PHP7 开发体验报告

运营中心：林志勇

1. 重要的写在前面

这是一个双赢的版本升级，既节省服务器硬件成本，同时提升了开发效率。网上有很多 Benchmark 报告显示 PHP7 对比 PHP5.6 在普通服务器上的性能提升达到 **5 倍以上**，除此之外 PHP7 终于迎来了这十年间其它语言早已有的主流语法特性。

由于对新版本的期待，我查看了相关官方介绍并在自己的开发环境上测评了一番。本文档不是官方文档的译文，我只挑选一些自己站在应用开发者日常使用的角度认为很重要的技术点进行详细介绍。

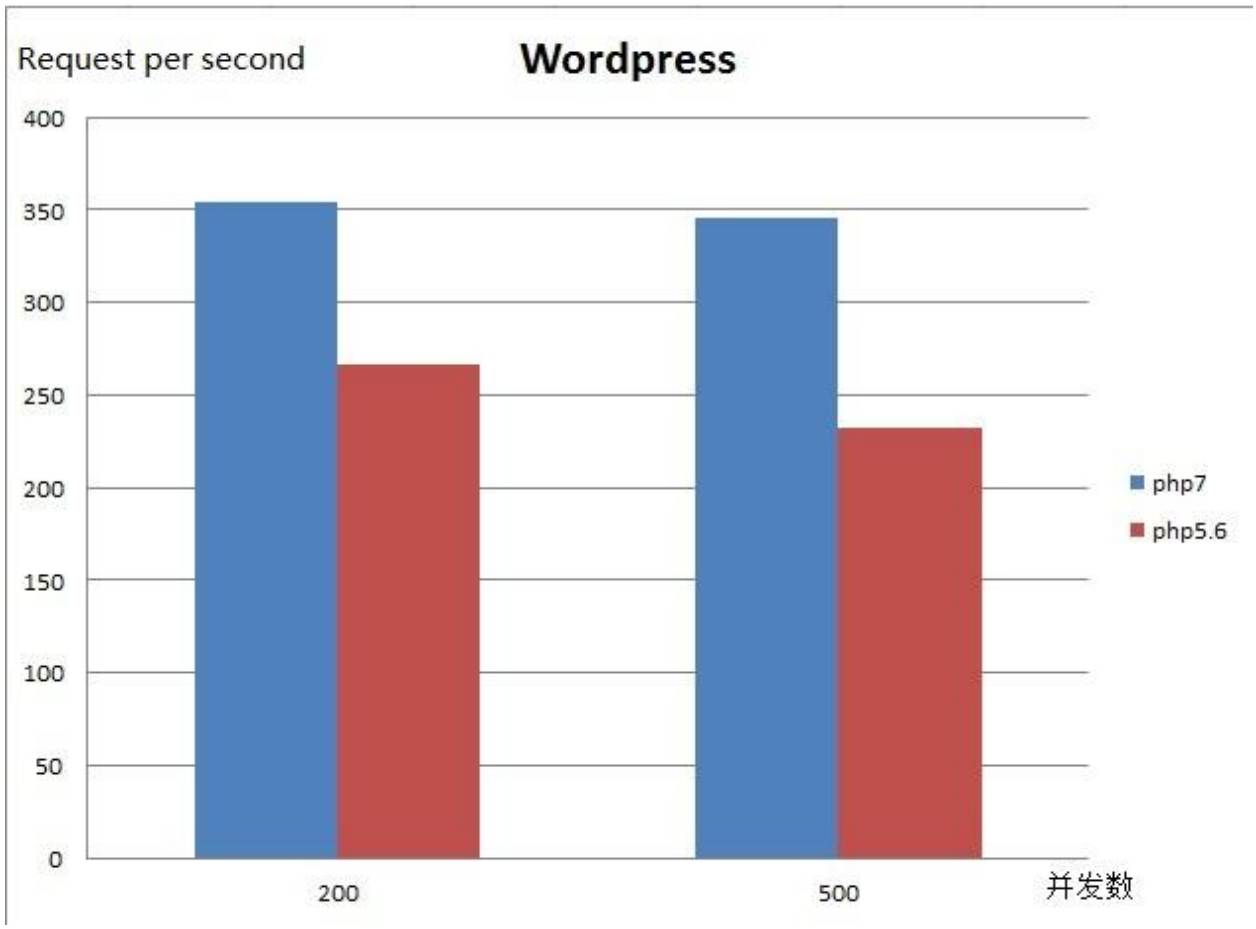
- 性能提升效果明显，在我的**老 PC 机上比 5.6 提升至少 30%**
- 语法引入了时下主流的其它语言都有的特性，使开发效率得到明显的提升
- 不兼容旧版本的地方不多，基本不会触碰到

