- 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 final

PHP предоставляет ключевое слово `final`, разместив которое перед объявлениями методов или констант класса, можно предотвратить их переопределение в дочерних классах. Если же сам класс определяется с этим ключевым словом, то он не сможет быть унаследован.

Пример #1 Пример окончательных (final) методов

```
<?php
class BaseClass {
public function test() {
echo "Вызван метод BaseClass::test()\n";
}

final public function moreTesting() {
echo "Вызван метод BaseClass::moreTesting()\n";
}
}

class ChildClass extends BaseClass {
public function moreTesting() {
echo "Вызван метод ChildClass::moreTesting()\n";
}
}
// Выполнение заканчивается фатальной ошибкой: Cannot override final method BaseClass::moreTesting()
// (Окончательный (final) метод BaseClass::moreTesting() не может быть переопределён)
?>
```

Пример #2 Пример окончательного (final) класса

```
<?php
final class BaseClass {
public function test() {
echo "Вызван метод BaseClass::test()\n";
}

// Поскольку класс уже является final, ключевое слово final является избыточным
final public function moreTesting() {
echo "BaseClass::moreTesting() called\n";
}
}

class ChildClass extends BaseClass {
}
// Выполнение заканчивается фатальной ошибкой: Class ChildClass may not inherit from final class (BaseClass)
// (Класс ChildClass не может быть унаследован от окончательного класса (BaseClass))
?>
```

Пример #3 Пример окончательной (final) константы класса, начиная с PHP 8.1.0

```
<?php
class Foo
{
final public const X = "foo";
}

class Bar extends Foo
{
public const X = "bar";
}
```

```
// Ошибка: Bar::X не может переопределить окончательную константу Foo::X
?>
```

> **Замечание**: Свойства не могут быть объявлены окончательными: только классы, методы и константы (начиная с PHP 8.1.0) могут быть объявлены как окончательные (final). Начиная с PHP 8.0.0, закрытые методы не могут быть объявлены окончательными, за исключением конструктора.

+ add a note

User Contributed Notes 14 notes

up
down
300
jriddy at gmail dot com ¶
14 years ago

```
Note for Java developers: the 'final' keyword is not used for class constants in PHP. We use the keyword 'const'.

http://php.net/manual/en/language.oop5.constants.php
```

up
down
105
penartur at yandex dot ru ¶
16 years ago

```
Note that you cannot ovverride final methods even if they are defined as private in parent class.
Thus, the following example:
<?php
class parentClass {
final private function someMethod() { }
}
class childClass extends parentClass {
private function someMethod() { }
}
?>
dies with error "Fatal error: Cannot override final method parentClass::someMethod() in ***.php on line 7"

Such behaviour looks slight unexpected because in child class we cannot know, which private methods exists in a parent class and vice versa.

So, remember that if you defined a private final method, you cannot place method with the same name in child class.
```

