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

Абстрактные классы »
« Оператор разрешения области видимости (::)

- Руководство по PHP
- Справочник языка
- Классы и объекты

Change language: Russian

# Ключевое слово static

**Подсказка**

Эта страница описывает использование ключевого слова `static` для определения статических методов и свойств. `static` также может использоваться для определения статических переменных, определения статических анонимных функций и позднего статического связывания. Для получения информации о таком применении ключевого слова `static` обратитесь по вышеуказанным страницам.

Объявление свойств и методов класса статическими позволяет обращаться к ним без создания экземпляра класса. К ним также можно получить доступ статически в созданном экземпляре объекта класса.

## Статические методы

Так как статические методы вызываются без создания экземпляра класса, то псевдопеременная *$this* недоступна внутри статических методов.

**Внимание**

Вызов нестатических методов статически вызывает ошибку Error.

До PHP 8.0.0 вызов нестатических методов статически был объявлен устаревшим и вызывал ошибку уровня **`E_DEPRECATED`**.

**Пример #1 Пример статического метода**

```
<?php
class Foo {
public static function aStaticMethod() {
// ...
}
}

Foo::aStaticMethod();
$classname = 'Foo';
$classname::aStaticMethod();
?>
```

## Статические свойства

Доступ к статическим свойствам осуществляется с помощью оператора разрешения области видимости (`::`), и к ним нельзя получить доступ через оператор объекта (`->`).

На класс можно ссылаться с помощью переменной. Значение переменной в таком случае не может быть ключевым словом (например, `self`, `parent` и `static`).

**Пример #2 Пример статического свойства**

```
<?php
class Foo
{
public static $my_static = 'foo';

public function staticValue() {
return self::$my_static;
}
}

class Bar extends Foo
{
public function fooStatic() {
return parent::$my_static;
}
```

```
}


print Foo::$my_static . "\n";

$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static . "\n"; // Не определено свойство my_static

print $foo::$my_static . "\n";
$classname = 'Foo';
print $classname::$my_static . "\n";

print Bar::$my_static . "\n";
$bar = new Bar();
print $bar->fooStatic() . "\n";
?>
```

Результат выполнения приведённого примера в PHP 8 аналогичен:

```
foo
foo

Notice: Accessing static property Foo::$my_static as non static in /in/V0Rvv on line 23

Warning: Undefined property: Foo::$my_static in /in/V0Rvv on line 23

foo
foo
foo
foo
```



## User Contributed Notes 28 notes


219
***inkredibl ¶***
**16 years ago**
Note that you should read "Variables/Variable scope" if you are looking for static keyword use for declaring static variables inside functions (or methods). I myself had this gap in my PHP knowledge until recently and had to google to find this out. I think this page should have a "See also" link to static function variables.
http://www.php.net/manual/en/language.variables.scope.php


152
***payal001 at gmail dot com ¶***
**12 years ago**
Here statically accessed property prefer property of the class for which it is called. Where as self keyword enforces use of current class only. Refer the below example:

```
<?php
class a{

static protected $test="class a";

public function static_test(){

echo static::$test; // Results class b
echo self::$test; // Results class a

}
```

```
}

class b extends a{

static protected $test="class b";

}

$obj = new b();
$obj->static_test();
?>
```

up
down
30

***artekpuck at gmail dot com ¶***

**5 years ago**

It is worth mentioning that there is only one value for each static variable that is the same for all instances

up
down
15

***Anonymous ¶***

**18 years ago**

You misunderstand the meaning of inheritance : there is no duplication of members when you inherit from a base class. Members are shared through inheritance, and can be accessed by derived classes according to visibility (public, protected, private).

The difference between static and non static members is only that a non static member is tied to an instance of a class although a static member is tied to the class, and not to a particular instance.
That is, a static member is shared by all instances of a class although a non static member exists for each instance of class.

