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

PHP 7.0.x 弃用的功能 »
« 不向后兼容的变更

- PHP 手册
- 附录
- 从PHP 5.6.x 移植到 PHP 7.0.x

Change language: Chinese (Simplified) ▼

Edit Report a Bug

新特性

标量类型声明

标量类型声明 有两种模式: 强制 (默认) 和 严格模式。 现在可以使用下列类型参数（无论用强制模式还是严格模式）： 字符串(string), 整数 (*int*), 浮点数 (float), 以及布尔值 (*bool*)。它们扩充了PHP5中引入的其他类型：类名，接口，数组和 回调类型。

```
<?php
// Coercive mode
function sumOfInts(int ...$ints)
{
   return array_sum($ints);
}

var_dump(sumOfInts(2, '3', 4.1));
```

以上例程会输出：

```
int(9)
```

要使用严格模式，一个 *declare* 声明指令必须放在文件的顶部。这意味着严格声明标量是基于文件可配的。 这个指令不仅影响参数的类型声明，也影响到函数的返回值声明（参见 返回值类型声明, 内置的PHP函数以及扩展中加载的PHP函数）

完整的标量类型声明文档和示例参见类型声明章节。

返回值类型声明

PHP 7 增加了对返回类型声明的支持。 类似于参数类型声明，返回类型声明指明了函数返回值的类型。可用的类型与参数声明中可用的类型相同。

```
<?php

function arraysSum(array ...$arrays): array
{
   return array_map(function(array $array): int {
      return array_sum($array);
   }, $arrays);
}

print_r(arraysSum([1,2,3], [4,5,6], [7,8,9]));
```

以上例程会输出：

```
Array
(
   [0] => 6
   [1] => 15
   [2] => 24
)
```

完整的标量类型声明文档和示例可参见 返回值类型声明.

null合并运算符

由于日常使用中存在大量同时使用三元表达式和 isset()的情况， 我们添加了null合并运算符 (??) 这个语法糖。如果变量存在且值不为NULL， 它就会返回自身的值，否则返回它的第二个操作数。

```
<?php
// Fetches the value of $_GET['user'] and returns 'nobody'
// if it does not exist.
$username = $_GET['user'] ?? 'nobody';
// This is equivalent to:
$username = isset($_GET['user']) ? $_GET['user'] : 'nobody';

// Coalesces can be chained: this will return the first
// defined value out of $_GET['user'], $_POST['user'], and
// 'nobody'.
$username = $_GET['user'] ?? $_POST['user'] ?? 'nobody';
?>
```

太空船操作符（组合比较符）

太空船操作符用于比较两个表达式。当*$a*小于、等于或大于*$b*时它分别返回-1、0或1。 比较的原则是沿用 PHP 的常规比较规则进行的。

```
<?php
// 整数
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1

// 浮点数
echo 1.5 <=> 1.5; // 0
echo 1.5 <=> 2.5; // -1
echo 2.5 <=> 1.5; // 1

// 字符串
echo "a" <=> "a"; // 0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1
?>
```

通过 define() 定义常量数组

Array 类型的常量现在可以通过 define() 来定义。在 PHP5.6 中仅能通过 *const* 定义。

```
<?php
define('ANIMALS', [
  'dog',
  'cat',
  'bird'
]);
```

```
echo ANIMALS[1]; // 输出 "cat"
?>
```

匿名类

现在支持通过*new class* 来实例化一个匿名类，这可以用来替代一些“用后即焚”的完整类定义。

```
<?php
interface Logger {
  public function log(string $msg);
}

class Application {
  private $logger;

  public function getLogger(): Logger {
     return $this->logger;
  }

  public function setLogger(Logger $logger) {
     $this->logger = $logger;
  }
}

$app = new Application;
$app->setLogger(new class implements Logger {
  public function log(string $msg) {
    echo $msg;
  }
});

var_dump($app->getLogger());
?>
```

以上例程会输出：

```
object(class@anonymous)#2 (0) {
}
```

详细文档可以参考 匿名类.

Unicode codepoint 转译语法

这接受一个以16进制形式的 Unicode codepoint，并打印出一个双引号或heredoc包围的 UTF-8 编码格式的字符串。 可以接受任何有效的 codepoint，并且开头的 0 是可以省略的。

```
echo "\u{aa}";
echo "\u{0000aa}";
echo "\u{9999}";
```

以上例程会输出：

a

a (same as before but with optional leading 0's)
香

Closure::call()

Closure::call() 现在有着更好的性能，简短干练的暂时绑定一个方法到对象上闭包并调用它。

```
<?php
class A {private $x = 1;}

// PHP 7 之前版本的代码
$getXCB = function() {return $this->x;};
$getX = $getXCB->bindTo(new A, 'A'); // 中间层闭包
echo $getX();

// PHP 7+ 及更高版本的代码
$getX = function() {return $this->x;};
echo $getX->call(new A);
```

以上例程会输出：

```
1
1
```

为unserialize()提供过滤

这个特性旨在提供更安全的方式解包不可靠的数据。它通过白名单的方式来防止潜在的代码注入。

```
<?php

// 将所有的对象都转换为 __PHP_Incomplete_Class 对象
$data = unserialize($foo, ["allowed_classes" => false]);

// 将除 MyClass 和 MyClass2 之外的所有对象都转换为 __PHP_Incomplete_Class 对象
$data = unserialize($foo, ["allowed_classes" => ["MyClass", "MyClass2"]);

// 默认情况下所有的类都是可接受的，等同于省略第二个参数
$data = unserialize($foo, ["allowed_classes" => true]);
```

