[PHP7开发体验报告 3](#_Toc438376255)

[1. 重要的写在前面 3](#_Toc438376256)

[2. Benchmark简图 3](#_Toc438376257)

[2.1. Wordpress性能测试 3](#_Toc438376258)

[2.2. Ab压力测试 4](#_Toc438376259)

[3. 新特性 6](#_Toc438376260)

[3.1. 用Closure::call()直接绑定函数Context 6](#_Toc438376261)

[3.2. Generator的优化 6](#_Toc438376262)

[ 加入return语句 6](#_Toc438376263)

[ Generator委派(delegation) 7](#_Toc438376264)

[3.3. 引入静态类型语言的精粹 8](#_Toc438376265)

[ 函数参数类型声明(Scalar Type Hints) 8](#_Toc438376266)

[ 函数返回值类型声明(Return Type Hints) 8](#_Toc438376267)

[3.4. 匿名类 9](#_Toc438376268)

[3.5. 可以用define来定义常量数组 9](#_Toc438376269)

[3.6. 新的运算符 10](#_Toc438376270)

[ "<=>"运算符 10](#_Toc438376271)

[ 取默认值运算符: "??" 10](#_Toc438376272)

[3.7. Session选项配置更灵活 11](#_Toc438376273)

[3.8. use声明可分组 11](#_Toc438376274)

[4. 不兼容的地方 12](#_Toc438376275)

[4.1. error与exception的转变 12](#_Toc438376276)

[4.2. 构造函数失败时会抛异常 12](#_Toc438376277)

[4.3. 改变了E\_STRICT通知的严重性 12](#_Toc438376278)

[4.4. 新的抽象语法树导致变量解释方式的不兼容 12](#_Toc438376279)

[ 变量解释方向相反了 12](#_Toc438376280)

[ list()的赋值顺序相反了 13](#_Toc438376281)

[4.5. foreach的重要变化 14](#_Toc438376282)

[4.6. yield不再是函数，而是一个右关联的运算符 15](#_Toc438376283)

# PHP7开发体验报告

运营中心：林志勇

# 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。

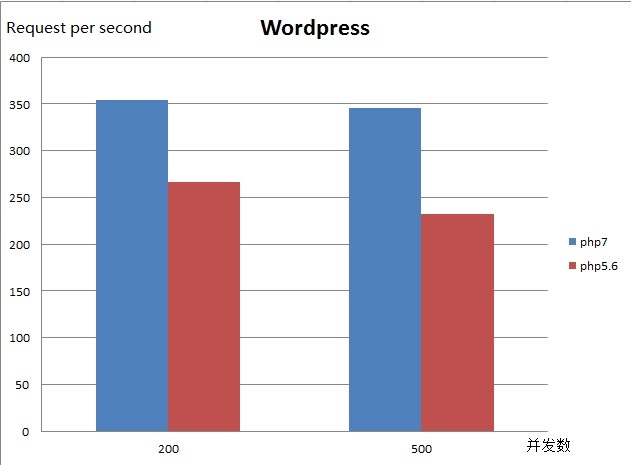
* 性能提升效果明显，在我的**老PC机上比5.6提升至少30%**
* 语法引入了时下主流的其它语言都有的特性，使开发效率得到明显的提升
* 不兼容旧版本的地方不多，基本不会触碰到