Thus, in your example, the static property has the correct value, according to principles of object oriented conception.

```
class Base
{
public $a;
public static $b;
}

class Derived extends Base
{
public function __construct()
{
$this->a = 0;
parent::$b = 0;
}
public function f()
{
$this->a++;
parent::$b++;
}
}

$i1 = new Derived;
$i2 = new Derived;

$i1->f();
echo $i1->a, ' ', Derived::$b, "\n";
$i2->f();
echo $i2->a, ' ', Derived::$b, "\n";

outputs
1 1
```

1 2
up
down
8
***rahul dot anand77 at gmail dot com ¶***
**8 years ago**
To check if a method declared in a class is static or not, you can us following code. PHP5 has a Reflection Class, which is very helpful.

```
try {
$method = new ReflectionMethod( 'className::methodName );
if ( $method->isStatic() )
{
// Method is static.
}
}
catch ( ReflectionException $e )
{
// method does not exist
echo $e->getMessage();
}
```

*You can read more about Reflection class on http://php.net/manual/en/class.reflectionclass.php
up
down
16
***vinayak dot anivase at gmail dot com ¶***
**5 years ago**
This is also possible:

```
class Foo {
public static $bar = 'a static property';
}

$baz = (new Foo)::$bar;
echo $baz;
```
up
down
2
***admin at shopinson dot com ¶***
**3 years ago**
I used instantiation to access the access the a static property directly.

A Simple ticky art, you may apply (using object to access static property in a class) with the scope resolution operator

```
<?php
class Shopinson {
const MY_CONSTANT = 'the value of MY_CONSTANT ';
}

class Godwin extends Shopinson
{
public static $myconstant = ' The Paamayim Nekudotayim or double-colon.';
public function SaySomething(){
echo parent::MY_CONSTANT .PHP_EOL; // outputs: the value of MY_CONSTANT
echo self::$myconstant; // outputs: The Paamayim Nekudotayim or double-colon.
}
}

$my_class = new Godwin();
print $my_class::$myconstant;
$my_class::SaySomething();
echo Godwin::$myconstant;
```

```
Godwin::SaySomething();

?>

print $my_class::$myconstant;
```

up
down
9

***Anonymous ¶***

**10 years ago**

It should be noted that in 'Example #2', you can also call a variably defined static method as follows:

```
<?php
class Foo {
public static function aStaticMethod() {
// ...
}
}

$classname = 'Foo';
$methodname = 'aStaticMethod';
$classname::{$methodname}(); // As of PHP 5.3.0 I believe
?>
```

up
down
6

***sideshowAnthony at googlemail dot com ¶***

**8 years ago**

The static keyword can still be used (in a non-oop way) inside a function. So if you need a value stored with your class, but it is very function specific, you can use this:

```
class aclass {
public static function b(){
static $d=12; // Set to 12 on first function call only
$d+=12;
return "$d\n";
}
}

echo aclass::b(); //24
echo aclass::b(); //36
echo aclass::b(); //48
echo aclass::$d; //fatal error
```

up
down
10

***ASchmidt at Anamera dot net ¶***

**5 years ago**

It is important to understand the behavior of static properties in the context of class inheritance:

- Static properties defined in both parent and child classes will hold DISTINCT values for each class. Proper use of self:: vs. static:: are crucial inside of child methods to reference the intended static property.

- Static properties defined ONLY in the parent class will share a COMMON value.

```
<?php
declare(strict_types=1);

class staticparent {
static $parent_only;
static $both_distinct;
```

```
function __construct() {
static::$parent_only = 'fromparent';
static::$both_distinct = 'fromparent';
}
}

class staticchild extends staticparent {
static $child_only;
static $both_distinct;

function __construct() {
static::$parent_only = 'fromchild';
static::$both_distinct = 'fromchild';
static::$child_only = 'fromchild';
}
}

$a = new staticparent;
$a = new staticchild;

echo 'Parent: parent_only=', staticparent::$parent_only, ', both_distinct=', staticparent::$both_distinct, "<br/>\r\n";
echo 'Child: parent_only=', staticchild::$parent_only, ', both_distinct=', staticchild::$both_distinct, ', child_only=',
staticchild::$child_only, "<br/>\r\n";
?>

will output:
Parent: parent_only=fromchild, both_distinct=fromparent
Child: parent_only=fromchild, both_distinct=fromchild, child_only=fromchild
```

up
down
14
***davidn at xnet dot co dot nz ¶***
**14 years ago**

```
Static variables are shared between sub classes

<?php
class MyParent {

protected static $variable;
}

class Child1 extends MyParent {

function set() {

self::$variable = 2;
}
}

class Child2 extends MyParent {

function show() {

echo(self::$variable);
}
}

$c1 = new Child1();
$c1->set();
$c2 = new Child2();
$c2->show(); // prints 2
?>
```

up
down
14
***aidan at php dot net ¶***
**18 years ago**
To check if a function was called statically or not, you'll need to do:

```
<?php
function foo () {
$isStatic = !(isset($this) && get_class($this) == __CLASS__);
}
?>
```