IntlChar

新增加的 IntlChar 类旨在暴露出更多的 ICU 功能。这个类自身定义了许多静态方法用于操作多字符集的 unicode 字符。

```
<?php

printf('%x', IntlChar::CODEPOINT_MAX);
echo IntlChar::charName('@');
var_dump(IntlChar::ispunct('!'));
```

以上例程会输出：

```
10ffff
COMMERCIAL AT
bool(true)
```

若要使用此类，请先安装Intl扩展

预期

预期是向后兼用并增强之前的 assert() 的方法。 它使得在生产环境中启用断言为零成本，并且提供当断言失败时抛出特定异常的能力。

老版本的API出于兼容目的将继续被维护，assert()现在是一个语言结构，它允许第一个参数是一个表达式，而不仅仅是一个待计算的 string或一个待测试的boolean。

```
<?php
ini_set('assert.exception', 1);

class CustomError extends AssertionError {}

assert(false, new CustomError('Some error message'));
?>
```

以上例程会输出：

```
Fatal error: Uncaught CustomError: Some error message
```

关于这个特性的完整说明，包括如何在开发和生产环境中配置它，可以在assert()的 expectations section章节找到。

Group *use* declarations

从同一 *namespace* 导入的类、函数和常量现在可以通过单个 *use* 语句 一次性导入了。

```
<?php

// PHP 7 之前的代码
use some\namespace\ClassA;
use some\namespace\ClassB;
use some\namespace\ClassC as C;

use function some\namespace\fn_a;
use function some\namespace\fn_b;
use function some\namespace\fn_c;

use const some\namespace\ConstA;
use const some\namespace\ConstB;
use const some\namespace\ConstC;

// PHP 7+ 及更高版本的代码
use some\namespace\{ClassA, ClassB, ClassC as C};
use function some\namespace\{fn_a, fn_b, fn_c};
use const some\namespace\{ConstA, ConstB, ConstC};
?>
```

生成器可以返回表达式

此特性基于 PHP 5.5 版本中引入的生成器特性构建的。 它允许在生成器函数中通过使用 *return* 语法来返回一个表达式 （但是不允许返回引用值）， 可以通过调用 *Generator::getReturn()* 方法来获取生成器的返回值， 但是这个方法只能在生成器完成产生工作以后调用一次。

```
<?php

$gen = (function() {
  yield 1;
  yield 2;

  return 3;
})();

foreach ($gen as $val) {
  echo $val, PHP_EOL;
}

echo $gen->getReturn(), PHP_EOL;
```

以上例程会输出：

```
1
2
3
```

在生成器中能够返回最终的值是一个非常便利的特性， 因为它使得调用生成器的客户端代码可以直接得到生成器（或者其他协同计算）的返回值， 相对于之前版本中客户端代码必须先检查生成器是否产生了最终的值然后再进行响应处理 来得方便多了。

Generator delegation

现在，只需在最外层生成其中使用 *yield from*， 就可以把一个生成器自动委派给其他的生成器，**Traversable** 对象或者 array。

```
<?php

function gen()
{
  yield 1;
  yield 2;

  yield from gen2();
}

function gen2()
{
  yield 3;
  yield 4;
}
```

```
foreach (gen() as $val)
{
    echo $val, PHP_EOL;
}

?>
```

以上例程会输出：

```
1
2
3
4
```

整数除法函数 intdiv()

新加的函数 intdiv() 用来进行 整数的除法运算。

```
<?php

var_dump(intdiv(10, 3));
?>
```

以上例程会输出：

```
int(3)
```

会话选项

session_start() 可以接受一个 array 作为参数， 用来覆盖 php.ini 文件中设置的 会话配置选项。

在调用 session_start() 的时候， 传入的选项参数中也支持 session.lazy_write 行为， 默认情况下这个配置项是打开的。它的作用是控制 PHP 只有在会话中的数据发生变化的时候才 写入会话存储文件，如果会话中的数据没有发生改变，那么 PHP 会在读取完会话数据之后， 立即关闭会话存储文件，不做任何修改，可以通过设置 *read_and_close* 来实现。

例如，下列代码设置 session.cache_limiter 为 *private*，并且在读取完毕会话数据之后马上关闭会话存储文件。

```
<?php
session_start([
    'cache_limiter' => 'private',
    'read_and_close' => true,
]);
?>
```

preg_replace_callback_array()

在 PHP 7 之前，当使用 preg_replace_callback() 函数的时候， 由于针对每个正则表达式都要执行回调函数，可能导致过多的分支代码。 而使用新加的 preg_replace_callback_array() 函数， 可以使得代码更加简洁。

现在，可以使用一个关联数组来对每个正则表达式注册回调函数， 正则表达式本身作为关联数组的键， 而对应的回调函数就是关联数组的值。

CSPRNG Functions

新加入两个跨平台的函数： random_bytes() 和 random_int() 用来产生高安全级别的随机字符串和随机整数。

可以使用 list() 函数来展开实现了 ArrayAccess 接口的对象

在之前版本中，list() 函数不能保证 正确的展开实现了 **ArrayAccess** 接口的对象， 现在这个问题已经被修复。

其他特性

- 允许在克隆表达式上访问对象成员，例如： *(clone $foo)->bar()*。

add a note

User Contributed Notes 2 notes

up
down
101
PawelD ¶
2 years ago

```
<?php
class foo { static $bar = 'baz'; }
var_dump('foo'::$bar);
?>
```

if < php7.0

then we will receive a syntax error, unexpected '::' (T_PAAMAYIM_NEKUDOTAYIM)

but php7 returns string(3) "baz"

I think it's not documented anywhere

up
down
14
TerryE ¶
1 year ago

```
$a = ''; // or 0 or false

$b = $a ?? 'a';
// $b is '' or 0 or false

$c = $a ?: 'a';
// $c is 'a'
```

add a note

- 从PHP 5.6.x 移植到 PHP 7.0.x