### 迭代器生成器

可以用for迭代的对象统称为可迭代对象（Iterable）。可以被next()函数调用并不断返回下一个值的对象称为迭代器（Iterator）。所有的可迭代对象可以通过iter()函数变成迭代器。

迭代器的特点是可以通过next()内置函数或者调用自身的\_\_next\_\_()方法来获取元素，当元素获取完则会抛出StopIteration异常。所以迭代器的元素只能使用一次，在迭代之后元素会被销毁，这也正是迭代器的一大魅力。

例：

>>> x = [1,2,3]

>>> y = iter(x)

>>> y

<list\_iterator object at 0x01021210>

>>> next(y)

1

>>> y.\_\_next\_\_ ()

2

>>> next(y)

3

>>> y.\_\_next\_\_ ()

Traceback (most recent call last):

File "<pyshell#85>", line 1, in <module>

y.\_\_next\_\_ ()

StopIteration

>>>

生成器（generator）也是一种迭代器。可以通过列表生成式的方式把列表解析的[]换成()，创建生成器对象。

例：

>>> l = [x\*x for x in range(5)]

>>> l

[0, 1, 4, 9, 16]

>>> g = (y\*y for y in range(3))

>>> g

<generator object <genexpr> at 0x036D7930>

>>> next(g)

0

>>> g.\_\_next\_\_ ()

1

>>> next(g)

4

>>> next(g)

Traceback (most recent call last):

File "<pyshell#13>", line 1, in <module>

next(g)

StopIteration

>>>

例子中的l是列表解析生成的列表对象，使用的for迭代range函数，元素以x\*x的形式组成一个新的列表对象。将中括号换成小括号生成的g就是生成器对象，使用next()函数或者\_\_next\_\_方法获取里面的值，当值被取完之后返回StopIteration异常。

在函数中使用yield关键字可以创建生成器函数，带有yield的函数就不再是普通函数。生成器函数可以生产一个无限的序列，调用生成器函数时会返回yeild后面的值。

例：

>>> def ge():

n=1

while True:

yield n

n+=1

>>> ge

<function ge at 0x035711E0>

>>> ge()

<generator object ge at 0x03567900>

>>> y = ge()

>>> next(y)

1

>>> next(y)

2

>>> next(y)

3

>>>

上面例子中的ge函数就是一个生成器，先创建个n值为1，然后是while True的死循环，循环里面使用关键字yield，yield n会让next()函数来取值时每次返回n的值，之后对n进行加1的操作。把ge()这个生成器对象赋值给y，用next()函数会每次返回一个n的值，因为里面是个死循环，所以这里面的值是无限的，也就不会出现StopIteration的异常。

生成器函数里面在没有return时会执行到没有值时返回的StopIteration的异常。但如果有return这个关键字，函数在执行的过程中遇到return 后就会出现StopIteration的异常，return后面的值也不再是程序的返回值而是StopIteration异常的说明。

例：

>>> def ge():

n=1

while True:

yield n

n+=1

return 'sfdsdffredsf'

>>> n = ge()

>>> next(n)

1

>>> next(n)

Traceback (most recent call last):

File "<pyshell#23>", line 1, in <module>

next(n)

StopIteration: sfdsdffredsf

>>>

在之前的例子中加了一个return后，在使用next()时，会将程序挂起在yield的位置，当下一次执行next()的时候会在yield的后一句开始执行，所以这里当执行到return时就返回了StopIteration的异常，而异常的说明则是return后面的表达式。

生成器的方法：

close() 手动关闭生成器函数，再次调用时就会返回StopIteration的异常。

Send()可以接受外部传入的一个变量，并根据变量内容计算结果后返回。首先要通过g.send(None)或者next启动生成器函数，并执行到第一个yield语句结束的位置。

throw() 用来向生成器函数送入一个异常。

例：

程序：

def ge():

n = 1

while True:

try:

m = yield n

n += m

except TypeError:

print('send int type')

运行后shell中：

>>> a = ge()

>>> next(a)

1

>>> a.send(12)

13

>>> a.send(12)

25

>>> a.send('123')

send int type

25

>>> a.throw(TypeError)

send int type

25

>>> a.close()

>>> a.send(12)

Traceback (most recent call last):

File "<pyshell#65>", line 1, in <module>

a.send(12)

StopIteration

>>>

这个例子中m = yield n 会使得send方法传进来的值赋给m，返回值是yield后面的n，然后让n与m相加后重新赋给n。这里首先要注意的是要使用send的方法这个生成器就必须处于开启的状态，可以是用send(None)或者next()方法来开启。那么如果传入的m不能与n相加则会报错，所以在循环中捕抓TypeError的异常。使用throw方法传入了异常那么就会直接返回这个一场，这里对异常做了处理，所以执行了except里面的语句。使用close()方法关闭生成器，如果再次使用时就会返回StopIteration的异常。