2. Benchmark 简图

这里仅放出部分测评的简图，详细 Benchmark 过程请查阅附件另一份文档《PHP7_BenchmarkDetail.pdf》。

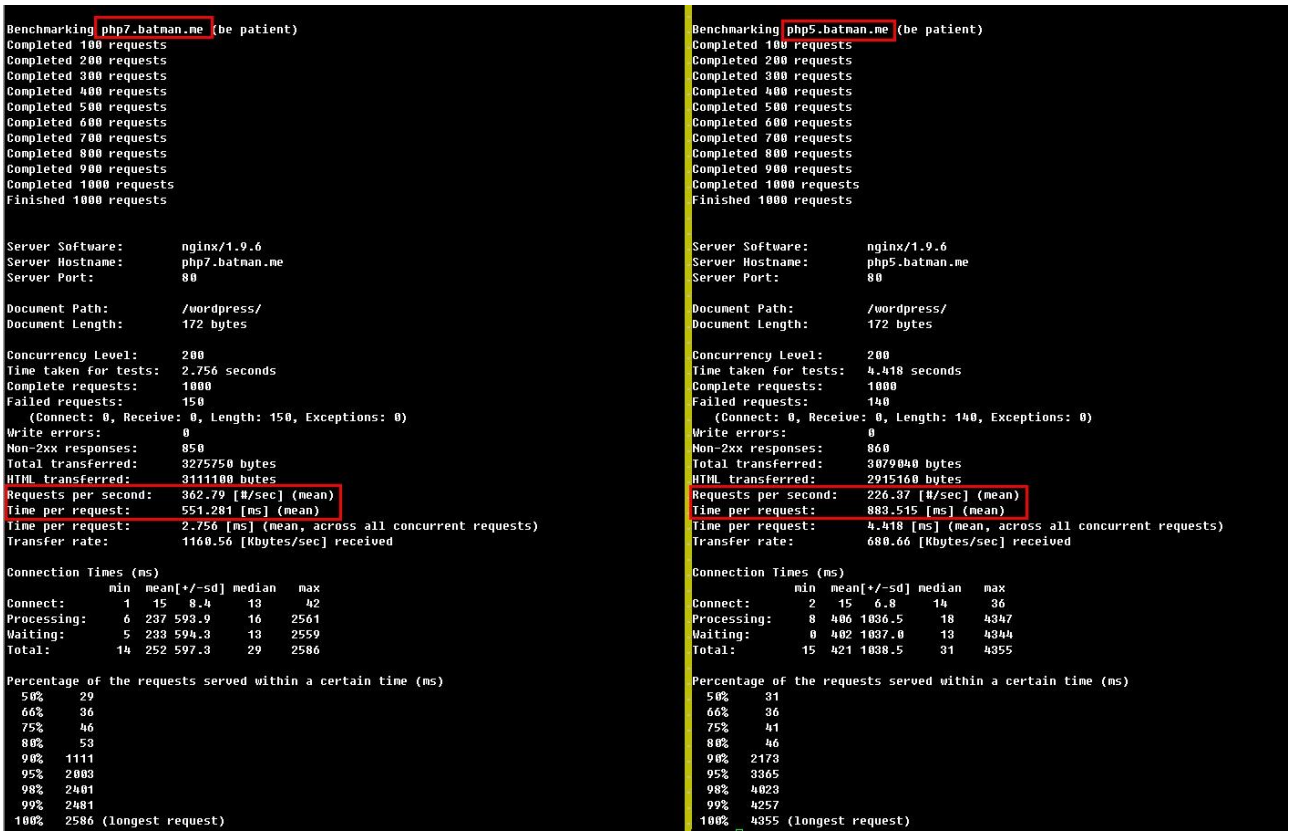
2.1. Wordpress 性能测试

Y 轴是 Request Per Second，X 轴是并发数



原图 URL:
https://raw.githubusercontent.com/sunshine17/sunshine17.github.io/master/images/statistic.jpg

