python中yield的用法详解——最简单，最清晰的解释

原创 冯爽朗 于 2019-04-02 13:29:31 发布 678887 收藏 5757 版权
分类专栏： python

重要声明：本文为博主原创文章。转载请务必附上链接，谢谢！
本文链接：https://blog.csdn.net/mieleizhi0522/article/details/82142856

首先我要吐槽一下，看程序的过程中遇见了 yield 这个关键字，然后百度的时候，发现没有一个能简单的让我懂的，讲起来真TM的都是头头是道，什么参数，什么传递的，还口口声声说自己的教程是最简单的，最浅显易懂的，我就想问没有有考虑过读者的感受。

接下来是正题：

首先，如果你还没有对yield有个初步分认识，那么你先把yield看做“return”，这个是直观的，它首先是个return，普通的return是什么意思，就是在程序中返回某个值，返回之后程序就不再往下运行了。看做return之后再把它看做一个是生成器（generator）的一部分（带yield的函数才是真正的迭代器），好了，如果你对这些不明白的话，那先把yield看做return,然后直接看下面的程序，你就会明白yield的全部意思了：

```
def foo():
    print("starting...")
    while True:
        res = yield 4
        print("res:",res)
g = foo()
print(next(g))
print("*"*20)
print(next(g))
```

就这么简单的几行代码就让你明白什么是yield，代码的输出这个：

```
starting...
4
********************
res: None
4
```

我直接解释代码运行顺序，相当于代码单步调试：

1.程序开始执行以后，因为foo函数中有yield关键字，所以foo函数并不会真的执行，而是先得到一个生成器g(相当于一个对象)

2.直到调用next方法，foo函数正式开始执行，先执行foo函数中的print方法，然后进入while循环

3.程序遇到yield关键字，然后把yield想想成return,return了一个4之后，程序停止，并没有执行赋值给res操作，此时next(g)语句执行完成，所以输出的前两行（第一个是while上面的print的结果,第二个是return出的结果）是执行print(next(g))的结果，

4.程序执行print("*"*20)，输出20个*

5.又开始执行下面的print(next(g)),这个时候和上面那个差不多，不过不同的是，这个时候是从刚才那个next程序停止的地方开始执行的，也就是要执行res的赋值操作，这时候要注意，这个时候赋值操作的右边是没有值的（因为刚才那个是return出去了，并没有给赋值操作的左边传参数），所以这个时候res赋值是None,所以接着下面的输出就是res:None,

6.程序会继续在while里执行，又一次碰到yield,这个时候同样return 出4，然后程序停止，print函数输出的4就是这次return出的4.

到这里你可能就明白yield和return的关系和区别了，带yield的函数是一个生成器，而不是一个函数了，这个生成器有一个函数就是next函数，next就相当于“下一步”生成哪个数，这一次的next开始的地方是接着上一次的next停止的地方执行的，所以调用next的时候，生成器并不会从foo函数的开始执行，只是接着上一步停止的地方开始，然后遇到yield后，return出要生成的数，此步就结束。

****

```
def foo():
    print("starting...")
    while True:
        res = yield 4
        print("res:",res)
g = foo()
print(next(g))
print("*"*20)
print(g.send(7))
```

再看一个这个生成器的send函数的例子，这个例子就把上面那个例子的最后一行换掉了，输出结果：

```
starting...
4
********************
```

```
4 res: 7
        5 4
```

先大致说一下send函数的概念：此时你应该注意到上面那个的紫色的字，还有上面那个res的值为什么是None，这个变成了7，到底为什么，这是因为，send是发送一个参数给res的，因为上面讲到，return的时候，并没有把4赋值给res，下次执行的时候只好继续执行赋值操作，只好赋值为None了，而如果用send的话，开始执行的时候，先接着上一次（return 4之后）执行，先把7赋值给了res,然后执行next的作用，遇见下一回的yield，return出结果后结束。

5.程序执行g.send(7)，程序会从yield关键字那一行继续向下运行，send会把7这个值赋值给res变量

6.由于send方法中包含next()方法，所以程序会继续向下运行执行print方法，然后再次进入while循环

7.程序执行再次遇到yield关键字，yield会返回后面的值后，程序再次暂停，直到再次调用next方法或send方法。

这就结束了，说一下，为什么用这个生成器，是因为如果用List的话，会占用更大的空间，比如说取0,1,2,3,4,5,6.....1000

你可能会这样：

```
1 for n in range(1000):
2     a=n
```

这个时候range(1000)就默认生成一个含有1000个数的list了，所以很占内存。

这个时候你可以用刚才的yield组合成生成器进行实现，也可以用xrange(1000)这个生成器实现

yield组合：

```
1 def foo(num):
2     print("starting...")
3     while num<10:
4         num=num+1
5         yield num
6 for n in foo(0):
7     print(n)
```

输出：

```
1  starting...
2  1
3  2
4  3
5  4
6  5
7  6
8  7
9  8
10 9
11 10
```

xrange(1000):

```
1 for n in xrange(1000):
2     a=n
```

其中要注意的是python3时已经没有xrange()了，在python3中，range()就是xrange()了，你可以在python3中查看range()的类型，它已经是个<class 'range'>了，而不是一个list了，毕竟这个是需要优化的。

谢谢大家

如果你感觉对你有帮助，你的赞赏是对我最大的支持！