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

Mixed »
« Ресурсы

Change language: Russian

Callable и callback-функции

Callback-функции разрешено обозначать объявлением типа callable.

Функции вроде call_user_func() или usort() принимают определённые пользователем callback-функции в качестве параметра. Callback-функциями бывают как простые функции, так и методы объектов, включая статические методы классов.

Передача

PHP-функция передаётся по имени в виде строки. Разрешено передавать любую встроенную или пользовательскую функцию, исключая языковые конструкции: array(), echo, empty(), eval(), exit(), isset(), list(), print или unset().

Метод созданного объекта (object) передаётся как массив, который в индексе 0 содержит — объект, а в индексе 1 — название метода. Изнутри класса разрешён доступ к закрытым и защищённым методам.

Статические методы класса тоже передаются — или как массив, в индексе 0 которого вместо передачи объекта указывают название класса, или как строка вида `'ClassName::methodName'`.

Кроме пользовательских функций, в callback-параметр разрешено передавать анонимные и стрелочные функции.

> **Замечание**:
>
> Начиная с PHP 8.1.0 анонимные функции разрешено создавать синтаксисом callable-объектов первого класса.

Объекты, которые реализуют магический метод __invoke(), разрешно передавать в callback-параметр.

Пример #1 Пример callback-функции

```
<?php

// Пример callback-функции
function my_callback_function() {
echo 'Привет, мир!';
}

// Пример callback-метода
class MyClass {
static function myCallbackMethod() {
echo 'Привет, мир!';
}
}

// Тип 1: Простой callback
call_user_func('my_callback_function');

// Тип 2: Вызов статического метода класса
call_user_func(array('MyClass', 'myCallbackMethod'));

// Тип 3: Вызов метода класса
$obj = new MyClass();
call_user_func(array($obj, 'myCallbackMethod'));

// Тип 4: Вызов статического метода класса
call_user_func('MyClass::myCallbackMethod');

// Тип 5: Вызов относительного статического метода
class A {
public static function who() {
echo "A\n";
}
}

class B extends A {
public static function who() {
echo "B\n";
}
}

call_user_func(array('B', 'parent::who')); // A, устарело, начиная с PHP 8.2.0

// Тип 6: Объекты, которые реализуют магический метод __invoke, разрешено использовать как callable-объекты
class C {
public function __invoke($name) {
echo 'Привет ', $name, "\n";
}
}

$c = new C();
call_user_func($c, 'PHP!');

?>
```

Пример #2 Пример передачи замыкания в callback-параметр

```
<?php

// Замыкание
$double = function($a) {
return $a * 2;
};

// Диапазон чисел
$numbers = range(1, 5);

// Передаём в параметр замыкание в качестве callback-функции
// для удвоения каждого элемента в созданном выше диапазоне
$new_numbers = array_map($double, $numbers);

print implode(' ', $new_numbers);

?>
```

Результат выполнения приведённого примера:

```
2 4 6 8 10
```

> **Замечание**:
>
> Callback-функции, зарегистрированные такими функциями как call_user_func() и call_user_func_array(), не будут вызваны при наличии не пойманного исключения, брошенного в предыдущей callback-функции.

+ add a note

User Contributed Notes 12 notes

up
down
272
andrewbessa at gmail dot com ¶
11 years ago

```
You can also use the $this variable to specify a callback:

<?php
class MyClass {
```

```
public $property = 'Hello World!';

public function MyMethod()
{
call_user_func(array($this, 'myCallbackMethod'));
}

public function MyCallbackMethod()
{
echo $this->property;
}

}
?>
```
up
down
192
Riikka K ¶
8 years ago
A note on differences when calling callbacks as "variable functions" without the use of call_user_func() (e.g. "<?php $callback = 'printf'; $callback('Hello World!') ?>"):

- Using the name of a function as string has worked since at least 4.3.0
- Calling anonymous functions and invokable objects has worked since 5.3.0
- Using the array structure [$object, 'method'] has worked since 5.4.0

Note, however, that the following are not supported when calling callbacks as variable functions, even though they are supported by call_user_func():

- Calling static class methods via strings such as 'foo::doStuff'
- Calling parent method using the [$object, 'parent::method'] array structure

All of these cases are correctly recognized as callbacks by the 'callable' type hint, however. Thus, the following code will produce an error "Fatal error: Call to undefined function foo::doStuff() in /tmp/code.php on line 4":

```
<?php
class foo {
static function callIt(callable $callback) {
$callback();
}

static function doStuff() {
echo "Hello World!";
}
}

foo::callIt('foo::doStuff');
?>
```

