA' only, but... :)
}

/**
* Now we drop redeclaring the setOther() method, use cA with 'self::' just for fun.
*/
}

class cD extends cA
{
/**
* Test property with redefined default value
*/
protected static $item = 'Foxtrot';

/**
* Now we drop redeclaring all methods to complete this issue.
*/
}
```

```
cB::setOther('cB'); // It's cB::method()!
cB::method(); // It's cA::method()!
cC::setOther('cC'); // It's cA::method()!
cC::method(); // It's cC::method()!
cD::setOther('cD'); // It's cA::method()!
cD::method(); // It's cA::method()!

/**
* Results: ->
* Foo
* cB
* Tango
* cC
* Foo
* cD
*
* What the hell?! :)
*/

?>
```

up
down
0
developit at mail dot ru ¶
18 years ago

You use 'self' to access this class, 'parent' - to access parent class, and what will you do to access a parent of the parent? Or to access the very root class of deep class hierarchy? The answer is to use classnames. That'll work just like 'parent'. Here's an example to explain what I mean. Following code

```
<?php
class A
{
protected $x = 'A';
public function f()
{
return '['.$this->x.']';
}
}

class B extends A
{
protected $x = 'B';
public function f()
{
return '{'.$this->x.'}';
}
}

class C extends B
{
protected $x = 'C';
public function f()
{
return '('.$this->x.')'.parent::f().B::f().A::f();
}
}

$a = new A();
$b = new B();
$c = new C();

print $a->f().'<br/>';
print $b->f().'<br/>';
print $c->f().'<br/>';
?>

will output

[A] -- {B} -- (C){C}{C}[C]
```

+ add a note