Search

Getting Started
- Introduction
- A simple tutorial

Language Reference
- Basic syntax
- Types
- Variables
- Constants
- Expressions
- Operators
- Control Structures
- Functions
- Classes and Objects
- Namespaces
- Enumerations
- Errors
- Exceptions
- Fibers
- Generators
- Attributes
- References Explained
- Predefined Variables
- Predefined Exceptions
- Predefined Interfaces and Classes
- Predefined Attributes
- Context options and parameters
- Supported Protocols and Wrappers

Security
- Introduction
- General considerations
- Installed as CGI binary
- Installed as an Apache module
- Session Security
- Filesystem Security
- Database Security
- Error Reporting
- User Submitted Data
- Hiding PHP
- Keeping Current

Features
- HTTP authentication with PHP
- Cookies
- Sessions
- Dealing with XForms
- Handling file uploads
- Using remote files
- Connection handling
- Persistent Database Connections
- Command line usage

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

Сравнение генераторов с объектами класса Iterator »
« Знакомство с генераторами

- Руководство по PHP
- Справочник языка
- Генераторы

Change language: Russian

Синтаксис генераторов

Генератор в целом выглядит как обычная функция, за исключением того, что вместо возвращения одного значения, генератор будет перебирать столько значений, сколько необходимо. Любая функция, содержащая yield, является функцией генератора.

Когда вызывается генератор, он возвращает объект, который можно итерировать. Когда вы итерируете этот объект (например, в цикле foreach), PHP вызывает методы итерации объекта каждый раз, когда вам нужно новое значение, после чего сохраняет состояние генератора и при следующем вызове возвращает следующее значение.

Когда все значения в генераторе закончились, генератор просто завершит работу, ничего не вернув. После этого основной код продолжит работу, как если бы в массиве закончились элементы для перебора.

> **Замечание**:
>
> Генераторы могут возвращать значения, которые можно получить с помощью Generator::getReturn().

Ключевое слово yield

Вся суть генератора заключается в ключевом слове **yield**. В самом простом варианте оператор "yield" можно рассматривать как оператор "return", за исключением того, что вместо прекращения работы функции, "yield" только приостанавливает её выполнение и возвращает текущее значение, и при следующем вызове функции она возобновит выполнение с места, на котором прервалась.

Пример #1 Простой пример выдачи значений

```
<?php
function gen_one_to_three() {
for ($i = 1; $i <= 3; $i++) {
// Обратите внимание, что $i сохраняет своё значение между вызовами.
yield $i;
}
}

$generator = gen_one_to_three();
foreach ($generator as $value) {
echo "$value\n";
}
?>
```

Результат выполнения приведённого примера:

```
1
2
3
```

> **Замечание**:
>
> Последовательность целых чисел будет связана с получаемыми от генератора значениями, как будто перебирается неассоциативный массив.

Получение значений с ключами

PHP поддерживает ассоциативные массивы, и генераторы не являются исключением. Так же как можно получать простые значения, как показано выше, вы можете получать значения с ключами.

Синтаксис получения ключ/значение очень похож на синтаксис ассоциативных массивов, как показано ниже.

Пример #2 Получение пар ключ/значение

```
<?php
/* $input содержит пары ключ/значение разделённые точкой с запятой */

$input = <<<'EOF'
```

```
1;PHP;Любит знаки доллара
2;Python;Любит пробелы
3;Ruby;Любит блоки
EOF;

function input_parser($input) {
foreach (explode("\n", $input) as $line) {
$fields = explode(';', $line);
$id = array_shift($fields);

yield $id => $fields;
}
}

foreach (input_parser($input) as $id => $fields) {
echo "$id:\n";
echo " $fields[0]\n";
echo " $fields[1]\n";
}
?>
```

Результат выполнения приведённого примера:

```
1:
    PHP
    Любит знаки доллара
2:
    Python
    Любит пробелы
3:
    Ruby
    Любит блоки
```

Получение NULL

Для получения **null** нужно вызвать "yield" без аргументов. Ключ сгенерируется автоматически.

Пример #3 Получение null

```
<?php
function gen_three_nulls() {
foreach (range(1, 3) as $i) {
yield;
}
}

var_dump(iterator_to_array(gen_three_nulls()));
?>
```

Результат выполнения приведённого примера:

```
array(3) {
  [0]=>
  NULL
  [1]=>
  NULL
  [2]=>
  NULL
}
```

Получение значения по ссылке

Генераторы могут отдавать значения по ссылке. Это делается так же, как возврат ссылок из функций: добавлением амперсанда (&) перед именем функции.