The code would work fine, if we replaced the '$callback()' with 'call_user_func($callback)' or if we used the array ['foo', 'doStuff'] as the callback instead.
up
down
193
computrius at gmail dot com ¶
10 years ago
When specifying a call back in array notation (ie. array($this, "myfunc")) the method can be private if called from inside the class, but if you call it from outside you'll get a warning:

```
<?php
```

```
class mc {
public function go(array $arr) {
array_walk($arr, array($this, "walkIt"));
}

private function walkIt($val) {
echo $val . "<br />";
}

public function export() {
return array($this, 'walkIt');
}
}

$data = array(1,2,3,4);

$m = new mc;
$m->go($data); // valid

array_walk($data, $m->export()); // will generate warning

?>

Output:
1<br />2<br />3<br />4<br />
Warning: array_walk() expects parameter 2 to be a valid callback, cannot access private method mc::walkIt() in /in/tfh7f
on line 22
```

up
down
208
steve at mrclay dot org ¶
11 years ago

```
Performance note: The callable type hint, like is_callable(), will trigger an autoload of the class if the value looks
like a static method callback.
```

up
down
191
edanschwartz at gmail dot com ¶
9 years ago

```
You can use 'self::methodName' as a callable, but this is dangerous. Consider this example:

<?php
class Foo {
public static function doAwesomeThings() {
FunctionCaller::callIt('self::someAwesomeMethod');
}

public static function someAwesomeMethod() {
// fantastic code goes here.
}
}

class FunctionCaller {
public static function callIt(callable $func) {
call_user_func($func);
}
}

Foo::doAwesomeThings();
?>

This results in an error:
```

```
Warning: class 'FunctionCaller' does not have a method 'someAwesomeMethod'.

For this reason you should always use the full class name:
<?php
FunctionCaller::callIt('Foo::someAwesomeMethod');
?>

I believe this is because there is no way for FunctionCaller to know that the string 'self' at one point referred to to
`Foo`.
```

up
down
174

metamarkers at gmail dot com ¶

10 years ago

```
you can pass an object as a callable if its class defines the __invoke() magic method..
```

up
down
115

mariano dot REMOVE dot perez dot rodriguez at gmail dot com ¶

8 years ago

```
I needed a function that would determine the type of callable being passed, and, eventually,
normalized it to some extent. Here's what I came up with:

<?php

/**
* The callable types and normalizations are given in the table below:
*
* Callable | Normalization | Type
* ---------------------------------+---------------------------------+--------------
* function (...) use (...) {...} | function (...) use (...) {...} | 'closure'
* $object | $object | 'invocable'
* "function" | "function" | 'function'
* "class::method" | ["class", "method"] | 'static'
* ["class", "parent::method"] | ["parent of class", "method"] | 'static'
* ["class", "self::method"] | ["class", "method"] | 'static'
* ["class", "method"] | ["class", "method"] | 'static'
* [$object, "parent::method"] | [$object, "parent::method"] | 'object'
* [$object, "self::method"] | [$object, "method"] | 'object'
* [$object, "method"] | [$object, "method"] | 'object'
* ---------------------------------+---------------------------------+--------------
* other callable | idem | 'unknown'
* ---------------------------------+---------------------------------+--------------
* not a callable | null | false
*
* If the "strict" parameter is set to true, additional checks are
* performed, in particular:
* - when a callable string of the form "class::method" or a callable array
* of the form ["class", "method"] is given, the method must be a static one,
* - when a callable array of the form [$object, "method"] is given, the
* method must be a non-static one.
*
*/
function callableType($callable, $strict = true, callable& $norm = null) {
if (!is_callable($callable)) {
switch (true) {
case is_object($callable):
$norm = $callable;
return 'Closure' === get_class($callable) ? 'closure' : 'invocable';
case is_string($callable):
$m = null;
if (preg_match('~^(?<class>[a-z_][a-z0-9_]*)::(?<method>[a-z_][a-z0-9_]*)$~i', $callable, $m)) {
```

```
list($left, $right) = [$m['class'], $m['method']];
if (!$strict || (new \ReflectionMethod($left, $right))->isStatic()) {
$norm = [$left, $right];
return 'static';
}
} else {
$norm = $callable;
return 'function';
}
break;
case is_array($callable):
$m = null;
if (preg_match('~^(:?(?<reference>self|parent)::)?(?<method>[a-z_][a-z0-9_]*)$~i', $callable[1], $m)) {
if (is_string($callable[0])) {
if ('parent' === strtolower($m['reference'])) {
list($left, $right) = [get_parent_class($callable[0]), $m['method']];
} else {
list($left, $right) = [$callable[0], $m['method']];
}
if (!$strict || (new \ReflectionMethod($left, $right))->isStatic()) {
$norm = [$left, $right];
return 'static';
}
} else {
if ('self' === strtolower($m['reference'])) {
list($left, $right) = [$callable[0], $m['method']];
} else {
list($left, $right) = $callable;
}
if (!$strict || !(new \ReflectionMethod($left, $right))->isStatic()) {
$norm = [$left, $right];
return 'object';
}
}
}
break;
}
$norm = $callable;
return 'unknown';
}
$norm = null;
return false;
}

?>
```