up
down
51
someone dot else at elsewhere dot net ¶
10 years ago

```
@thomas at somewhere dot com

The 'final' keyword is extremely useful. Inheritance is also useful, but can be abused and becomes problematic in large applications. If you ever come across a finalized class or method that you wish to extend, write a decorator instead.

<?php
final class Foo
{
public method doFoo()
{
// do something useful and return a result
}
}

final class FooDecorator
{
private $foo;
public function __construct(Foo $foo)
{
$this->foo = $foo;
}

public function doFoo()
{
$result = $this->foo->doFoo();
// ... customize result ...
return $result;
}
}
?>
```
up
down
10
santoshjoshi2003 at yahoo dot co dot in ¶
15 years ago
```
The use of final keyword is just like that occurs in Java
In java final has three uses
1) prevent class Inheritance
2) prevent method overriding or redifination of
method in subclass
3) and to declare constants
But the third point seems to be missing from the PHP
I guess, as i am a java developer Currently gaining competence in PHP
```
up
down
10
cottton at i-stats dot net ¶
9 years ago
```
imo good to know:
<?php
class BaseClass
{
protected static $var = 'i belong to BaseClass';

public static function test()
{
echo '<hr>'.
'i am `'.__METHOD__.'()` and this is my var: `'.self::$var.'`<br>';
}
public static function changeVar($val)
{
self::$var = $val;
echo '<hr>'.
'i am `'.__METHOD__.'()` and i just changed my $var to: `'.self::$var.'`<br>';
}
final public static function dontCopyMe($val)
{
self::$var = $val;
echo '<hr>'.
'i am `'.__METHOD__.'()` and i just changed my $var to: `'.self::$var.'`<br>';
}
}

class ChildClass extends BaseClass
{
protected static $var = 'i belong to ChildClass';

public static function test()
{
echo '<hr>'.
'i am `'.__METHOD__.'()` and this is my var: `'.self::$var.'`<br>'.
'and this is my parent var: `'.parent::$var.'`';
}
public static function changeVar($val)
{
self::$var = $val;
echo '<hr>'.
'i am `'.__METHOD__.'()` and i just changed my $var to: `'.self::$var.'`<br>'.
'but the parent $var is still: `'.parent::$var.'`';
}
public static function dontCopyMe($val) // Fatal error: Cannot override final method BaseClass::dontCopyMe() in ...
{
self::$var = $val;
echo '<hr>'.
'i am `'.__METHOD__.'()` and i just changed my $var to: `'.self::$var.'`<br>';
}
}

BaseClass::test(); // i am `BaseClass::test()` and this is my var: `i belong to BaseClass`
ChildClass::test(); // i am `ChildClass::test()` and this is my var: `i belong to ChildClass`
// and this is my parent var: `i belong to BaseClass`
ChildClass::changeVar('something new'); // i am `ChildClass::changeVar()` and i just changed my $var to: `something new`
// but the parent $var is still: `i belong to BaseClass`
BaseClass::changeVar('something different'); // i am `BaseClass::changeVar()` and i just changed my $var to: `something different`
BaseClass::dontCopyMe('a text'); // i am `BaseClass::dontCopyMe()` and i just changed my $var to: `a text`
ChildClass::dontCopyMe('a text'); // Fatal error: Cannot override final method BaseClass::dontCopyMe() in ...
?>
```
up
down
10
mattsch at gmail dot com ¶
9 years ago

You can use final methods to replace class constants. The reason for this is you cannot unit test a class constant used in another class in isolation because you cannot mock a constant. Final methods allow you to have the same functionality as a constant while keeping your code loosely coupled.

Tight coupling example (bad to use constants):

```
<?php
interface FooInterface
{
}

class Foo implements FooInterface
{
const BAR = 1;

public function __construct()
{
}
}

interface BazInterface
{
public function getFooBar();
}

// This class cannot be unit tested in isolation because the actual class Foo must also be loaded to get the value of Foo::BAR
```

```
class Baz implements BazInterface
{
private $foo;

public function __construct(FooInterface $foo)
{
$this->foo = $foo;
}

public function getFooBar()
{
return Foo::BAR;
}

}

$foo = new Foo();
$baz = new Baz($foo);
$bar = $baz->getFooBar();
?>
```

Loose coupling example (eliminated constant usage):

```
<?php
interface FooInterface
{
public function bar();
}

class Foo implements FooInterface
{
public function __construct()
{
}

final public function bar()
{
return 1;
}
}

interface BazInterface
{
public function getFooBar();
}

// This class can be unit tested in isolation because class Foo does not need to be loaded by mocking FooInterface and calling the final bar method.
class Baz implements BazInterface
{
private $foo;

public function __construct(FooInterface $foo)
{
$this->foo = $foo;
}

public function getFooBar()
{
return $this->foo->bar();
}
}

$foo = new Foo();
$baz = new Baz($foo);
$bar = $baz->getFooBar();
?>
```

up
down
5

slorenzo at clug dot org dot ve ¶

16 years ago

```
<?php
class parentClass {
public function someMethod() { }
}
class childClass extends parentClass {
public final function someMethod() { } //override parent function
}

$class = new childClass;
$class->someMethod(); //call the override function in chield class
?>
```

up
down
6

Anonymous ¶

13 years ago

The behaviour of FINAL is not as serious as you may think. A little explample:

```
<?php
class A {
final private function method(){}
}

class B extends A {
private function method(){}
}
?>
```

Normally you would expect some of the following will happen:
- An error that final and private keyword cannot be used together
- No error as the private visibility says, that a method/var/etc. is only visible within the same class

But what happens is PHP is a little curios: "Cannot override final method A::method()"

So its possible to deny method names in subclasses! Don't know if this is a good behavior, but maybe its useful for your purpose.

up
down
2

Baldurien ¶

13 years ago

"Note for Java developers: the 'final' keyword is not used for class constants in PHP. We use the keyword 'const'."

http://php.net/manual/en/language.oop5.constants.php

This is more or less true, regardless of the fact that constant (being defined at class level or not) in PHP are only scalar (int, string, etc) while in Java they may be pure object (ex: java.awat.Color.BLACK). The only possible solution of having such kind of constant is :

```
<?php
class Bar {...}
class Foo {
```

```
public static $FOOBAR;

static function __init() {
static $init = false;
if ($init) throw new Exception('Constants were already initialized');
self::$FOOBAR = new Bar();
$init = true;
}
}
Foo::__init();
?>
```

That said, perhaps it is useless unless PHP automatically calls the __init() method.

