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

Магические методы »
« Перегрузка

- Руководство по PHP
- Справочник языка
- Классы и объекты

Change language: Russian

Итераторы объектов

PHP предоставляет такой способ объявления объектов, который даёт возможность пройти по списку элементов данного объекта, например, с помощью оператора foreach. По умолчанию, в этом обходе (итерации) будут участвовать все видимые свойства объекта.

Пример #1 Итерация простого объекта

```
<?php
class MyClass
{
public $var1 = 'значение 1';
public $var2 = 'значение 2';
public $var3 = 'значение 3';

protected $protected = 'защищённая переменная';
private $private = 'закрытая переменная';

function iterateVisible() {
echo "MyClass::iterateVisible:\n";
foreach ($this as $key => $value) {
print "$key => $value\n";
}
}
}

$class = new MyClass();

foreach($class as $key => $value) {
print "$key => $value\n";
}
echo "\n";


$class->iterateVisible();

?>
```

Результат выполнения приведённого примера:

```
var1 => значение 1
var2 => значение 2
var3 => значение 3

MyClass::iterateVisible:
var1 => значение 1
var2 => значение 2
var3 => значение 3
protected => защищённая переменная
private => закрытая переменная
```

Как показывает результат, foreach проитерировал все доступные и принадлежащие объекту видимые свойства.

Смотрите также

- Генераторы
- Iterator
- IteratorAggregate
- Итераторы

User Contributed Notes 19 notes

up
down
20
php dot net dot nsp at cvogt dot org ¶
11 years ago
there is still an open bug about using current() etc. with iterators
https://bugs.php.net/bug.php?id=49369
up
down
16
wavetrex A(nospam)T gmail DOT com ¶
15 years ago
By reading the posts below I wondered if it really is impossible to make an ArrayAccess implementation really behave like a true array (by being multi level)

Seems like it's not impossible. Not very preety but usable

```
<?php

class ArrayAccessImpl implements ArrayAccess {

private $data = array();

public function offsetUnset($index) {}

public function offsetSet($index, $value) {
// echo ("SET: ".$index."<br>");

if(isset($data[$index])) {
unset($data[$index]);
}

$u = &$this->data[$index];
if(is_array($value)) {
$u = new ArrayAccessImpl();
foreach($value as $idx=>$e)
$u[$idx]=$e;
} else
$u=$value;
}

public function offsetGet($index) {
// echo ("GET: ".$index."<br>");

if(!isset($this->data[$index]))
$this->data[$index]=new ArrayAccessImpl();

return $this->data[$index];
}

public function offsetExists($index) {
// echo ("EXISTS: ".$index."<br>");

if(isset($this->data[$index])) {
if($this->data[$index] instanceof ArrayAccessImpl) {
if(count($this->data[$index]->data)>0)
return true;
else
return false;
} else
return true;
} else
```

```
return false;
}

}

echo "ArrayAccess implementation that behaves like a multi-level array<hr />";

$data = new ArrayAccessImpl();

$data['string']="Just a simple string";
$data['number']=33;
$data['array']['another_string']="Alpha";
$data['array']['some_object']=new stdClass();
$data['array']['another_array']['x']['y']="LOL @ Whoever said it can't be done !";
$data['blank_array']=array();

echo "'array' Isset? "; print_r(isset($data['array'])); echo "<hr />";
echo "<pre>"; print_r($data['array']['non_existent']); echo "</pre>If attempting to read an offset that doesn't exist it
returns a blank object! Use isset() to check if it exists!<br>";
echo "'non_existent' Isset? "; print_r(isset($data['array']['non_existent'])); echo "<br />";
echo "<pre>"; print_r($data['blank_array']); echo "</pre>A blank array unfortunately returns similar results :(<br />";
echo "'blank_array' Isset? "; print_r(isset($data['blank_array'])); echo "<hr />";
echo "<pre>"; print_r($data); echo "</pre> (non_existent remains in the structure. If someone can help to solve this I'll
appreciate it)<hr />";

echo "Display some value that exists: ".$data['array']['another_string'];

?>

(in the two links mentioned below by artur at jedlinski... they say you can't use references, so I didn't used them.
My implementation uses recursive objects)

If anyone finds a better (cleaner) sollution, please e-mail me.
Thanks,
Wave.
```

