Search

Getting Started
- Introduction
- A simple tutorial

Language Reference
- Basic syntax
- Types
- Variables
- Constants
- Expressions
- Operators
- Control Structures
- Functions
- Classes and Objects
- Namespaces
- Enumerations
- Errors
- Exceptions
- Fibers
- Generators
- Attributes
- References Explained
- Predefined Variables
- Predefined Exceptions
- Predefined Interfaces and Classes
- Predefined Attributes
- Context options and parameters
- Supported Protocols and Wrappers

Security
- Introduction
- General considerations
- Installed as CGI binary
- Installed as an Apache module
- Session Security
- Filesystem Security
- Database Security
- Error Reporting
- User Submitted Data
- Hiding PHP
- Keeping Current

Features
- HTTP authentication with PHP
- Cookies
- Sessions
- Dealing with XForms
- Handling file uploads
- Using remote files
- Connection handling
- Persistent Database Connections
- Command line usage

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

Оператор разрешения области видимости (::) »
« Область видимости

- Руководство по PHP
- Справочник языка
- Классы и объекты

Change language: Russian

Наследование

Наследование — это хорошо зарекомендовавший себя принцип программирования, и PHP использует этот принцип в своей объектной модели. Этот принцип повлияет на то, как многие классы и объекты связаны друг с другом.

Например, при расширении класса дочерний класс наследует все общедоступные и защищённые методы, свойства и константы родительского класса. До тех пор пока эти методы не будут переопределены, они будут сохранять свою исходную функциональность.

Это полезно для определения и абстрагирования функциональности и позволяет реализовать дополнительную функциональность в похожих объектах без необходимости реализовывать всю общую функциональность.

Закрытые методы родительского класса недоступны для дочернего класса. В результате дочерние классы могут повторно реализовать закрытый метод без учёта обычных правил наследования. Однако до PHP 8.0.0 к закрытым методам применялись ограничения `final` и `static`. Начиная с PHP 8.0.0, единственное ограничение закрытого метода, которое применяется - это конструкторы `private final`, поскольку это обычный способ "отключить" конструктор при использовании вместо него статичных фабричных методов.

Видимость методов, свойств и констант можно ослабить, например, `защищённый` метод может быть помечен как `общедоступный`, но нельзя ограничить видимость, например, нельзя пометить `общедоступное` свойство как `закрытое`. Исключением являются конструкторы, видимость которых может быть ограничена, например, `общедоступный` конструктор может быть помечен как `закрытый` в дочернем классе.

> **Замечание**:
>
> Если не используется автозагрузка, классы должны быть объявлены до того, как они будут использоваться. Если класс расширяет другой, то родительский класс должен быть объявлен до наследующего класса. Это правило применяется к классам, которые наследуют другие классы или интерфейсы.

> **Замечание**:
>
> Не разрешается переопределять свойство чтения-записи с помощью readonly-свойства или наоборот.

```php
<?php
class A {
public int $prop;
}
class B extends A {
// Нельзя: read-write -> readonly
public readonly int $prop;
}
?>
```

Пример #1 Пример наследования

```php
<?php

class Foo
{
public function printItem($string)
{
echo 'Foo: ' . $string . PHP_EOL;
}

public function printPHP()
{
echo 'PHP просто супер.' . PHP_EOL;
}
}

class Bar extends Foo
{
public function printItem($string)
```

```
{
echo 'Bar: ' . $string . PHP_EOL;
}
}

$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Выведет: 'Foo: baz'
$foo->printPHP(); // Выведет: 'PHP просто супер'
$bar->printItem('baz'); // Выведет: 'Bar: baz'
$bar->printPHP(); // Выведет: 'PHP просто супер'

?>
```

Совместимость типов возвращаемых значений с внутренними классами

До PHP 8.1.0 большинство внутренних классов или методов не объявляли свои типы возвращаемых значений и при их расширении допускался любой тип возвращаемого значения.

Начиная с PHP 8.1.0, большинство внутренних методов начали "предварительно" объявлять тип возвращаемого значения. В этом случае тип возвращаемого значения методов должен быть совместим с расширяемым родителем; в противном случае выдаётся уведомление об устаревании. Обратите внимание, что отсутствие явного объявления типа возвращаемого значения также считается несоответствием сигнатуры и, соответственно, приводит к уведомлению об устаревании.