More at (http://blog.phpdoc.info/archives/4-Schizophrenic-Methods.html).

(I'll add this to the manual soon).
up
down
4
***b1tchcakes ¶***
**7 years ago**
```
<?php

trait t {
protected $p;
public function testMe() {echo 'static:'.static::class. ' // self:'.self::class ."\n";}
}

class a { use t; }
class b extends a {}

echo (new a)->testMe();
echo (new b)->testMe();

outputs
static:a // self:t
static:b // self:t
```
up
down
3
***manishpatel2280 at gmail dot com ¶***
**10 years ago**
In real world, we can say will use static method when we dont want to create object instance.

```
e.g ...

validateEmail($email) {
if(T) return true;
return false;
}

//This makes not much sense
$obj = new Validate();
$result = $obj->validateEmail($email);

//This makes more sense
$result = Validate::validateEmail($email);
```
up
down
8
***tolean_dj at yahoo dot com ¶***
**13 years ago**

Starting with php 5.3 you can get use of new features of static keyword. Here's an example of abstract singleton class:

```
<?php

abstract class Singleton {

protected static $_instance = NULL;

/**
* Prevent direct object creation
*/
final private function __construct() { }

/**
* Prevent object cloning
*/
final private function __clone() { }

/**
* Returns new or existing Singleton instance
* @return Singleton
*/
final public static function getInstance(){
if(null !== static::$_instance){
return static::$_instance;
}
static::$_instance = new static();
return static::$_instance;
}

}
?>
```

up
down
11
***webmaster at removethis dot weird-webdesign dot de ¶***
**13 years ago**

On PHP 5.2.x or previous you might run into problems initializing static variables in subclasses due to the lack of late static binding:

```
<?php
class A {
protected static $a;

public static function init($value) { self::$a = $value; }
public static function getA() { return self::$a; }
}

class B extends A {
protected static $a; // redefine $a for own use

// inherit the init() method
public static function getA() { return self::$a; }
}

B::init('lala');
echo 'A::$a = '.A::getA().'; B::$a = '.B::getA();
?>
```

This will output:
A::$a = lala; B::$a =

If the init() method looks the same for (almost) all subclasses there should be no need to implement init() in every subclass and by that producing redundant code.

Solution 1:
Turn everything into non-static. BUT: This would produce redundant data on every object of the class.

Solution 2:
Turn static $a on class A into an array, use classnames of subclasses as indeces. By doing so you also don't have to redefine $a for the subclasses and the superclass' $a can be private.

Short example on a DataRecord class without error checking:

```
<?php
abstract class DataRecord {
private static $db; // MySQLi-Connection, same for all subclasses
private static $table = array(); // Array of tables for subclasses

public static function init($classname, $table, $db = false) {
if (!($db === false)) self::$db = $db;
self::$table[$classname] = $table;
}

public static function getDB() { return self::$db; }
public static function getTable($classname) { return self::$table[$classname]; }
}

class UserDataRecord extends DataRecord {
public static function fetchFromDB() {
$result = parent::getDB()->query('select * from '.parent::getTable('UserDataRecord').';');

// and so on ...
return $result; // An array of UserDataRecord objects
}
}

$db = new MySQLi(...);
UserDataRecord::init('UserDataRecord', 'users', $db);
$users = UserDataRecord::fetchFromDB();
?>
```

I hope this helps some people who need to operate on PHP 5.2.x servers for some reason. Late static binding, of course, makes this workaround obsolete.

