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

Синтаксис callable-объектов первого класса »
« Анонимные функции

- Руководство по PHP
- Справочник языка
- Функции

Change language: Russian

Стрелочные функции

Стрелочные функции появились в PHP 7.4 как лаконичный синтаксис для анонимных функций.

И анонимные, и стрелочные функции реализованы через класс Closure.

Основной вид записи стрелочных функций: `fn (argument_list) => expr`.

Стрелочные функции работают так же, как анонимные функции, за исключением того, что доступ к переменным родительской области выполняется автоматически.

Когда стрелочная функция использует переменную, которую определили в родительской области, переменная неявно захватывается по значению. В следующем примере функции *$fn1* и *$fn2* ведут себя одинаково.

Пример #1 Стрелочные функции захватывают переменные по значению автоматически

```
<?php

$y = 1;

$fn1 = fn($x) => $x + $y;
// эквивалентно использованию $y по значению:
$fn2 = function ($x) use ($y) {
return $x + $y;
};

var_export($fn1(3));

?>
```

Результат выполнения приведённого примера:

```
4
```

Это также работает во вложенных стрелочных функциях:

Пример #2 Стрелочные функции захватывают переменные по значению автоматически, даже когда они вложены

```
<?php

$z = 1;
$fn = fn($x) => fn($y) => $x * $y + $z;
// Выведет 51
var_export($fn(5)(10));

?>
```

Подобно анонимным функциям, синтаксис стрелочных функций допускает произвольные сигнатуры функций, включая типы параметров и возвращаемых значений, значения по умолчанию, переменные, а также передачу и возврат по ссылке. Ниже приведены корректные примеры стрелочных функций:

Пример #3 Примеры использования стрелочных функций

```
<?php

fn(array $x) => $x;
static fn(): int => $x;
fn($x = 42) => $x;
fn(&$x) => $x;
fn&($x) => $x;
fn($x, ...$rest) => $rest;

?>
```

Стрелочные функции используют привязку переменных по значению. Это примерно эквивалентно выполнению `use($x)` для каждой переменной *$x*, используемой внутри стрелочной функции. Привязка по значению означает, что невозможно изменить какие-либо значения из внешней области. Вместо этого можно использовать анонимные функции для привязок по ссылкам.

Пример #4 Стрелочные функции не умеют изменять значения из внешней области видимости

```
<?php

$x = 1;
$fn = fn() => $x++; // Ничего не изменит
$fn();
var_export($x); // Выведет 1

?>
```

Список изменений

Версия	Описание
7.4.0	Стали доступны стрелочные функции.

Примечания

> **Замечание**: Можно вызывать функции func_num_args(), func_get_arg() и func_get_args() в стрелочной функции.

＋add a note

User Contributed Notes 5 notes

up
down
34
InvisibleSmiley ¶
3 years ago
Unlike anonymous functions, arrow functions cannot have a void return type declaration.

May seem obvious, but if you thought you could make use of the benefits of arrow functions (using variables from the parent scope) to simplify a function or method call, keep in mind that this is only possible if you do NOT tell PHP that the arrow function does indeed return void.

up
down
35
Koushil Mankali ¶
3 years ago
In example 4 (Values from the outer scope cannot be modified by arrow functions)

```
<?php

$x = 1;
$fn = fn() => $x++; // Has no effect
$fn();
var_export($x); // Outputs 1

?>
```

Here we can use reference variable in fn(&$x) and pass the value from function call $fn($x) so that we will get the output as expected with out using Anonymous functions.

Example:

```
<?php
```

```
$x = 1;
$fn = fn(&$x) => $x++;
$fn($x);
var_export($x);

?>
```

Output : 2 (as expected)

But here it will not take values from parent scope automatically but we have to pass them explicitly.

up
down
14
itsunclexo at gmail dot com ¶
2 years ago

As you already know, variable bindings occur in arrow functions by "by-value". That means, an arrow function returns a copy of the value of the variable used in it from the outer scope.

Now let us see an example of how a arrow function returns a reference instead of a copy of a value.

```
<?php

$x = 0;

$fn = fn &(&$x) => $x; // Returns a reference

$y = &$fn($x); // Now $y represents the reference

var_dump($y); // Outputs: 0

$y = 3; // Changing value of $y affects $x

var_dump($x); // Ouputs: 3

?>
```

up
down
11
dexen dot devries at gmail dot com ¶
3 years ago

Beware compact() not being able to access (import) variables from external scope (known in versions: 7.4.0, 7.4.8) (bug: https://bugs.php.net/bug.php?id=78970).

A workaround is available - use the variable directly; this will cause it to be imported into the arrow function's namespace and make it available to the compact() too.

```
<?php
$aa = 111;
$accessing_variable_works = fn($bb) => [ $aa, $bb ];
$compact_is_broken = fn($bb) => compact('aa', 'bb');
$compact_can_work_with_workaround = fn($bb) => compact('aa', 'bb') + ['workaround' => $aa];
var_dump($accessing_variable_works(333));
var_dump($compact_is_broken(555));
var_dump($compact_can_work_with_workaround(777));
?>

result:
array(2) {
[0]=>
int(111)
[1]=>
int(333)
```

```
}
PHP Notice: compact(): Undefined variable: aa in /home/m/vlt/guitar/tlb/s/public_html/index.php on line 9
array(1) {
["bb"]=>
int(555)
}
array(3) {
["aa"]=>
int(111)
["bb"]=>
int(777)
["workaround"]=>
int(111)
}
```

up
down
-44

zhangchengming at kkguan dot com ¶
3 years ago

```
<?php

$x = 1;

(fn() => print($x))(); // Outputs 1

(fn($x) => print($x))(2); // Outputs 2
```

＋add a note