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

Наследование »
« Конструкторы и деструкторы

- Руководство по PHP
- Справочник языка
- Классы и объекты

Change language: Russian

Область видимости

Область видимости свойства, метода или константы (начиная с PHP 7.1.0) определяют, добавляя перед объявлением ключевое слово: `public`, `protected` или `private`. Доступ к свойствам и методам класса, объявленным как public (общедоступный), разрешён отовсюду. Модификатор protected (защищённый) разрешает доступ самому классу, наследующим его классам и родительским классам. Модификатор private (закрытый) ограничивает область видимости так, что только класс, где объявлен сам элемент, имеет к нему доступ.

Область видимости свойства

Свойства класса могут быть определены как public, private или protected. Свойства, объявленные без явного ключевого слова области видимости, определяются как общедоступные (public).

Пример #1 Объявление свойства класса

```
<?php
/**
* Определение MyClass
*/
class MyClass
{
public $public = 'Public';
protected $protected = 'Protected';
private $private = 'Private';

function printHello()
{
echo $this->public;
echo $this->protected;
echo $this->private;
}
}

$obj = new MyClass();
echo $obj->public; // Работает
echo $obj->protected; // Неисправимая ошибка
echo $obj->private; // Неисправимая ошибка
$obj->printHello(); // Выводит Public, Protected и Private



/**
* Определение MyClass2
*/
class MyClass2 extends MyClass
{
// Мы можем переопределить общедоступные и защищённые свойства, но не закрытые
public $public = 'Public2';
protected $protected = 'Protected2';

function printHello()
{
echo $this->public;
echo $this->protected;
echo $this->private;
}
}

$obj2 = new MyClass2();
echo $obj2->public; // Работает
echo $obj2->private; // Неопределён
```

```
echo $obj2->protected; // Неисправимая ошибка
$obj2->printHello(); // Выводит Public2, Protected2, Undefined

?>
```

Область видимости метода

Методы класса могут быть определены как public, private или protected. Методы, объявленные без указания области видимости, определяются как public.

Пример #2 Объявление метода

```
<?php
/**
* Определение MyClass
*/
class MyClass
{
// Объявление общедоступного конструктора
public function __construct() { }

// Объявление общедоступного метода
public function MyPublic() { }

// Объявление защищённого метода
protected function MyProtected() { }

// Объявление закрытого метода
private function MyPrivate() { }

// Это общедоступный метод
function Foo()
{
$this->MyPublic();
$this->MyProtected();
$this->MyPrivate();
}
}

$myclass = new MyClass;
$myclass->MyPublic(); // Работает
$myclass->MyProtected(); // Неисправимая ошибка
$myclass->MyPrivate(); // Неисправимая ошибка
$myclass->Foo(); // Работает общедоступный, защищённый и закрытый


/**
* Определение MyClass2
*/
class MyClass2 extends MyClass
{
// Это общедоступный метод
function Foo2()
{
$this->MyPublic();
$this->MyProtected();
$this->MyPrivate(); // Неисправимая ошибка
}
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Работает
```

```php
$myclass2->Foo2(); // Работает общедоступный и защищённый, закрытый не работает

class Bar
{
public function test() {
$this->testPrivate();
$this->testPublic();
}

public function testPublic() {
echo "Bar::testPublic\n";
}

private function testPrivate() {
echo "Bar::testPrivate\n";
}
}

class Foo extends Bar
{
public function testPublic() {
echo "Foo::testPublic\n";
}

private function testPrivate() {
echo "Foo::testPrivate\n";
}
}

$myFoo = new Foo();
$myFoo->test(); // Bar::testPrivate
// Foo::testPublic
?>
```

Область видимости констант

Начиная с PHP 7.1.0, константы класса могут быть определены как public, private или protected. Константы, объявленные без указания области видимости, определяются как public.

Пример #3 Объявление констант, начиная с PHP 7.1.0

```php
<?php
/**
* Объявление класса MyClass
*/
class MyClass
{
// Объявление общедоступной константы
public const MY_PUBLIC = 'public';

// Объявление защищённой константы
protected const MY_PROTECTED = 'protected';

// Объявление закрытой константы
private const MY_PRIVATE = 'private';

public function foo()
{
echo self::MY_PUBLIC;
echo self::MY_PROTECTED;
echo self::MY_PRIVATE;
}
```

```
}

$myclass = new MyClass();
MyClass::MY_PUBLIC; // Работает
MyClass::MY_PROTECTED; // Неисправимая ошибка
MyClass::MY_PRIVATE; // Неисправимая ошибка
$myclass->foo(); // Выводятся константы public, protected и private



/**
* Объявление класса MyClass2
*/
class MyClass2 extends MyClass
{
// Публичный метод
function foo2()
{
echo self::MY_PUBLIC;
echo self::MY_PROTECTED;
echo self::MY_PRIVATE; // Неисправимая ошибка
}
}

$myclass2 = new MyClass2;
echo MyClass2::MY_PUBLIC; // Работает
$myclass2->foo2(); // Выводятся константы public и protected, но не private
?>
```

Видимость из других объектов

Объекты, которые имеют общий тип (наследуются от одного класса), имеют доступ к элементам с модификаторами private и protected друг друга, даже если не являются одним и тем же экземпляром. Это объясняется тем, что реализация видимости элементов известна внутри этих объектов.