up
down
7

***gratcypalma at gmail dot om ¶***

**12 years ago**

```
<?php
class foo {
private static $getInitial;

public static function getInitial() {
if (self::$getInitial == null)
self::$getInitial = new foo();
return self::$getInitial;
}
}

foo::getInitial();

/*
this is the example to use new class with static method..
```

```
i hope it help
*/

?>
```

up
down
3

***zerocool at gameinsde dot ru ¶***

**15 years ago**

Hi, here's my simple Singleton example, i think it can be useful for someone. You can use this pattern to connect to the database for example.

```
<?php

class MySingleton
{
private static $instance = null;

private function __construct()
{
$this-> name = 'Freddy';

}

public static function getInstance()
{
if(self::$instance == null)
{
print "Object created!<br>";
self::$instance = new self;

}

return self::$instance;

}

public function sayHello()
{
print "Hello my name is {$this-> name}!<br>";

}

public function setName($name)
{
$this-> name = $name;

}

}

//

$objA = MySingleton::getInstance(); // Object created!

$objA-> sayHello(); // Hello my name is Freddy!

$objA-> setName("Alex");

$objA-> sayHello(); // Hello my name is Alex!

$objB = MySingleton::getInstance();
```

```
$objB-> sayHello(); // Hello my name is Alex!

$objB-> setName("Bob");

$objA-> sayHello(); // Hello my name is Bob!

?>
```

up
down
5

***ssj dot narutovash at gmail dot com ¶***

**16 years ago**

It's come to my attention that you cannot use a static member in an HEREDOC string. The following code

```
class A
{
public static $BLAH = "user";

function __construct()
{
echo <<<EOD
<h1>Hello {self::$BLAH}</h1>
EOD;
}
}

$blah = new A();
```

produces this in the source code:

```
<h1>Hello {self::}</h1>
```

Solution:

before using a static member, store it in a local variable, like so:

```
class B
{
public static $BLAH = "user";

function __construct()
{
$blah = self::$BLAH;
echo <<<EOD
<h1>Hello {$blah}</h1>
EOD;
}
}
```

and the output's source code will be:

```
<h1>Hello user</h1>
```

up
down
1

***Jay Cain ¶***

**14 years ago**

Regarding the initialization of complex static variables in a class, you can emulate a static constructor by creating a static function named something like init() and calling it immediately after the class definition.

```
<?php
```

```
class Example {
private static $a = "Hello";
private static $b;

public static function init() {
self::$b = self::$a . " World!";
}
}
Example::init();
?>
```

up
down
2

***michalf at ncac dot torun dot pl ¶***

**18 years ago**

Inheritance with the static elements is a nightmare in php. Consider the following code:

```
<?php
class BaseClass{
public static $property;
}

class DerivedClassOne extends BaseClass{
}

class DerivedClassTwo extends BaseClass{
}

DerivedClassOne::$property = "foo";
DerivedClassTwo::$property = "bar";

echo DerivedClassOne::$property; //one would naively expect "foo"...
?>
```

What would you expect as an output? "foo"? wrong. It is "bar"!!! Static variables are not inherited, they point to the BaseClass::$property.

At this point I think it is a big pity inheritance does not work in case of static variables/methods. Keep this in mind and save your time when debugging.

best regards - michal

up
down
0

***Mirco ¶***

**13 years ago**

The simplest static constructor.

Because php does not have a static constructor and you may want to initialize static class vars, there is one easy way, just call your own function directly after the class definition.

for example.

```
<?php
function Demonstration()
{
return 'This is the result of demonstration()';
}

class MyStaticClass
{
//public static $MyStaticVar = Demonstration(); //!!! FAILS: syntax error
```

```
public static $MyStaticVar = null;

public static function MyStaticInit()
{
//this is the static constructor
//because in a function, everything is allowed, including initializing using other functions

self::$MyStaticVar = Demonstration();
}
} MyStaticClass::MyStaticInit(); //Call the static constructor

echo MyStaticClass::$MyStaticVar;
//This is the result of demonstration()
?>
```

up
down
-2

***jkenigso at utk dot edu ¶***

**10 years ago**

```
It bears mention that static variables (in the following sense) persist:

<?php
class StaticVars
{
public static $a=1;
}
$b=new StaticVars;
$c=new StaticVars;

echo $b::$a; //outputs 1
$c::$a=2;
echo $b::$a; //outputs 2!
?>

Note that $c::$a=2 changed the value of $b::$a even though $b and $c are totally different objects.
```