2.2. Ab 压力测试

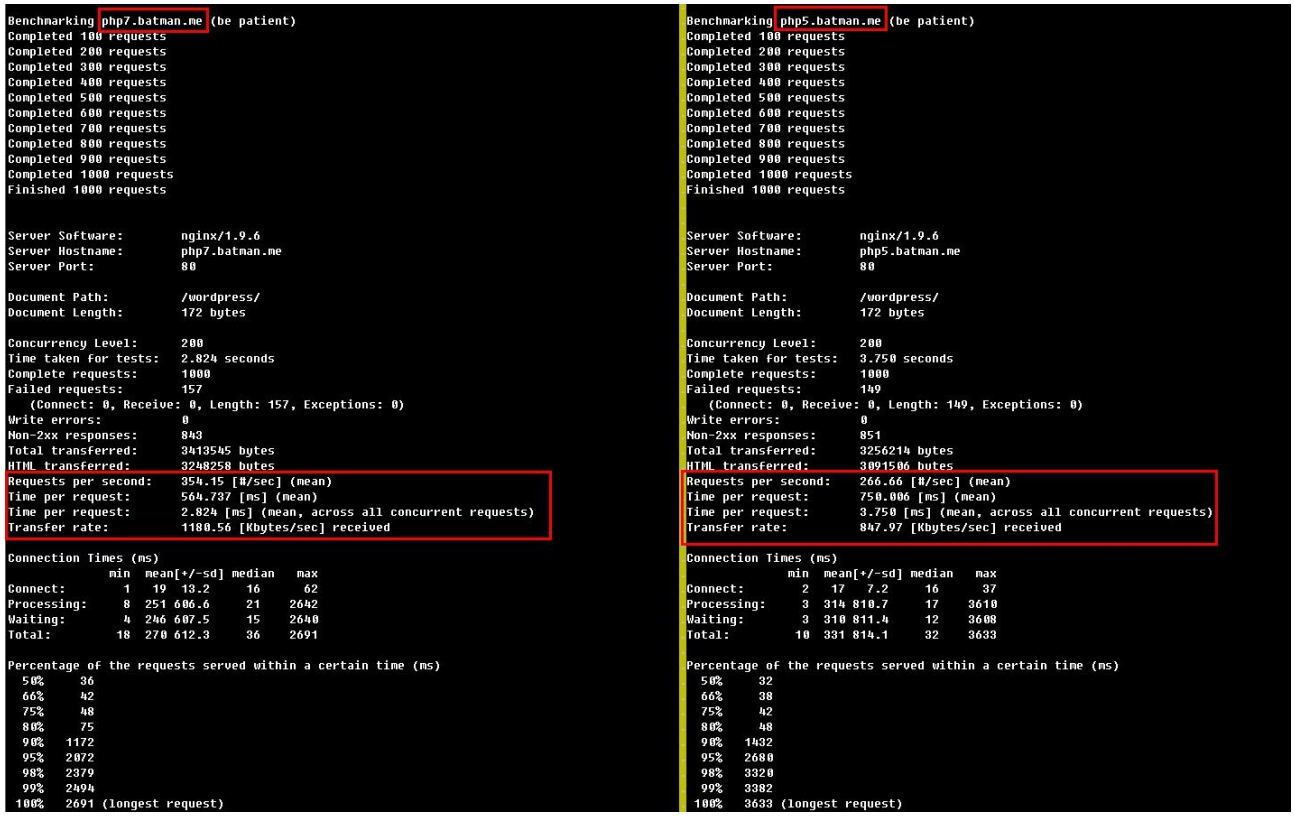


```
Benchmarking php7.batman.me (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        nginx/1.9.6
Server Hostname:        php7.batman.me
Server Port:            80

Document Path:          /wordpress/
Document Length:        172 bytes

Concurrency Level:      200
Time taken for tests:   2.756 seconds
Complete requests:      1000
Failed requests:        150
   (Connect: 0, Receive: 0, Length: 150, Exceptions: 0)
Write errors:           0
Non-2xx responses:      850
Total transferred:      3275750 bytes
HTML transferred:       3111100 bytes
Requests per second:    362.79 [#/sec] (mean)
Time per request:       551.281 [ms] (mean)
Time per request:       2.756 [ms] (mean, across all concurrent requests)
Transfer rate:          1160.56 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   15   8.4     13      42
Processing:     6  237 593.9     16    2561
Waiting:        5  233 594.3     13    2559
Total:         14  252 597.3     29    2586

Percentage of the requests served within a certain time (ms)
  50%     29
  66%     36
  75%     46
  80%     53
  90%   1111
  95%   2003
  98%   2401
  99%   2481
 100%   2586 (longest request)
```

```
Benchmarking php5.batman.me (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        nginx/1.9.6
Server Hostname:        php5.batman.me
Server Port:            80

Document Path:          /wordpress/
Document Length:        172 bytes

Concurrency Level:      200
Time taken for tests:   4.418 seconds
Complete requests:      1000
Failed requests:        140
   (Connect: 0, Receive: 0, Length: 140, Exceptions: 0)
Write errors:           0
Non-2xx responses:      860
Total transferred:      3079040 bytes
HTML transferred:       2915160 bytes
Requests per second:    226.37 [#/sec] (mean)
Time per request:       883.515 [ms] (mean)
Time per request:       4.418 [ms] (mean, across all concurrent requests)
Transfer rate:          680.66 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2   15   6.8     14      36
Processing:     8  406 1036.5     18    4347
Waiting:        0  402 1037.8     13    4344
Total:         15  421 1038.5     31    4355

Percentage of the requests served within a certain time (ms)
  50%     31
  66%     36
  75%     41
  80%     46
  90%   2173
  95%   3365
  98%   4023
  99%   4257
 100%   4355 (longest request)
```