Если тип возвращаемого значения не может быть объявлен для переопределяемого метода из-за проблем с совместимостью с различными версиями PHP, может быть добавлен атрибут ReturnTypeWillChange, чтобы заглушить уведомление об устаревании.

Пример #2 Переопределяющий метод не объявляет никакого типа возвращаемого значения

```
<?php
class MyDateTime extends DateTime
{
public function modify(string $modifier) { return false; }
}

// "Deprecated: Return type of MyDateTime::modify(string $modifier) should either be compatible with
DateTime::modify(string $modifier): DateTime|false, or the #[\ReturnTypeWillChange] attribute should be used to
temporarily suppress the notice", начиная с PHP 8.1.0
?>
```

Пример #3 Переопределяющий метод объявляет неверный тип возвращаемого значения

```
<?php
class MyDateTime extends DateTime
{
public function modify(string $modifier): ?DateTime { return null; }
}

// "Deprecated: Return type of MyDateTime::modify(string $modifier): ?DateTime should either be compatible with
DateTime::modify(string $modifier): DateTime|false, or the #[\ReturnTypeWillChange] attribute should be used to
temporarily suppress the notice", начиная с PHP 8.1.0
?>
```

Пример #4 Переопределяющий метод объявляет неверный тип возвращаемого значения без уведомления об устаревании

```
<?php
class MyDateTime extends DateTime
{
/**
* @return DateTime|false
*/
```

```php
#[\ReturnTypeWillChange]
public function modify(string $modifier) { return false; }
}

// Уведомление об устаревании не выводится
?>
```


User Contributed Notes 8 notes

210
jackdracona at msn dot com ¶
13 years ago

Here is some clarification about PHP inheritance - there is a lot of bad information on the net. PHP does support Multi-level inheritance. (I tested it using version 5.2.9). It does not support multiple inheritance.

This means that you cannot have one class extend 2 other classes (see the extends keyword). However, you can have one class extend another, which extends another, and so on.

Example:

```php
<?php
class A {
// more code here
}

class B extends A {
// more code here
}

class C extends B {
// more code here
}



$someObj = new A(); // no problems
$someOtherObj = new B(); // no problems
$lastObj = new C(); // still no problems

?>
```


100
Mohammad Istanbouly ¶
6 years ago

I think the best way for beginners to understand inheritance is through a real example so here is a simple example I can gave to you

```php
<?php

class Person
{
public $name;
protected $age;
private $phone;

public function talk(){
//Do stuff here
}
```

```
protected function walk(){
//Do stuff here
}

private function swim(){
//Do stuff here
}
}

class Tom extends Person
{
/*Since Tom class extends Person class this means
that class Tom is a child class and class person is
the parent class and child class will inherit all public
and protected members(properties and methods) from
the parent class*/

/*So class Tom will have these properties and methods*/

//public $name;
//protected $age;
//public function talk(){}
//protected function walk(){}

//but it will not inherit the private members
//this is all what Object inheritance means
}
```

up
down
23
strata_ranger at hotmail dot com ¶
13 years ago

I was recently extending a PEAR class when I encountered a situation where I wanted to call a constructor two levels up the class hierarchy, ignoring the immediate parent. In such a case, you need to explicitly reference the class name using the :: operator.

Fortunately, just like using the 'parent' keyword PHP correctly recognizes that you are calling the function from a protected context inside the object's class hierarchy.

E.g:

```
<?php
class foo
{
public function something()
{
echo __CLASS__; // foo
var_dump($this);
}
}

class foo_bar extends foo
{
public function something()
{
echo __CLASS__; // foo_bar
var_dump($this);
}
}

class foo_bar_baz extends foo_bar
{
```

```
public function something()
{
echo __CLASS__; // foo_bar_baz
var_dump($this);
}

public function call()
{
echo self::something(); // self
echo parent::something(); // parent
echo foo::something(); // grandparent
}
}

error_reporting(-1);

$obj = new foo_bar_baz();
$obj->call();

// Output similar to:
// foo_bar_baz
// object(foo_bar_baz)[1]
// foo_bar
// object(foo_bar_baz)[1]
// foo
// object(foo_bar_baz)[1]

?>
```

up
down
15

akashwebdev at gmail dot com ¶

8 years ago

The Idea that multiple inheritence is not supported is correct but with tratits this can be reviewed.

