Generators: All About the Yield

Justin Yost
Web Developer at Loadsys

What is a Generator?

Think Iterators without the overhead of writing an Iterator.

What is a Generator?

The big trick here is that Generators provide for a looping mechanism without the memory overhead of the thing you are looping over.

Iterator vs. Array vs. Generator

- Iterators - Object Based, Rewind/Forward/Filter/etc, Limited memory usage
- Array - Simple, One way, Direct Access, Memory constrained
- Generator - Simple, One way, Limited memory usage

Generators: All About the Yield

`yield` - acts as the return from a generator

Simple Example

```
function getRange($max = 15) {
    for ($i = 1; $i < $max; $i++) {
        yield $i;
    }
}
foreach(getRange(4) as $value) {
    echo "value: {$value} ";
}
// value: 1 value: 2 value: 3
```

Key and Value Yield Example

```
function getRange($max = 15) {
    for ($i = 1; $i < $max; $i++) {
        yield $i => $max;
    }
}
foreach(getRange(4) as $key => $value) {
    echo "key: {$key} value: {$value} ";
}
// key: 1 value: 4 key: 2 value: 4 key: 3 value: 4
```

Empty Yield

```
function getRange($max = 15) {
    for ($i = 1; $i < $max; $i++) {
        yield;
    }
}
foreach(getRange(4) as $value) {
    echo "value: {$value} ";
}
// value: value: value:
```

Yield by Reference

```
function &getRange($max = 15) {
    for ($i = 1; $i < $max; $i++) {
        yield $i => $max;
    }
}
foreach(getRange(4) as &$key => $value) {
    echo "key: {$key} value: {$value} "; ($key++);
}
// key: 1 value: 4 key: 3 value: 4
```

Return

```
function getRange($max = 15) {
    for ($i = 1; $i < $max; $i++) {
        return;
    }
}
foreach(getRange(4) as $key => $value) {
    echo "key: {$key} value: {$value} ";
}
//
```

Non Empty Return

```
function getRange($max = 15) {
    for ($i = 1; $i < $max; $i++) {
        return $i;
    }
}
foreach(getRange(4) as $key => $value) {
    echo "key: {$key} value: {$value} ";
}
// !!!! Error !!!!
```

Code Samples

Generators

- Generators are Iterators without the Iterator overhead
- Generators can be interrupted in processing via `yield`
- Empty `return` ends a generator
- You can operate a Generator using Iterator `current`, `next`, etc
- Except for `rewind`, Generators are forward only Iterators

Coroutines

- Coroutines are programs that allow for nonpreemptive multitasking via multiple entry points for suspending and returning.

Generators and Coroutines

- `yield` is the trick here, we can pause executing of one method and continue on in a different method

More knowledge of Generators

Generators::send

```
function genPrint() {
    while (true) {
        $string = yield;
        echo $string . " ";
    }
}
$print = genPrint(); // (instanceof Iterator && Generator)
$print->send('fizz'); $print->send('buzz');
// fizz buzz
```

Send and Receive

```
function genPrint() {
    while (true) {
        $sent = (yield 'return-val ');
        echo $sent . " ";
    }
}
$print = genPrint();
echo $print->send('fizz'); echo $print->send('buzz');
// fizz return-val buzz return-val
```

Send and Receive

- `send` executes the generator by passing the input
- then yields the return value of the generator

Send and Receive and Current

```
function genPrint() {
    while (true) {
        $sent = (yield 'return-val ');
        echo $sent . " ";
    }
}
$print = genPrint();
echo $print->current(); echo $print->send('fizz'); echo $print->send('buzz');
// return-val fizz return-val buzz return-val
```

Send and Receive and Current

- `send` executes the generator by passing the input
- then `yields` the return value of the generator
- `current` just `yields` the return value of the generator

Multiple Yields with Send and Receive and Current

```
function genPrint() {
    while (true) {
        $sent = (yield 'return-val ');
        echo $sent . " ";
        $sent = (yield 'return-val2 ');
        echo $sent . " ";
    }
}
$print = genPrint();
echo $print->current(); echo $print->send('fizz'); echo $print->send('buzz');
// return-val fizz return-val2 buzz return-val
```

Multiple Yields with Send and Receive and Current

- `send` executes the generator by passing the input
- then `yields` the return value of the generator
- `current` just `yields` the return value of the generator
- each `yield` means we have another exit depending on where in the iteration we are
- each iteration - first yield then second, then loop, repeat

Code Example

Sourced from:

- https://scotch.io/tutorials/understanding-php-generators
- https://nikic.github.io/2012/12/22/Cooperative-multitasking-using-coroutines-in-PHP.html

Thanks/Questions?

- twitter.com/justinyost
- github.com/justinyost
- justinyost.com
- loadsys.com
- lynda.com/justinyost