Hope someone else finds it useful.

up

down

11

***InvisibleSmiley* ¶**

2 years ago

If you pass a callable method to a function with a callable type declaration, the error message is misleading:

```
<?php
class X {
protected function foo(): void {}
}

function bar(callable $c) {}

$x = new X;
```

```
$c = [$x, 'foo'];
bar($c);
?>
```

Error message will be something like "Argument #1 ($c) must be of type callable, array given" while the actual problem here is only the visibility of method "foo". All you need to do is changing it to public (or use a different approach, e.g. with a Closure).

up

down

9

gulaschsuppe2 at gmail dot com ¶

4 years ago

I tried many possible ways of calling functions by function name directly and assigned to a variable on 3v4l. Not mentioned yet, it is possible to use an array as a caller, at least since PHP 7.1.25. The following script contains all the information I gained:

```
<?php

// Call function via function name:
// Basics:
// A function can also be called by using its string name:
function callbackFunc() {
echo 'Hello World';
}

'callbackFunc'(); // Hello World

// A function can also be called if its name is assigned to a variable:
function callbackFunc() {
echo 'Hello World';
}

$funcName = 'callbackFunc';
$funcName(); // Hello World

// Static class method:
// It is also possible to call a public static class method via 'ClassName::functioName' notation:
class A {
public static function callbackMethod() {
echo "Hello World\n";
}
}
'A::callbackMethod'(); // Hello World

$funcName = 'A::callbackMethod';
$funcName(); // Hello World

// Non static class method:
// It is also possible to call non static class methods by creating an array which first element is the object the method should be called on and the second element is the non static method to be called. The array can directly be used as a caller:
class A {
private $prop = "Hello World\n";

public function callbackMethod() {
echo $this->prop;
}
}

$a = new A;
[$a, 'callbackMethod']();
$funcCallArr = [$a, 'callbackMethod'];
```

```
$funcCallArr();

// Of course this also works inside the class with '$this':
class A {
private function privCallback() {
echo 'Private';
}

public function privCallbackCaller($funcName) {
[$this, $funcName]();
}
}

(new A)->privCallbackCaller('privCallback'); // Private

?>
```
up
down
24
bradyn at NOSPAM dot bradynpoulsen dot com ¶
7 years ago

```
When trying to make a callable from a function name located in a namespace, you MUST give the fully qualified function
name (regardless of the current namespace or use statements).

<?php

namespace MyNamespace;

function doSomethingFancy($arg1)
{
// do something...
}

$values = [1, 2, 3];

array_map('doSomethingFancy', $values);
// array_map() expects parameter 1 to be a valid callback, function 'doSomethingFancy' not found or invalid function name

array_map('MyNamespace\doSomethingFancy', $values);
// => [..., ..., ...]
```
up
down
4
pawel dot tadeusz dot niedzielski at gmail dot com ¶
7 years ago

```
@edanschwartz at gmail dot com

You can use ::class property to always indicate the class you're in when using static methods:

<?php
class Foo {
public static function doAwesomeThings() {
FunctionCaller::callIt(self::class . '::someAwesomeMethod');
}

public static function someAwesomeMethod() {
// fantastic code goes here.
}
}

class FunctionCaller {
public static function callIt(callable $func) {
```

```
call_user_func($func);
}
}

Foo::doAwesomeThings();
?>
```
up
down
2
chris dot rutledge at gmail dot com ¶
5 years ago

```
Having read this line in the manual above,

"A method of an instantiated object is passed as an array containing an object at index 0 and the method name at index 1.
Accessing protected and private methods from within a class is allowed."

I decided to do some testing to see if I could access private methods using the call_user_func methods. Thankfully not,
but for completeness here is my test which also covers using static and object contexts

<?php
class foo {

public static $isInstance = false;

public function __construct() {
self::$isInstance = true;
}

public function bar() {
var_dump(self::$isInstance);
echo __METHOD__;
}

private function baz() {
var_dump(self::$isInstance);
echo __METHOD__;
}

public function qux() {
$this->baz();
}

public function quux() {
self::baz();
}
}

call_user_func(['foo','bar']); //fase, foo:bar

call_user_func(['foo','baz']); //warning, cannot access private method

call_user_func(['foo','quux']); //false, foo::baz

call_user_func(['foo','qux']); //fatal, Using $this when not in object context

$foo = new foo;

call_user_func([$foo,'bar']); //true, foo::bar
call_user_func([$foo,'baz']); //warning, cannot access private method
call_user_func([$foo,'qux']); //true, foo::baz

call_user_func(['foo','bar']); //true, foo::bar (static call, yet $isInstance is true)
```

```
?>
```

+add a note