Python_Generator

1.生成器中的 next()操作当超出范围的时候就会产生一个 StopIteration 异常，for 函数是可以捕捉这类异常的。

2.生成器函数在每次暂停执行时，函数体内的所有变量都将被封存(freeze)在生成器中，并将在恢复执行时还原，并且类似于闭包，即使是同一个 生成器函数返回的生成器，封存的变量也是互相独立的。定义一个生成器来展示这个特点，这个例子中我们定义了一个生成器用于获取斐波那契数列。

```
def fibonacci():
    a=b=1
    yield a
    yield b
    while True:
        a, b = b, a+b
        yield b


for num in fibonacci():
    if num > 100: break
    print num,
```

输出为 1 1 2 3 5 8 13 21 34 55 89
因为生成器可以挂起，所以无限循环并没有关系

3.生成器是可以带参数的，因为生成器函数也是函数的一种。以下为 itertools.count 的源码（其他 itertools 模块中的迭代器都是有参数的）。

```
def count(start=0, step=1):
    # count(10) --> 10 11 12 13 14 ...
    # count(2.5, 0.5) -> 2.5 3.0 3.5 ...
    n = start
    while True:
        yield n
        n += step
```

send 是除 next 外另一个恢复生成器的方法。Python 2.5 中，yield 语句变成了 yield 表达式，这意味着 yield 现在可以有一个值，而这个值就是在生成器的 send 方法被调用从而恢复执行时，调用 send 方法的参数。

```
>>> def repeater():
...     n = 0
...     while True:
...         n = (yield n)
...
>>> r = repeater()
```

```
>>> r.next()
0
>>> r.send(10)
10
```

*调用 send 传入非 None 值前，生成器必须处于挂起状态，否则将抛出异常。所以为启动的生成器是不能传递非 None 值的，不过，未启动的生成器仍可以使用 None 作为参数调用 send。

*如果使用 next 恢复生成器，yield 表达式的值将是 None。

5.close():

这个方法用于关闭生成器。对关闭的生成器后再次调用 next 或 send 将抛出 StopIteration 异常。

6.throw(type, value=None, traceback=None):

这个方法用于在生成器内部（生成器的当前挂起处，或未启动时在定义处）抛出一个异常