up
down
5

hlegius at gmail dot com ¶

15 years ago

```
Iterator interface usign key() next() rewind() is MORE slow than extends ArrayIterator with ArrayIterator::next(),
ArrayIterator::rewind(), etc.,
```

up
down
5

elias at need dot spam ¶

18 years ago

```
The MyIterator::valid() method above ist bad, because it
breaks on entries with 0 or empty strings, use key() instead:

<?php
public function valid()
{
return ! is_null(key($this->var));
}
?>

read about current() drawbacks:
http://php.net/current
```

up
down
5

strrev('ed.relpmeur@ekneos'); ¶

18 years ago

Use the SPL ArrayAccess interface to call an object as array:

http://www.php.net/~helly/php/ext/spl/interfaceArrayAccess.html

up

down

3

markushe at web dot de ¶

18 years ago

Just something i noticed:

It seems, that when you are implementing the interface Iterator, yout method key() has to return a string or integer.

I was trying to return a object an got this error:

Illegal type returned from MyClass::key()

up

down

4

knj at aider dot dk ¶

19 years ago

if you in a string define classes that implements IteratorAggregate.

you cant use the default;

```
<?
...
public function getIterator() {
return new MyIterator(\\$this-><What ever>);
}
..
?>
```

at least not if you want to use eval(<The string>).

You have to use:

```
<?
...
public function getIterator() {
\\$arrayObj=new ArrayObject(\\$this-><What ever>);
return \\$arrayObj->getIterator();
}
...
?>
```

up

down

5

rune at zedeler dot dk ¶

16 years ago

The iterator template from knj at aider dot dk does not yield correct results.

If you do

```
<?
reset($a);
next($a);
echo current($a);
?>
```

where $a is defined over the suggested template, then the first element will be output, not the second, as expected.

up

down

2

just_somedood at yahoo dot com ¶

18 years ago

To clarify on php at moechofe's post, you CAN use the SPL to overide the array operator for a class. This, with the new features of object, and autoloading (among a buch of other things) has me completely sold on PHP5. You can also find this information on the SPL portion of the manual, but I'll post it here as well so it isn't passed up. The below Collection class will let you use the class as an array, while also using the foreach iterator:

```
<?php

class Collection implements ArrayAccess,IteratorAggregate
{
public $objectArray = Array();
//**these are the required iterator functions
function offsetExists($offset)
{
if(isset($this->objectArray[$offset])) return TRUE;
else return FALSE;
}

function & offsetGet($offset)
{
if ($this->offsetExists($offset)) return $this->objectArray[$offset];
else return (false);
}

function offsetSet($offset, $value)
{
if ($offset) $this->objectArray[$offset] = $value;
else $this->objectArray[] = $value;
}

function offsetUnset($offset)
{
unset ($this->objectArray[$offset]);
}

function & getIterator()
{
return new ArrayIterator($this->objectArray);
}
//**end required iterator functions

public function doSomething()
{
echo "I'm doing something";
}
}

?>
```

I LOVE the new SPL stuff in PHP! An example of usage is below:

```
<?php
class Contact
{
protected $name = NULL;

public function set_name($name)
{
$this->name = $name;
}

public function get_name()
{
return ($this->name);
}
}

$bob = new Collection();
```

```
$bob->doSomething();
$bob[] = new Contact();
$bob[5] = new Contact();
$bob[0]->set_name("Superman");
$bob[5]->set_name("a name of a guy");

foreach ($bob as $aContact)
{
echo $aContact->get_name() . "\r\n";
}
?>
```

Would work just fine. This makes code so much simpler and easy to follow, it's great. This is exactly the direction I had hoped PHP5 was going!

up
down
3

chad 0x40 herballure 0x2e com ¶

17 years ago

The example code given for valid() will break if the array contains a FALSE value. This code prints out a single "bool(true)" and exits the loop when it gets to the FALSE:

```
<?php
$A = array(TRUE, FALSE, TRUE, TRUE);
while(current($A) !== FALSE) {
var_dump(current($A));
next($A);
}
?>
```