Пример #4 Получение значений по ссылке

```
<?php
```

```php
function &gen_reference() {
$value = 3;

while ($value > 0) {
yield $value;
}
}

/* Обратите внимание, что мы можем изменять $number в цикле, и
* так как генератор возвращает ссылку, $value
* в gen_reference() также изменится. */
foreach (gen_reference() as &$number) {
echo (--$number).'... ';
}
?>
```

Результат выполнения приведённого примера:

```
2... 1... 0...
```

Делегирование генератора с помощью yield from

Делегирование генератора позволяет вам получать значения из другого генератора, объекта Traversable, или массива, используя **yield from**. Внешний генератор будет возвращать значения из внутреннего генератора, объекта или массива, до того момента, пока они их отдают, после чего продолжится выполнение внешнего генератора.

Если генератор используется с **yield from**, то выражение **yield from** также будет возвращать значения из внутреннего генератора.

Предостережение

Сохранение в массив (например, с помощью iterator_to_array())

yield from не сбрасывает ключи. Ключи, возвращённые из объекта Traversable или массива, сохранятся. Таким образом, некоторые значения, могут пересекаться по ключам с другими **yield** или **yield from**, что, при записи в массив, повлечёт за собой перезапись прежних значений.

Общий случай, когда это имеет значение, это когда iterator_to_array() возвращает массив с ключами по умолчанию. В этом случае можно получить неожиданный результат. iterator_to_array() имеет второй параметр `preserve_keys`, который можно установить в **`false`**, для генерации собственных ключей и игнорирования ключей, переданных из объекта Generator.

Пример #5 yield from с iterator_to_array()

```php
<?php
function inner() {
yield 1; // ключ 0
yield 2; // ключ 1
yield 3; // ключ 2
}
function gen() {
yield 0; // ключ 0
yield from inner(); // ключи 0-2
yield 4; // ключ 1
}
// Задайте false вторым параметром для получения массива [0, 1, 2, 3, 4]
var_dump(iterator_to_array(gen()));
?>
```

Результат выполнения приведённого примера:

```
array(3) {
  [0]=>
  int(1)
  [1]=>
```

```
  int(4)
  [2]=>
  int(3)
}
```

Пример #6 Основы использования yield from

```
<?php
function count_to_ten() {
yield 1;
yield 2;
yield from [3, 4];
yield from new ArrayIterator([5, 6]);
yield from seven_eight();
yield 9;
yield 10;
}

function seven_eight() {
yield 7;
yield from eight();
}

function eight() {
yield 8;
}

foreach (count_to_ten() as $num) {
echo "$num ";
}
?>
```

Результат выполнения приведённого примера:

```
1 2 3 4 5 6 7 8 9 10
```

Пример #7 yield from и возвращаемые значения

```
<?php
function count_to_ten() {
yield 1;
yield 2;
yield from [3, 4];
yield from new ArrayIterator([5, 6]);
yield from seven_eight();
return yield from nine_ten();
}

function seven_eight() {
yield 7;
yield from eight();
}

function eight() {
yield 8;
}

function nine_ten() {
yield 9;
return 10;
}

$gen = count_to_ten();
foreach ($gen as $num) {
```

```
echo "$num ";
}
echo $gen->getReturn();
?>
```

Результат выполнения приведённого примера:

```
1 2 3 4 5 6 7 8 9 10
```

＋add a note

User Contributed Notes 9 notes

up
down
122
Adil Ihan (adilmedya at gmail dot com) ¶
10 years ago

```
For example yield keyword with Fibonacci:

function getFibonacci()
{
$i = 0;
$k = 1; //first fibonacci value
yield $k;
while(true)
{
$k = $i + $k;
$i = $k - $i;
yield $k;
}
}

$y = 0;

foreach(getFibonacci() as $fibonacci)
{
echo $fibonacci . "\n";
$y++;
if($y > 30)
{
break; // infinite loop prevent
}
}
```

up
down
47
info at boukeversteegh dot nl ¶
9 years ago

```
[This comment replaces my previous comment]

You can use generators to do lazy loading of lists. You only compute the items that are actually used. However, when you
want to load more items, how to cache the ones already loaded?

Here is how to do cached lazy loading with a generator:

<?php
class CachedGenerator {
protected $cache = [];
protected $generator = null;

public function __construct($generator) {
$this->generator = $generator;
```

```
}

public function generator() {
foreach($this->cache as $item) yield $item;

while( $this->generator->valid() ) {
$this->cache[] = $current = $this->generator->current();
$this->generator->next();
yield $current;
}
}
}
class Foobar {
protected $loader = null;

protected function loadItems() {
foreach(range(0,10) as $i) {
usleep(200000);
yield $i;
}
}

public function getItems() {
$this->loader = $this->loader ?: new CachedGenerator($this->loadItems());
return $this->loader->generator();
}
}

$f = new Foobar;

# First
print "First\n";
foreach($f->getItems() as $i) {
print $i . "\n";
if( $i == 5 ) {
break;
}
}

# Second (items 1-5 are cached, 6-10 are loaded)
print "Second\n";
foreach($f->getItems() as $i) {
print $i . "\n";
}

# Third (all items are cached and returned instantly)
print "Third\n";
foreach($f->getItems() as $i) {
print $i . "\n";
}
?>
```

up
down
19

Hayley Watson ¶

8 years ago

If for some strange reason you need a generator that doesn't yield anything, an empty function doesn't work; the function needs a yield statement to be recognised as a generator.

```
<?php

function gndn()
```

```
{
}

foreach(gndn() as $it)
{
echo 'FNORD';
}

?>
```