原图 URL：

https://raw.githubusercontent.com/sunshine17/sunshine17.github.io/master/images/ab-1.jpg



```
Benchmarking php7.batman.me (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        nginx/1.9.6
Server Hostname:        php7.batman.me
Server Port:            80

Document Path:          /wordpress/
Document Length:        172 bytes

Concurrency Level:      200
Time taken for tests:   2.824 seconds
Complete requests:      1000
Failed requests:        157
   (Connect: 0, Receive: 0, Length: 157, Exceptions: 0)
Write errors:           0
Non-2xx responses:      843
Total transferred:      3413545 bytes
HTML transferred:       3248258 bytes
Requests per second:    354.15 [#/sec] (mean)
Time per request:       564.737 [ms] (mean)
Time per request:       2.824 [ms] (mean, across all concurrent requests)
Transfer rate:          1180.56 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        1   19  13.2     16      62
Processing:     8  251 606.6     21    2642
Waiting:        4  246 607.5     15    2640
Total:         18  270 612.3     36    2691

Percentage of the requests served within a certain time (ms)
  50%     36
  66%     42
  75%     48
  80%     75
  90%   1172
  95%   2072
  98%   2379
  99%   2494
 100%   2691 (longest request)
```

```
Benchmarking php5.batman.me (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        nginx/1.9.6
Server Hostname:        php5.batman.me
Server Port:            80

Document Path:          /wordpress/
Document Length:        172 bytes

Concurrency Level:      200
Time taken for tests:   3.750 seconds
Complete requests:      1000
Failed requests:        149
   (Connect: 0, Receive: 0, Length: 149, Exceptions: 0)
Write errors:           0
Non-2xx responses:      851
Total transferred:      3256214 bytes
HTML transferred:       3091506 bytes
Requests per second:    266.66 [#/sec] (mean)
Time per request:       750.006 [ms] (mean)
Time per request:       3.750 [ms] (mean, across all concurrent requests)
Transfer rate:          847.97 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        2   17   7.2     16      37
Processing:     3  314 810.7     17    3610
Waiting:        3  310 811.4     12    3608
Total:         10  331 814.1     32    3633

Percentage of the requests served within a certain time (ms)
  50%     32
  66%     38
  75%     42
  80%     48
  90%   1432
  95%   2680
  98%   3320
  99%   3382
 100%   3633 (longest request)
```