Instead, the key() function should be used, since it returns NULL only at the end of the array. This code displays all four elements and then exits:

```
<?php
$A = array(TRUE, FALSE, TRUE, TRUE);
while(!is_null(key($A))) {
var_dump(current($A));
next($A);
}
?>
```

up
down
1

phpnet at nicecupofteaandasitdown dot com ¶

18 years ago

You should be prepared for your iterator's current method to be called before its next method is ever called. This certainly happens in a foreach loop. If your means of finding the next item is expensive you might want to use something like this

```
private $item;

function next() {
$this->item = &$this->getNextItem();
return $this->item;
}

public function current() {
if(!isset($this->item)) $this->next();
return $this->item;
}
```

up
down

1

celsowm ¶

4 years ago

I've created a dinamic version of grzeniufication code to allow un-, serialize more than one property:

```
<?php

class Person implements \Serializable {

public $id;
public $name;
public $birthDate;
public $surname;

public function serialize() {
return serialize((array) $this);
}

public function unserialize($serialized): void {

foreach (unserialize($serialized) as $p => $v) {
$this->{$p} = $v;
}
}

}
```

up

down

3

baldurien at bbnwn dot eu ¶

17 years ago

Beware of how works iterator in PHP if you come from Java!

In Java, iterator works like this :

```
<?php
interface Iterator<O> {
boolean hasNext();
O next();
void remove();
}
?>
```

But in php, the interface is this (I kept the generics and type because it's easier to understand)

```
<?php
interface Iterator<O> {
boolean valid();
mixed key();
O current();
void next();
void previous();
void rewind();
}
?>
```

1. valid() is more or less the equivalent of hasNext()
2. next() is not the equivalent of java next(). It returns nothing, while Java next() method return the next object, and move to next object in Collections. PHP's next() method will simply move forward.

Here is a sample with an array, first in java, then in php :

```
<?php
class ArrayIterator<O> implements Iterator<O> {
```

```
private final O[] array;
private int index = 0;

public ArrayIterator(O[] array) {
this.array = array;
}

public boolean hasNext() {
return index < array.length;
}

public O next() {
if ( !hasNext())
throw new NoSuchElementException('at end of array');
return array[index++];
}

public void remove() {
throw new UnsupportedOperationException('remove() not supported in array');
}
}
?>
```

And here is the same in php (using the appropriate function) :

```
<?php
/**
* Since the array is not mutable, it should use an internal
* index over the number of elements for the previous/next
* validation.
*/
class ArrayIterator implements Iterator {
private $array;
public function __construct($array) {
if ( !is_array($array))
throw new IllegalArgumentException('argument 0 is not an array');
$this->array = array;
$this->rewind();
}
public function valid() {
return current($this->array) !== false;
// that's the bad method (should use arrays_keys, + index)
}
public function key() {
return key($this->array);
}
public function current() {
return current($this->array);
}
public function next() {
if ( $this->valid())
throw new NoSuchElementException('at end of array');
next($this->array);
}
public function previous() {
// fails if current() = first item of array
previous($this->array);
}
public function rewind() {
reset($this->array);
}
}
```

```
?>
```

The difference is notable : don't expect next() to return something like in Java, instead use current(). This also means that you have to prefetch your collection to set the current() object. For instance, if you try to make a Directory iterator (like the one provided by PECL), rewind should invoke next() to set the first element and so on. (and the constructor should call rewind())

Also, another difference :

```
<?php
class ArrayIterable<O> implements Iterable<O> {
private final O[] array;

public ArrayIterable(O[] array) {
this.array = array;
}

public Iterator<O> iterator() {
return new ArrayIterator(array);
}
}
?>
```

When using an Iterable, in Java 1.5, you may do such loops :

```
<?php
for ( String s : new ArrayIterable<String>(new String[] {"a", "b"})) {
...
}
?>
```

Which is the same as :

```
<?php
Iterator<String> it = new ArrayIterable<String>(new String[] {"a", "b"});
while (it.hasNext()) {
String s = it.next();
...
}
?>
```

While in PHP it's not the case :

```
<?php
foreach ( $iterator as $current ) {
...
}
?>
```

Is the same as :

```
<?php
for ( $iterator->rewind(); $iterator->valid(); $iterator->next()) {
$current = $iterator->current();
...
}
?>
```

(I think we may also use IteratorAggregate to do it like with Iterable).