# 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》。

## Wordpress性能测试

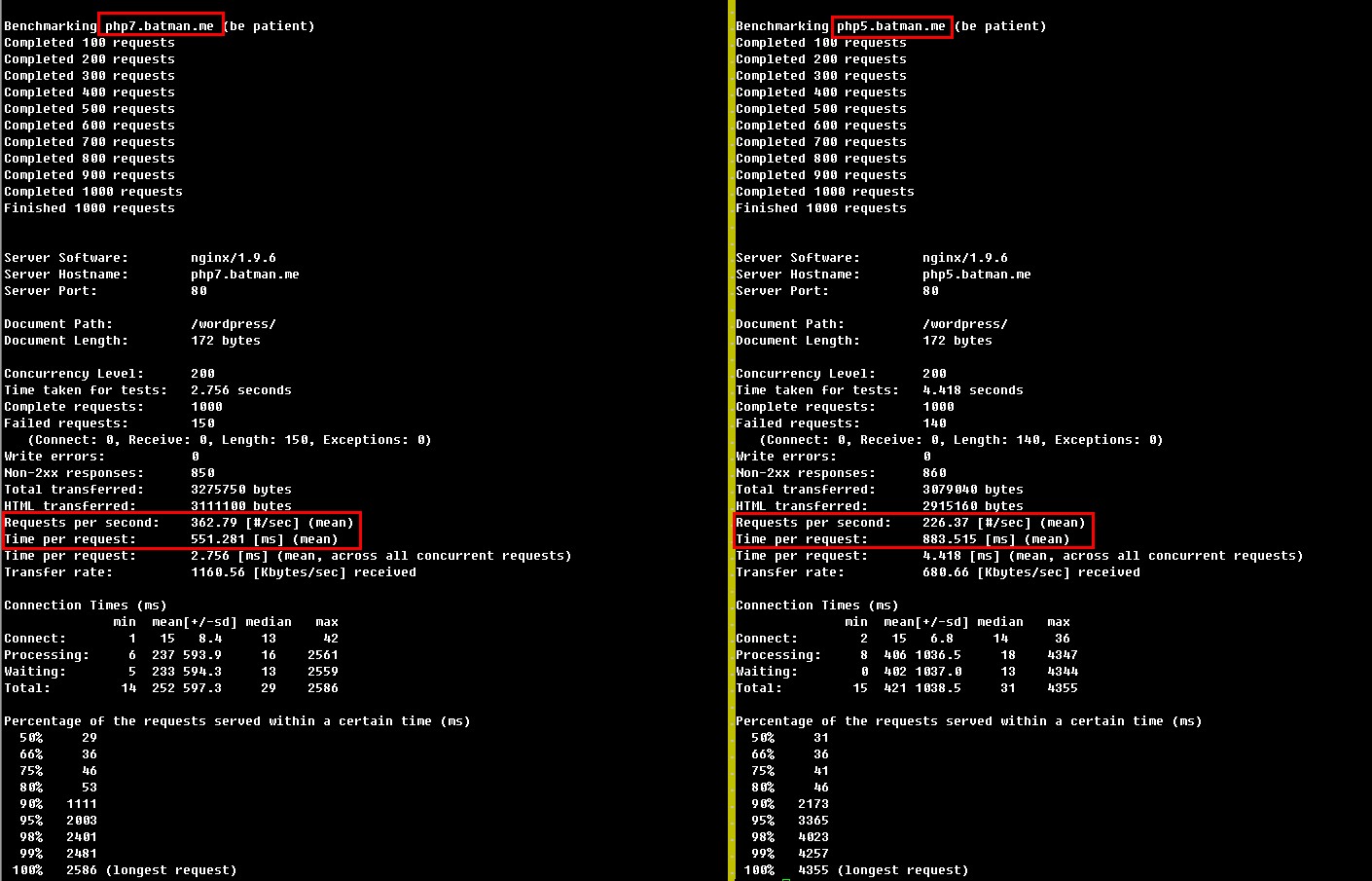
Y轴是Request Per Second，X轴是并发数



原图URL:

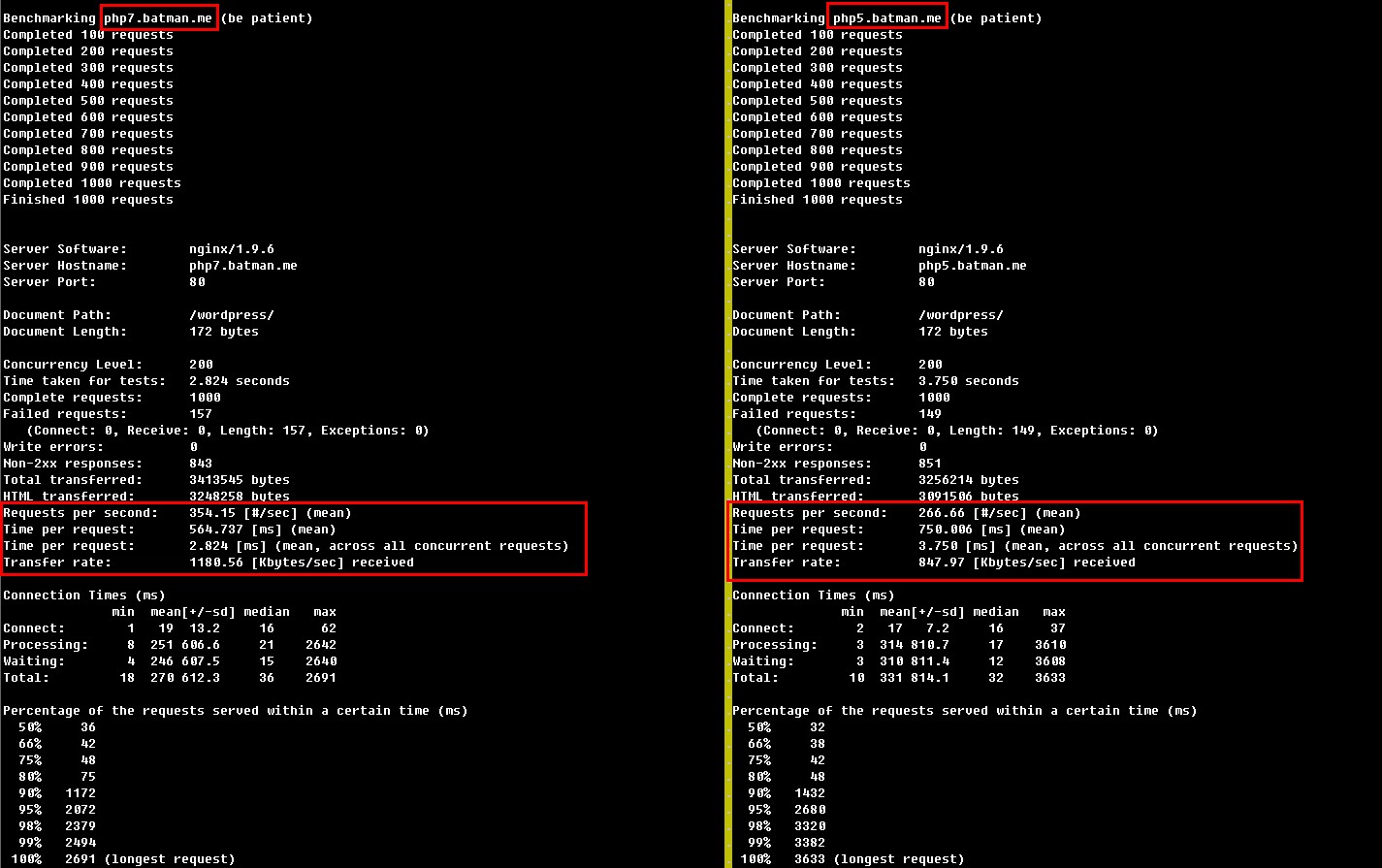
<https://raw.githubusercontent.com/sunshine17/sunshine17.github.io/master/images/statistic.jpg>

## Ab压力测试



原图URL:

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



原图URL:

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

# 新特性

## 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

## 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

?>

## 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

## 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) {

}

## 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"

## 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';

## 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,

]);

## 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};

# 不兼容的地方

## 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。

## 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。

## 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。

## 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'](https://github.com/sunshine17/sunshine17.github.io/blob/master) | $foo->{$bar['baz']}() | ($foo->$bar)['baz'](https://github.com/sunshine17/sunshine17.github.io/blob/master) |
| Foo::$bar['baz'](https://github.com/sunshine17/sunshine17.github.io/blob/master) | Foo::{$bar['baz']}() | (Foo::$bar)['baz'](https://github.com/sunshine17/sunshine17.github.io/blob/master) |

### 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;

?>

## 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)

## 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;

?>