However, one alternative that could be done in certain case is this :

```
<?php
function __autoload($className) {
... require the file where the class is ...
if (interface_exists($className, false)) return;
if (class_exists($className, false)) {
$rc = new ReflectionClass($className);
if (!$rc->hasMethod('__init')) return;
$m = $rc->getMethod('__init');
if (!($m->isStatic() && $m->isPrivate())) {
throw new Exception($className . ' __init() method must be private and static !');
}
$m->invoke(null);
return;
}
throw new Exception('Class or interface not found ' . $className);
}
?>
```

This can only work when one class is defined per file, since we are assured that __autoload() will be called to load the file containing the class.

eg:

```
test2.php:
<?php
class B {
public static $X;
private static function __init() {
echo 'B', "\n";
self::$X = array(1, 2);
}
}
class A {
public static $Y;
private static function __init() {
echo 'A', "\n";
self::$Y = array(3, 4);
}
}
?>
test.php:
<?php
function __autoload($n) {
if ($n == 'A' || $n == 'B') require 'test2.php';
... do our __init() trick ...
}
var_dump(B::$X); // shows B, then array(2) (1, 2)
var_dump(A::$Y); // shows NULL.
?>
```
up
down
1
Rumour ¶
9 months ago
Class constants CAN be “finalised” since PHP8.1. To partly contradict to the most popular user contribution, that was written a long time ago, they were still absolutely right.
up
down
-6
t at bestcodepractise dot com ¶
9 years ago
@someone
@thomas
Decorating a finalized class is not possible. The decorator that's mentioned is incomplete. There's a fundamental flaw in it. Look:
```
<?php
//copy'n'paste your FooBar, Foo definition

$f = new Foo;
$fd = new FooDecorator($f);

var_dump($fd instanceof $f); //FALSE
var_dump(is_a($fd, 'Foo')); //FALSE
fooFoo($fd); //E_RECOVERABLE_ERROR here !!!;

?>
```
What you've created is just an object that happens to have the same methods(a duck type). But if in the client code someone makes decision based on the type of your passed decorator they'll make incorrect decision - or to be more precise not the one that you, the author of the 'decorator' wants them to make.
FYI that's the correct implementation based on GoF:
```
<?php
class Foo
{
public method doFoo()
{
// do something useful and return a result
}
}

//decorator must inherit the interface(the methods, type info etc.) //of the decorated class by *extending* it.
class FooDecorator extends Foo
{
private $foo;

public function __construct(Foo $foo)
{
$this->foo = $foo;
}

public function doFoo()
{
$result = $this->foo->doFoo();
// ... customize result ...
return $result;
}
}

function fooFoo(Foo $f) {}
$f = new Foo;
```

```
$fd = new FooDecorator($f);

var_dump($fd instanceof $f); //true
fooFoo($fd); //no E_RECOVERABLE_ERROR here;

?>
```

I haven't come across any legitimate use of a finalizing class/method and I personally think that 'final' is has no much use and is just a copy'n'pasted from Java into PHP. The keyword makes code difficult to test. If you have to create a test double from a finalized class because you'll need create a derived type to shadow the methods you don't care about. If one of them is finalized you've already lost.

up
down
-6
xavier dot inghels at gmail dot com ¶
8 years ago

Final is not really about overriding or overloading a method.
PHP doesn't support (yet ?) method ovverriding.

The final keyword however prevent you to redefine a method already previously declared.

up
down
-9
suisuiruyi at gmail dot com ¶
7 years ago

Note: Properties cannot be declared final, only classes and methods may be declared as final.
you can use trait:

```
<?php
trait PropertiesTrait {
public $same = true;
public $different = false;
}

class PropertiesExample {
use PropertiesTrait;
public $same = true; // Strict Standards
public $different = true; // 致命
}
?>
```

up
down
-16
John smith ¶
6 years ago

Right way:

```
final protected function example() {
}
```

Wrong way:

```
protected final function example() {
}
```

Source: Practices

+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