Take that in mind if you come from Java.

I hope this explanation is not too long...

up
down
1

mike at eastghost dot com ¶

1 year ago

The days of lovely care-to-the-wind typecasting are coming to close. Finding this devilish bug took us entirely too long.

PHP-8.2.1 was throwing errors seemingly uncaught (they were eventually seen amassing in / var / log / apache / DOMAIN-ssl-err.log) due to mismatch between return types of the necessary interface methods in our 'implements \Iterator' class (which had worked fine for many years, until our leap up to 8.2.1) versus the interface methods required by PHP.

Particularly:

```
next()
=====
ours:
public function next() {...}

PHP-8.2.1's
public function next() : void {...}

valid()
======
ours:
public function valid() {...}

PHP-8.2.1's:
public function valid() : bool {...}

key()
====
ours:
public function key() {...}

PHP-8.2.1's:
public function key() : mixed {...}

rewind()
========
ours:
public function rewind() {...}

PHP-8.2.1's:
public function rewind() : void {...}

current()
=======
ours:
public function current() {...}

PHP-8.2.1's:
public function current() : mixed {...}
```

We added the missing / now all-important return types to our function/method declarations and everything instantly worked again.

This extreme stringency is not made clear enough, IMHO, in the Iterator manual page.

up

down

0

doctorrock83_at_gmail.com ¶

16 years ago

Please remember that actually the only PHP iterating structure that uses Iterator is foreach().

Any each() or list() applied to an Object implementing iterator will not provide the expected result

up
down
0
artur at jedlinski dot pl ¶
16 years ago
One should be aware that ArrayAccess functionality described by "just_somedood at yahoo dot com" below is currently broken and thus it's pretty unusable.

Read following links to find more:
http://bugs.php.net/bug.php?id=34783
http://bugs.php.net/bug.php?id=32983
up
down
-3
PrzemekG_ at poczta dot onet dot pl ¶
18 years ago
If you want to do someting like this:

```
<?php
foreach($MyObject as $key => &$value)
$value = 'new '.$value;
?>
```

you must return values by reference in your iterator object:

```
<?php
class MyObject implements Iterator
{
/* ...... other iterator functions ...... */
/* return by reference */
public function &current()
{
return $something;
}
?>
```

This won't change values:

```
<?php
foreach($MyObject as $key => $value)
$value = 'new '.$value;
?>
```

This will change values:

```
<?php
foreach($MyObject as $key => &$value)
$value = 'new '.$value;
?>
```

I think this should be written somewhere in the documentations, but I couldn't find it.
up
down
-17
jille at hexon dot cx ¶
11 years ago
Please note that if you implement your iterator this way instead of with an IteratorAggregate you can not nest foreach-loops. This is because when the inner-loop is done the cursor is beyond the last element, then the outer-loop asks for the next element and finds the cursor beyond the last element as the innter-loop left it there.

```
<?php
// Wont work!
foreach($collection as $a) {
foreach($collection as $b) {
var_dump($a->someFunc($b));
}
}
```

```
?>
```

up

down

-10

Anonymous ¶

5 years ago

```
szerintetek ez normális, hogy objektumon végigiterálhatsz???
```

＋add a note

- Классы и объекты
  - Введение
  - Основы
  - Свойства
  - Константы классов
  - Автоматическая загрузка классов
  - Конструкторы и деструкторы
  - Область видимости
  - Наследование
  - Оператор разрешения области видимости (::)
  - Ключевое слово static
  - Абстрактные классы
  - Интерфейсы объектов
  - Трейты
  - Анонимные классы
  - Перегрузка
  - Итераторы объектов
  - Магические методы
  - Ключевое слово final
  - Клонирование объектов
  - Сравнение объектов
  - Позднее статическое связывание
  - Объекты и ссылки
  - Сериализация объектов
  - Ковариантность и контравариантность
  - Журнал изменений ООП