But it's enough to have the yield syntactically present even if it's not reachable:

```
<?php

function gndn()
{
if(false) { yield; }
}

foreach(gndn() as $it)
{
echo 'FNORD';
}

?>
```

up
down
11
zilvinas at kuusas dot lt ¶
8 years ago

Do not call generator functions directly, that won't work.

```
<?php

function my_transform($value) {
var_dump($value);
return $value * 2;
}

function my_function(array $values) {
foreach ($values as $value) {
yield my_transform($value);
}
}

$data = [1, 5, 10];
// my_transform() won't be called inside my_function()
my_function($data);

# my_transform() will be called.
foreach (my_function($data) as $val) {
// ...
}
?>
```

up
down
14
Harun Yasar harunyasar at mail dot com ¶
8 years ago

That is a simple fibonacci generator.

```
<?php
```

```
function fibonacci($item) {
$a = 0;
$b = 1;
for ($i = 0; $i < $item; $i++) {
yield $a;
$a = $b - $a;
$b = $a + $b;
}
}

# give me the first ten fibonacci numbers
$fibo = fibonacci(10);
foreach ($fibo as $value) {
echo "$value\n";
}
?>
```

up
down
11
christophe dot maymard at gmail dot com ¶
9 years ago

```
<?php
//Example of class implementing IteratorAggregate using generator

class ValueCollection implements IteratorAggregate
{
private $items = array();

public function addValue($item)
{
$this->items[] = $item;
return $this;
}

public function getIterator()
{
foreach ($this->items as $item) {
yield $item;
}
}
}

//Initializes a collection
$collection = new ValueCollection();
$collection
->addValue('A string')
->addValue(new stdClass())
->addValue(NULL);

foreach ($collection as $item) {
var_dump($item);
}
```

up
down
6
Shumeyko Dmitriy ¶
10 years ago

This is little example of using generators with recursion. Used version of php is 5.5.5

```
[php]
<?php
define ("DS", DIRECTORY_SEPARATOR);
define ("ZERO_DEPTH", 0);
```

```
define ("DEPTHLESS", -1);
define ("OPEN_SUCCESS", True);
define ("END_OF_LIST", False);
define ("CURRENT_DIR", ".");
define ("PARENT_DIR", "..");

function DirTreeTraversal($DirName, $MaxDepth = DEPTHLESS, $CurrDepth = ZERO_DEPTH)
{
if (($MaxDepth === DEPTHLESS) || ($CurrDepth < $MaxDepth)) {
$DirHandle = opendir($DirName);
if ($DirHandle !== OPEN_SUCCESS) {
try{
while (($FileName = readdir($DirHandle)) !== END_OF_LIST) { //read all file in directory
if (($FileName != CURRENT_DIR) && ($FileName != PARENT_DIR)) {
$FullName = $DirName.$FileName;
yield $FullName;
if(is_dir($FullName)) { //include sub files and directories
$SubTrav = DirTreeTraversal($FullName.DS, $MaxDepth, ($CurrDepth + 1));
foreach($SubTrav as $SubItem) yield $SubItem;
}
}
}
} finally {
closedir($DirHandle);
}
}
}
}

$PathTrav = DirTreeTraversal("C:".DS, 2);
print "<pre>";
foreach($PathTrav as $FileName) printf("%s\n", $FileName);
print "</pre>";
[/php]
```

up

down

-3

christianggimenez at gmail dot com ¶

4 years ago

```
Module list of a number from 1 to 100.

<?php

function list_of_modulo(){

for($i = 1; $i <= 100; $i++){

if(($i % 2) == 0){
yield $i;
}
}
}

$modulos = list_of_modulo();

foreach($modulos as $value){

echo "$value\n";
}

?>
```

up

down
-45
denshadewillspam at HOTMAIL dot com ¶
9 years ago
Note that you can't use count() on generators.

```
/**
* @return integer[]
*/
function xrange() {
for ($a = 0; $a < 10; $a++)
{
yield $a;
}
}

function mycount(Traversable $traversable)
{
$skip = 0;
foreach($traversable as $skip)
{
$skip++;
}
return $skip;
}
echo "Count:" . count(xrange()). PHP_EOL;
echo "Count:" . mycount(xrange()). PHP_EOL;
```

＋add a note

- Генераторы
  - Знакомство с генераторами
  - Синтаксис генераторов
  - Сравнение генераторов с объектами класса Iterator