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

Классы и объекты »
« Стрелочные функции

- Руководство по PHP
- Справочник языка
- Функции

Change language: Russian

Синтаксис callable-объектов первого класса

Синтаксис callable-функций как объектов первого класса представили в PHP 8.1.0 как способ, которым создают анонимные функции из callable-объектов. Он заменяет существующий синтаксис вызываемых объектов со строками и массивами. Преимущество синтаксиса состоит в том, что он доступен для статического анализа и использует область видимости точки, в которой callable-объект был получен.

Синтаксис `CallableExpr(...)` создаёт объект Closure из callable-объекта. Часть `CallableExpr` принимает любое выражение, которое можно непосредственно вызвать в грамматике PHP:

Пример #1 Простой пример синтаксиса callable-объекта первого класса

```
<?php

class Foo {
public function method() {}
public static function staticmethod() {}
public function __invoke() {}
}
$obj = new Foo();
$classStr = 'Foo';
$methodStr = 'method';
$staticmethodStr = 'staticmethod';
$f1 = strlen(...);
$f2 = $obj(...); // Вызываемый объект
$f3 = $obj->method(...);
$f4 = $obj->$methodStr(...);
$f5 = Foo::staticmethod(...);
$f6 = $classStr::$staticmethodStr(...);
// Традиционный callable-синтаксис со строками и массивами
$f7 = 'strlen'(...);
$f8 = [$obj, 'method'](...);
$f9 = [Foo::class, 'staticmethod'](...);

?>
```

> **Замечание**:
>
> Многоточие `...` — часть синтаксиса, а не пропуск.

У выражения `CallableExpr(...)` та же семантика, что и у метода Closure::fromCallable(). То есть, в отличие от callable-синтаксиса со строками и массивами, синтаксис `CallableExpr(...)` учитывает область видимости в той точке, в которой он создан:

Пример #2 Сравнение области действия синтаксиса `CallableExpr(...)` и традиционного callable-синтаксиса

```
<?php

class Foo {
public function getPrivateMethod() {
return [$this, 'privateMethod'];
}
private function privateMethod() {
echo __METHOD__, "\n";
}
}
$foo = new Foo;
$privateMethod = $foo->getPrivateMethod();
$privateMethod();
// Fatal error: Call to private method Foo::privateMethod() from global scope
// Причина фатальной ошибки в том, что вызов выполнен за пределами класса Foo, и с этого момента будет проверяться
видимость.
```

```php
class Foo1 {
public function getPrivateMethod() {
// Использует область видимости, в которой получен callable-объект.
return $this->privateMethod(...); // Идентично вызову Closure::fromCallable([$this, 'privateMethod']);
}
private function privateMethod() {
echo __METHOD__, "\n";
}
}
$foo1 = new Foo1;
$privateMethod = $foo1->getPrivateMethod();
$privateMethod(); // Foo1::privateMethod

?>
```

> **Замечание**:
>
> Создание объекта этим синтаксисом (например, `new Foo(...)`) не поддерживается, поскольку синтаксис `new Foo()` не признаётся вызовом.

> **Замечание**:
>
> Синтаксис callable-объектов первого класса нельзя комбинировать с оператором Nullsafe. Оба следующих результата приводят к ошибке времени компиляции:
>
> ```php
> <?php
>
> $obj?->method(...);
> $obj?->prop->method(...);
>
> ?>
> ```


User Contributed Notes 1 note

0
bienvenunet at yahoo dot com ¶
8 months ago

There's a major gotcha with this syntax that may not be apparent until you use this syntax and find you're getting "Cannot rebind scope of closure created from method" exceptions in some random library code.

As the documentation indicates, the first-class callable uses the scope at the point where the callable is acquired. This is fine as long as nothing in your code will attempt to bind the callable with the \Closure::bindTo method.

I found this the hard way by changing callables going to Laravel's Macroable functionality from the array style to the first-class callable style. The Macroable functionality \Closure::bindTo calls on the callable.

AFAIK, the only workaround is to use the uglier array syntax.