原图 URL：

https://raw.githubusercontent.com/sunshine17/sunshine17.github.io/master/images/ab-2.jpg

3. 新特性

3.1. 用 Closure::call()直接绑定函数 Context

PHP7 前，要绑定一个函数执行时的 context，需要先调用另一个函数执行绑定操作。相信熟悉 Nodejs 的朋友都会觉得这绑定操作很多余，因为在 js 这种 function as first class citizen 的语言里， function 作为对象在语言层面就已经可以在被调用时直接绑定这个对象的 this。在 javascript 里，很简单的 this 绑定：

```
function full_name(last_name){
        return this.first_name + '.' + last_name;
}
// below returns "Jack.London"
full_name.apply({first_name: 'Jack'}, ['London'])
```

PHP7 后，终于可以把这个中间步骤略去了。通过 Closure::call()实现：

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

// PHP7 前

$getXCB = function() {return $this->x;};
$getX = $getXCB->bindTo(new A, 'A'); // intermediate closure
echo $getX();

// PHP7 后

$getX = function() {return $this->x;};
echo $getX->call(new A);

// 会输出

// 1
// 1
```

3.2. Generator 的优化

- **加入 return 语句**

Generator 是 PHP5.5 开始加入的特性，当时在 Generator 里不能有 return 语句，若要取得最后一个元素只能由调用者检查是否已取得最后一个元素。PHP7 后，可以直接调用 Generator::getReturn()来获取最后一个元素。例如:

```
<?php
$gen = (function() {
    yield 1;
    yield 2;
    return 3;
})();

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

echo $gen->getReturn(), PHP_EOL;

// Output:
// 1
// 2
// 3
```

- Generator **委派**(delegation)

可以在 Generator 里委派另一个 Generator 来生成数据，简单直接的写法：

```
<?php
function gen() {
    yield 1;
    yield 2;
    yield from gen2();
}
function gen2() {
    yield 3;
    yield 4;
}
foreach (gen() as $val) {
    echo $val, PHP_EOL;
}
```

```
// ===== OUTPUT =====
// 1
// 2
// 3
// 4
?>
```

3.3. 引入静态类型语言的精粹

- **函数参数类型声明**(Scalar Type Hints)

在函数参数前面声明参数类型，默认不强制检查参数类型，若传参类型与声明不匹配，解释器会自动转换类型而不报任何错误与警告。但你也可以启用严格模式(strict mode)，让解释器执行传参类型检查 ,若类型不匹配则抛出 Fatal error: Uncaught TypeError。例如：

```
<?php
declare(strict_types = 1);   // 在 php 文件的第一行声明使用严格模式
function double(int $value){
    return 2 * $value;
}
$a = double("5");
var_dump($a);
```

- **函数返回值类型声明**(Return Type Hints)

在函数定义的括号后用: type 声明返回值类型，例如:

```
<?php
function a() : bool{
    return 1;
}
var_dump(a());
```

返回时会自动把 1 转换为 true，与上面的函数参数类型声明一样，在严格模式下会报错:

```
Fatal error: Uncaught TypeError: Return value of a() must be of the type boolean, integer returned
```

3.4. 匿名类

写惯java或python的朋友终于等到了这个在静态类型语言中很寻常的匿名类特性。PHP7中可通过 new class 的方式直接创建匿名类的实例(当你要抛出异常对象时可能会感受到这特性的好处——无须再定义整个具体的异常类了)：

```
<?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());
?>
```

上面会输出：

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

3.5. 可以用 define 来定义常量数组

还记得这个特性我在 5 年前就一直在期待中，PHP5.6 时，常量数组只能用 const 来定义，PHP7 后终于可以用 define()来定义了：

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

echo ANIMALS[1]; // outputs "cat"
```

