**1、概述**

生成器是 PHP 5.5 引入的新特性，但是目测很少人用到它，其实这是个非常有用的功能。

生成器和迭代器有点类似，但是与标准的PHP迭代器不同，PHP生成器不要求类实现Iterator接口，从而减轻了类的开销和负担。生成器会根据需求每次计算并产出需要迭代的值，这对应用的性能有很大的影响：试想假如标准的PHP迭代器经常在内存中执行迭代操作，这要预先计算出数据集，性能低下；如果要使用特定方式计算大量数据，如操作Excel表数据，对性能影响更甚。此时我们可以使用生成器，即时计算并产出后续值，不占用宝贵的内存空间。

**2、创建生成器**

生成器的创建方式很简单，因为生成器就是PHP函数，只不过要在函数中一次或多次使用yield关键字。与普通的PHP函数不同的是，生成器从不返回值，只产出值。下面是一个简单的生成器实现：

function getLaravelAcademy() {

yield 'http://LaravelAcademy.org';

yield 'Laravel学院';

yield 'Laravel Academy';

}

很简单吧！调用此生成器函数时，PHP会返回一个属于Generator类的对象，这个对象可以使用foreach函数迭代，每次迭代，PHP会要求Generator实例计算并提供下一个要迭代的值。生成器的优雅体现在每次产出一个值之后，生成器的内部状态都会停顿；向生成器请求下一个值时，内部状态又会恢复。生成器内部的状态会一直在停顿和恢复之间切换，直到抵达函数定义体的末尾或遇到空的return语句为止。我们可以使用下面的代码调用并迭代上面定义的生成器：

foreach(getLaravelAcademy() as $yieldedValue) {

echo $yieldedValue, PHP\_EOL;

}

上面代码输出如下：

http://LaravelAcademy.org

Laravel学院

Laravel Academy

**3、使用生成器**

下面我们实现一个简单的函数用于生成一个范围内的数值，以此说明生成器是如何节省内存的。首先我们通过迭代器来实现：

function makeRange($length) {

$dataSet = [];

for ($i=0; $i<$length; $i++) {

$dataSet[] = $i;

}

return $dataSet;

}

$customRange = makeRange(1000000);

foreach ($customRange as $i) {

echo $i . PHP\_EOL;

}

此时执行会报错，提示超出单个PHP进程内存限制（要为100万个数字提供内存空间）：

memery-overflow-iterator

下面我们来改进实现方案，使用生成器实现如下：

function makeRange($length) {

for ($i=0; $i<$length; $i++) {

yield $i;

}

}

foreach (makeRange(1000000) as $i) {

echo $i . PHP\_EOL;

}

再次执行就可以毫无压力的打印出结果，因为生成器每次只需要为一个整数分配内存。

此外，一个常用的使用案例就是使用生成器迭代流资源（文件、音频等）。假设我们想要迭代一个大小为4GB的CSV文件，而虚拟私有服务器（VPS）只允许PHP使用1GB内存，因此不能把整个文件都加载到内存中，下面的代码展示了如何使用生成器完成这种操作：

function getRows($file) {

$handle = fopen($file, 'rb');

if ($handle == FALSE) {

throw new Exception();

}

while (feof($handle) === FALSE) {

yield fgetcsv($handle);

}

fclose($handle);

}

foreach ($getRows($file) as $row) {

print\_r($row);

}

上述示例一次只会为CSV文件中的一行分配内存，而不会把整个4GB的CSV文件都读取到内存中。

**4、总结**

生成器是功能多样性和简洁性之间的折中方案，生成器只是向前进的迭代器，这意味着不能使用生成器在数据集中执行后退、快进或查找操作，只能让生成器计算并产出下一个值。迭代大型数据集或数列时最适合使用生成器，因为这样占用的系统内存最少。生成器也能完成迭代器能完成的简单任务，而且使用的代码更少。

总而言之，生成器并没有为PHP添加新功能，不过使用生成器大大简化了某些任务，而且使用的内存更少，如果需要更多功能，例如在数据集中执行后退、快进以及查找功能，最好自己编写实现Iterator接口的类，或者使用PHP标准库（SPL）中某个原生的迭代器（<http://php.net/manual/spl.iterators.php>）。