for e.g.

```
<?php
trait custom
{
public function hello()
{
echo "hello";
}
}

trait custom2
{
public function hello()
{
echo "hello2";
}
}

class inheritsCustom
{
use custom, custom2
{
custom2::hello insteadof custom;
}
}
```

```
$obj = new inheritsCustom();
$obj->hello();
?>
```

up
down
11

jarrod at squarecrow dot com ¶

14 years ago

You can force a class to be strictly an inheritable class by using the "abstract" keyword. When you define a class with abstract, any attempt to instantiate a separate instance of it will result in a fatal error. This is useful for situations like a base class where it would be inherited by multiple child classes yet you want to restrict the ability to instantiate it by itself.

Example........

```
<?php

abstract class Cheese
{
//can ONLY be inherited by another class
}

class Cheddar extends Cheese
{
}

$dinner = new Cheese; //fatal error
$lunch = new Cheddar; //works!

?>
```

up
down
-3

niemans at pbsolo dot nl ¶

4 years ago

Inheritance works at create time, i.e. using the keyword 'new'. Static properties confused my understanding, so in order tho show the effect of visibility to inherintence I've created a simple demo script along with some set and get magic:

```
<?php
class A {
private $a = 'private';
protected $b = 'protected';
public $c = 'public';
static $d = 'static';
public function __construct()
{
$this->e = 'constructed';
}
public function __set($property, $value)
{
echo ' set ' . $property . '=' . $value;
$this->$property=$value;
}
public function __get($property)
{
echo ' get ' . $property;
$this->$property = 'dynamic'; // invokes __set() !!
return $this->$property;
}
}

class B extends A
```

```
{
public function constructMe()
{
$this->e = 'constructed2';
}
}

class C extends B
{
public function __construct()
{
parent::constructMe();
}
}

echo " \n";
$a = new A();
$b = new B();
echo " \n";
echo ' B:c='.$b->c;
echo " \n";
echo ' B:d=' .$b->d;
echo " \n";

$c = new C();
echo " \n";

print_r($a);
print_r($b);
print_r($c);

print_r(A::$d);
print_r(B::$d);
print_r(C::$d);

echo 'A class: ';
$R = new reflectionclass('A');
print_r($R->getdefaultproperties());
print_r($R->getstaticproperties());
echo 'B class: ';
$R = new reflectionclass('B');
print_r($R->getdefaultproperties());
print_r($R->getstaticproperties());

?>
```

This outputs:

```
set e=constructed
B:c=public
get d set d=dynamic B:d=dynamic
set e=constructed2
A Object
(
[a:A:private] => private
[b:protected] => protected
[c] => public
[e] => constructed
)
B Object
(
[a:A:private] => private
```

```
[b:protected] => protected
[c] => public
[d] => dynamic
)
C Object
(
[a:A:private] => private
[b:protected] => protected
[c] => public
[e] => constructed2
)
staticstaticstaticA class: Array
(
[d] => static
[a] => private
[b] => protected
[c] => public
)
Array
(
[d] => static
)
B class: Array
(
[d] => static
[b] => protected
[c] => public
)
Array
(
[d] => static
)
```

This shows how private variables ($a) are inherited, how static variables ($d) are inherited (by the class, not by the object) and that changing or adding variables in the parent ($e, $d) are not inherited by the child.

up
down
-5

Anonymous ¶

5 years ago

PHP7 gives you a warning if you redeclare a function in a child class with different parameters. For example:

```
class foo {
function print($text='') {
print text;
}
}

class bar extends foo {
function print($text1='',$text2='') {
print text1.text2
}
}
```

will give a PHP Warning: Declaration of bar::print($text1 = '', $text2 = '') should be compatible with foo::print($text='').

up
down
-4

sibian0218 at gmail dot com ¶

5 years ago

I've noticed one thing concerning inheritance...

When declaring an abstract class with a private method,
which is overridden by a sub-class, private takes precedence over public for child class...
(in the case you're redeclaring a method with a different signature in fact).

Hope this helps

- 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