Пример #4 Доступ к элементам с модификатором private из объектов одного типа

```
<?php
class Test
{
private $foo;

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

private function bar()
{
echo 'Доступ к закрытому методу.';
}

public function baz(Test $other)
{
// Мы можем изменить закрытое свойство:
$other->foo = 'привет';
var_dump($other->foo);

// Мы также можем вызвать закрытый метод:
$other->bar();
}
}
```

```php
$test = new Test('test');

$test->baz(new Test('other'));
?>
```

Результат выполнения приведённого примера:

```
string(6) "привет"
Доступ к закрытому методу.
```


User Contributed Notes 7 notes

60
pgl at yoyo dot org ¶
8 years ago
Just a quick note that it's possible to declare visibility for multiple properties at the same time, by separating them by commas.

eg:

```php
<?php
class a
{
protected $a, $b;

public $c, $d;

private $e, $f;
}
?>
```


19
r dot wilczek at web-appz dot de ¶
18 years ago
Beware: Visibility works on a per-class-base and does not prevent instances of the same class accessing each others properties!

```php
<?php
class Foo
{
private $bar;

public function debugBar(Foo $object)
{
// this does NOT violate visibility although $bar is private
echo $object->bar, "\n";
}

public function setBar($value)
{
// Neccessary method, for $bar is invisible outside the class
$this->bar = $value;
}

public function setForeignBar(Foo $object, $value)
{
// this does NOT violate visibility!
$object->bar = $value;
```

```
}
}

$a = new Foo();
$b = new Foo();
$a->setBar(1);
$b->setBar(2);
$a->debugBar($b); // 2
$b->debugBar($a); // 1
$a->setForeignBar($b, 3);
$b->setForeignBar($a, 4);
$a->debugBar($b); // 3
$b->debugBar($a); // 4
?>
```

up
down
1
alperenberatdurmus at gmail dot com ¶
9 months ago

```
Dynamic properties are "public".
<?php
class MyClass {
public function setProperty($value) {
$this->dynamicProperty = $value;
}
}
$obj = new MyClass();
$obj->setProperty('Hello World');
echo $obj->dynamicProperty; // Outputs "Hello World"
?>

This usage is the same as well:
<?php
class MyClass {
}
$obj = new MyClass();
$obj->dynamicProperty = 'Hello World';
echo $obj->dynamicProperty; // Outputs "Hello World"
?>
```

up
down
6
jc dot flash at gmail dot com ¶
11 years ago

```
if not overwritten, self::$foo in a subclass actually refers to parent's self::$foo
<?php
class one
{
protected static $foo = "bar";
public function change_foo($value)
{
self::$foo = $value;
}
}

class two extends one
{
public function tell_me()
{
echo self::$foo;
}
}
```

```
$first = new one;
$second = new two;

$second->tell_me(); // bar
$first->change_foo("restaurant");
$second->tell_me(); // restaurant
?>
```

up
down
5

Joshua Watt ¶

16 years ago

I couldn't find this documented anywhere, but you can access protected and private member varaibles in different instance of the same class, just as you would expect

i.e.

```
<?php
class A
{
protected $prot;
private $priv;

public function __construct($a, $b)
{
$this->prot = $a;
$this->priv = $b;
}

public function print_other(A $other)
{
echo $other->prot;
echo $other->priv;
}
}

class B extends A
{
}

$a = new A("a_protected", "a_private");
$other_a = new A("other_a_protected", "other_a_private");

$b = new B("b_protected", "ba_private");

$other_a->print_other($a); //echoes a_protected and a_private
$other_a->print_other($b); //echoes b_protected and ba_private

$b->print_other($a); //echoes a_protected and a_private
?>
```

up
down
2

bishop at php dot net ¶

7 years ago

> Members declared protected can be accessed only within
> the class itself and by inherited classes. Members declared
> as private may only be accessed by the class that defines
> the member.

This is not strictly true. Code outside the object can get and set private and protected members:

```
<?php
class Sealed { private $value = 'foo'; }

$sealed = new Sealed;
var_dump($sealed); // private $value => string(3) "foo"

call_user_func(\Closure::bind(
function () use ($sealed) { $sealed->value = 'BAZ'; },
null,
$sealed
));

var_dump($sealed); // private $value => string(3) "BAZ"

?>
```

The magic lay in \Closure::bind, which allows an anonymous function to bind to a particular class scope. The documentation on \Closure::bind says:

> If an object is given, the type of the object will be used
> instead. This determines the visibility of protected and
> private methods of the bound object.

So, effectively, we're adding a run-time setter to $sealed, then calling that setter. This can be elaborated to generic functions that can force set and force get object members:

```
<?php
function force_set($object, $property, $value) {
call_user_func(\Closure::bind(
function () use ($object, $property, $value) {
$object->{$property} = $value;
},
null,
$object
));
}

function force_get($object, $property) {
return call_user_func(\Closure::bind(
function () use ($object, $property) {
return $object->{$property};
},
null,
$object
));
}

force_set($sealed, 'value', 'quux');
var_dump(force_get($sealed, 'value')); // 'quux'

?>
```

You should probably not rely on this ability for production quality code, but having this ability for debugging and testing is handy.

up
down
0

kostya at eltexsoft dot com ¶

2 years ago

I see we can redeclare private properties into child class

```
<?php
class A{
```

```
private int $private_prop = 4;
protected int $protected_prop = 8;
}

class B extends A{
private int $private_prop = 7; // we can redeclare private property!!!
public function printAll() {
echo $this->private_prop;
echo $this->protected_prop;
}
}

$b = new B;
$b->printAll(); // show 78
}
?>
```