up
down
-2

***valentin at balt dot name ¶***

**14 years ago**

```
How to implement a one storage place based on static properties.

<?php
class a {

public function get () {
echo $this->connect();
}
}
class b extends a {
private static $a;

public function connect() {
return self::$a = 'b';
}
}
class c extends a {
private static $a;

public function connect() {
return self::$a = 'c';
}
```

```
}
$b = new b ();
$c = new c ();

$b->get();
$c->get();
?>
```

up
down
-2

***michael at digitalgnosis dot removethis dot com ¶***

**19 years ago**

If you are trying to write classes that do this:

```
<?php

class Base
{
static function Foo ()
{
self::Bar();
}
}

class Derived extends Base
{
function Bar ()
{
echo "Derived::Bar()";
}
}

Derived::Foo(); // we want this to print "Derived::Bar()"

?>
```

Then you'll find that PHP can't (unless somebody knows the Right Way?) since 'self::' refers to the class which owns the /code/, not the actual class which is called at runtime. (__CLASS__ doesn't work either, because: A. it cannot appear before ::, and B. it behaves like 'self')

But if you must, then here's a (only slightly nasty) workaround:

```
<?php

class Base
{
function Foo ( $class = __CLASS__ )
{
call_user_func(array($class,'Bar'));
}
}

class Derived extends Base
{
function Foo ( $class = __CLASS__ )
{
parent::Foo($class);
}

function Bar ()
{
echo "Derived::Bar()";
```

```
}
}

Derived::Foo(); // This time it works.

?>
```

Note that Base::Foo() may no longer be declared 'static' since static methods cannot be overridden (this means it will trigger errors if error level includes E_STRICT.)

If Foo() takes parameters then list them before $class=__CLASS__ and in most cases, you can just forget about that parameter throughout your code.

The major caveat is, of course, that you must override Foo() in every subclass and must always include the $class parameter when calling parent::Foo().

up
down
-2

***vvikramraj at yahoo dot com ¶***

**15 years ago**

when attempting to implement a singleton class, one might also want to either
a) disable __clone by making it private
b) bash the user who attempts to clone by defining __clone to throw an exception

up
down
-3

***Mathijs Vos ¶***

**15 years ago**

```
<?php
class foo
{
public static $myStaticClass;

public function __construct()
{
self::myStaticClass = new bar();
}
}

class bar
{
public function __construct(){}
}
?>
```

Please note, this won't work.
Use self::$myStaticClass = new bar(); instead of self::myStaticClass = new bar(); (note the $ sign).
Took me an hour to figure this out.

up
down
-2

***fakhar_anwar123 at hotmail dot com ¶***

**3 years ago**

Asnwer selcted as correct solves problem. There is a valid use case (Design Pattern) where class with static member function needs to call non-static member function and before that this static members should also instantiate singleton using constructor a constructor.

**Case:**
For example, I am implementing Swoole HTTP Request event providing it a call-back as a Class with static member. Static Member does two things; it creates Singleton Object of the class by doing initialization in class constructor, and second this static members does is to call a non-static method 'run()' to handle Request (by bridging with Phalcon). Hence, static class without constructor and non-static call will not work for me.

+ add a note

- Классы и объекты
  - Введение
  - Основы
  - Свойства
  - Константы классов
  - Автоматическая загрузка классов
  - Конструкторы и деструкторы
  - Область видимости
  - Наследование
  - Оператор разрешения области видимости (::)
  - Ключевое слово static
  - Абстрактные классы
  - Интерфейсы объектов
  - Трейты
  - Анонимные классы
  - Перегрузка
  - Итераторы объектов
  - Магические методы
  - Ключевое слово final
  - Клонирование объектов
  - Сравнение объектов
  - Позднее статическое связывание
  - Объекты и ссылки
  - Сериализация объектов
  - Ковариантность и контравариантность
  - Журнал изменений ООП


- My PHP.net
- Contact
- Other PHP.net sites
- Privacy policy