3.6. 新的运算符

- **”<=>”运算符**

用于比较两个表达式：

```
  $a <=> $b  返回
     -1: if $a > $b
     0: if $a == $b
     1: if $a < $b
```

- **取默认值运算符：”??”**

从此可以打少好多字符。

in php5:

```
$a = isset($b) ? $b : 'default';
```

in php7:

```
$a = $b ?? 'default';

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

3.7. Session 选项配置更灵活

以往 session 的选项是在 php.ini 中配置的，PHP7 后可以把这些选项以数组的方式传参给 session_start()函数，例如：

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

3.8. use 声明可分组

在同一命名空间下 import 的类、函数、常量现在可以写在同一个 use 中 ,作为一个分组，直接看例子：

```
<?php
// PHP 7 之前
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+ 后
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};
```

4. 不兼容的地方

4.1. error 与 exception 的转变

很多 fatal 与 recoverable fatal error 被转换成 exception 了 ,这意味着你自定义的 error handler 不会被触发，因为不再有 error，只会抛 exception，exception 需要你 cache。

4.2. 构造函数失败时会抛异常

PHP7 前，某些 php 内部的类在构造失败时会返回 NULL 或一个不可用的对象，但现在会直接抛 Exception，你需要去 catch，否则会出错。

4.3. 改变了 E_STRICT 通知的严重性

所有的 E_STRICT 通知都被重新分类到其它级别，但 E_STRICT 这个常量依然保留，只是 error_reporting(E_ALL|E_STRICT)不会再产生任何错误。

4.4. 新的抽象语法树导致变量解释方式的不兼容

PHP7 使用了新的抽象语法树来解释源代码，这样可以提高了语言的表达性（一些以前很奇怪难懂的写法现在可以用很简单直观的方式来表达了），但同时也带来了一些向后兼容性的问题。

- **变量解释方向相反了**

非直接变量（变量、属性、方法）现在会严格地遵从“从左到右”的解释规则，而非以前的混合方式。 新旧两种间接表达式的解释方式对比:

Expression	PHP 5 interpretation	PHP 7 interpretation
$$foo['bar']['baz']	${$foo['bar']['baz']}	($$foo)['bar']['baz']
$foo->$bar['baz']	$foo->{$bar['baz']}	($foo->$bar)['baz']
$foo->$bar'baz'	$foo->{$bar['baz']}()	($foo->$bar)'baz'
Foo::$bar'baz'	Foo::{$bar['baz']}()	(Foo::$bar)'baz'

➢ list()的赋值顺序相反了

以前 list()是反向的顺序，现在按定义的顺序来赋值。例如：

```
<?php
list($a[], $a[], $a[]) = [1, 2, 3];
var_dump($a);
?>
```

PHP 5 会输出：

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

PHP 7 会输出：

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

```
  [1]=>
  int(2)
  [2]=>
  int(3)
}
```

- list()不允许无参调用 直接看例子，以下在 PHP7 中会报错：

```
<?php
list() = $a;
list(,,) = $a;
list($x, list(), $y) = $a;
?>
```

4.5. foreach 的重要变化

- foreach 传值调用操作是数组的副本。这意味着默认的情况下你在 foreach 里修改数据的数据不会影响到原数组。

- 传引用调用时会动态检测到数据的变化 简单地说，就是你在 foreach 里添加或删除数据元素，会直接影响到当前正在操作的数组，直接看代码：

```
<?php
$array = [0];
foreach ($array as &$val) {
    var_dump($val);
    $array[1] = 1;
}
?>
```

在 PHP 5 会输出：

```
int(0)
```

而 PHP 7 会输出：

```
int(0)
int(1)
```

4.6. yield 不再是函数，而是一个右关联的运算符

函数调用与表达式求值的区别无需多说，后者对执行效率的提升与语言表达能力的增强是很明显的。应用上看就是运算符不需要加函数调用括号()，例如：

```
<?php
echo yield -1;
// Was previously interpreted as
echo (yield) - 1;
// And is now interpreted as
echo yield (-1);

yield $foo or die;
// Was previously interpreted as
yield ($foo or die);
// And is now interpreted as
(yield $foo) or die;
?>
```