寻找自我的博客

Python yield 用法

分类: Python 2012-10-09 10:23 30人阅读 评论(0) 收藏 举报

yield 简单说来就是一个生成器,生成器是这样一个函数,它记住上一次返回时在函数体中的位置。对生成器函数的第二次(或第 n 次)调用跳转至该函数中间,而上次调用的所有局部变量都保持不变。

■ 生成器 是 一个函数

函数的所有参数都会保留

■ 第二次调用 此函数 时

使用的参数是前一次保留下的.

■ 生成器还"记住"了它在流控制构造

生成器不仅"记住"了它数据状态。 生成器还"记住"了它在流控制构造(在命令式编程中,这种构造不只是数据值)中的位置。由于连续性使您在执行框架间任意跳转,而不总是返回到直接调用者的上下文(如同生成器那样),因此它仍是比较一般的。

当你问生成器要一个数时,生成器会执行,直至出现 yield 语句,生成器把 yield 的参数给你,之后生成器就不会往下继续运行。 当你问他要下一个数时,他会从上次的状态开始运行,直至出现yield语句,把参数给你,之后停下。如此反复直至退出函数。

http://developer.51cto.com/art/201003/186451.htm

Python编程语言作为一款比较新的程序应用语言,其中有很多方法是开发人员需要慢慢熟练掌握的。比如今天为大家介绍的Python yield就是一个比较特殊的应用。yield的英文单词意思是生产,刚接触Python的时候感到非常困惑,一直没弄明白Python yield的用法。只是粗略的知道yield可以用来为一个函数返回值塞数据,比如下面的例子:

- 取出alist的每一项,然后把i + 1塞进去。然后通过调用取出每一项:
 - 1. alist
 2.
 - 3.

这的确是Python yield应用的一个例子,但是,看过limodou的文章《2.5版yield之学习心得》,并自己反复体验后,对yield有了一个全新的理解。

1. 包含vield的函数

假如你看到某个函数包含了yield,这意味着这个函数已经是一个Generator,它的执行会和其他普通的函数有很多不同。比如下面的简单的函数:

- 2
- 3.
- 4.

可以看到,调用h()之后,print 语句并没有执行!这就是yield,那么,如何让print 语句执行呢?这就是后面要讨论的问题,通过后面的讨论和学习,就会明白yield的工作原理了。

2. yield是一个表达式

Python2.5以前,Python yield是一个语句,但现在2.5中,yield是一个表达式 (Expression),比如:

1. m vield

表达式(yield 5)的返回值将赋值给m,所以,认为 m = 5 是错误的。那么如何获取 (yield 5)的返回值呢?需要用到后面要介绍的send(msg)方法。

3. 透过next()语句看原理

现在,我们来揭晓yield的工作原理。我们知道,我们上面的h()被调用后并没有执行,因为它有yield表达式,因此,我们通过next()语句让它执行。next()语句将恢复Generator执行,并直到下一个yield表达式处。比如:

- 1.
- 2.
- 3.
- 4.
- 6

调用后,h()开始执行,直到遇到yield 5,因此输出结果:

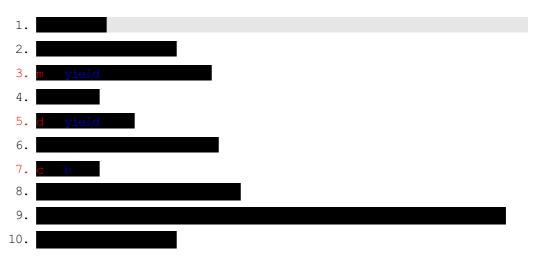
1.

当我们再次调用c.next()时,会继续执行,直到找到下一个yield表达式。由于后面没有 Python yield了,因此会抛出异常:

- 1.
- 2.
- 4. <module>
- 5.
- 6.

4. send(msg) 与 next()

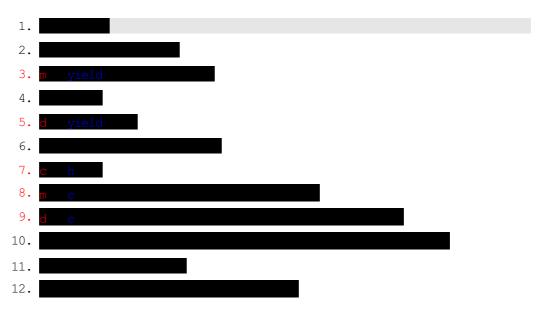
了解了next()如何让包含yield的函数执行后,我们再来看另外一个非常重要的函数 send(msg)。其实next()和send()在一定意义上作用是相似的,区别是send()可以传递yield 表达式的值进去,而next()不能传递特定的值,只能传递None进去。因此,我们可以看做 c.next()和 c.send(None)作用是一样的。来看这个例子:



需要提醒的是,第一次调用时,请使用next()语句或是send(None),不能使用send发送一个非None的值,否则会出错的,因为没有Python yield语句来接收这个值。

5. send(msg) 与 next()的返回值

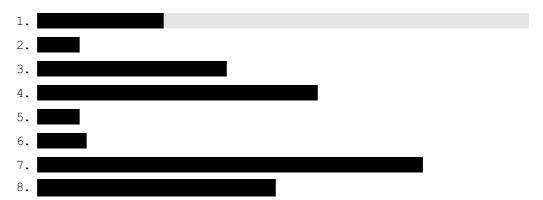
send(msg) 和 next()是有返回值的,它们的返回值很特殊,返回的是下一个yield表达式的参数。比如yield 5,则返回 5 。到这里,是不是明白了一些什么东西? 本文第一个例子中,通过for i in alist 遍历 Generator,其实是每次都调用了alist.Next(),而每次alist.Next()的返回值正是yield的参数,即我们开始认为被压进去的东东。我们再延续上面的例子:



6. throw() 与 close()中断 Generator

中断Generator是一个非常灵活的技巧,可以通过throw抛出一个GeneratorExit异常来终

止Generator。Close()方法作用是一样的,其实内部它是调用了throw(GeneratorExit)的。 我们看:



因此,当我们调用了close()方法后,再调用next()或是send(msg)的话会抛出一个异常:



以上就是我们对Python yield的相关介绍。