生成器

1. 生成器

利用迭代器，我们可以在每次迭代获取数据（通过next()方法）时按照特定的规律进行生成。但是我们在实现一个迭代器时，关于当前迭代到的状态需要我们自己记录，进而才能根据当前状态生成下一个数据。为了达到记录当前状态，并配合next()函数进行迭代使用，我们可以采用更简便的语法，即**生成器(generator)。生成器是一类特殊的迭代器**。

2. 创建生成器方法1

要创建一个生成器，有很多种方法。第一种方法很简单，只要把一个列表生成式的 [ ] 改成 ( )

```
In [15]: L = [ x*2 for x in range(5)]

In [16]: L
Out[16]: [0, 2, 4, 6, 8]

In [17]: G = ( x*2 for x in range(5))

In [18]: G
Out[18]: <generator object <genexpr> at 0x7f626c132db0>

In [19]:
```

创建 L 和 G 的区别仅在于最外层的 [ ] 和 ( ) ， L 是一个列表，而 G 是一个生成器。我们可以直接打印出列表L的每一个元素，而对于生成器G，我们可以按照迭代器的使用方法来使用，即可以通过next()函数、for循环、list()等方法使用。

千锋Python人工智能学院

```
In [19]: next(G)
Out[19]: 0

In [20]: next(G)
Out[20]: 2

In [21]: next(G)
Out[21]: 4

In [22]: next(G)
Out[22]: 6

In [23]: next(G)
Out[23]: 8

In [24]: next(G)
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-24-380e167d6934> in <module>()
----> 1 next(G)

StopIteration:

In [25]:
In [26]: G = ( x*2 for x in range(5))

In [27]: for x in G:
   ....:     print(x)
   ....:
0
2
4
6
8

In [28]:
```

千锋Python人工智能学院

3. 创建生成器方法2

generator非常强大。如果推算的算法比较复杂，用类似列表生成式的 for 循环无法实现的时候，还可以用函数来实现。

我们仍然用上一节提到的斐波那契数列来举例，回想我们在上一节用迭代器的实现方式：

```
class FibIterator(object):
    """斐波那契数列迭代器"""
    def __init__(self, n):
        """
        :param n: int, 指明生成数列的前n个数
        """
        self.n = n
        # current用来保存当前生成到数列中的第几个数了
        self.current = 0
        # num1用来保存前前一个数，初始值为数列中的第一个数0
        self.num1 = 0
        # num2用来保存前一个数，初始值为数列中的第二个数1
        self.num2 = 1

    def __next__(self):
        """被next()函数调用来获取下一个数"""
        if self.current < self.n:
            num = self.num1
            self.num1, self.num2 = self.num2, self.num1+self.num2
            self.current += 1
            return num
        else:
            raise StopIteration

    def __iter__(self):
        """迭代器的__iter__返回自身即可"""
        return self
```

注意，在用迭代器实现的方式中，我们要借助几个变量(n、current、num1、num2)来保存迭代的状态。现在我们用生成器来实现一下。

千锋Python人工智能学院

```
In [30]: def fib(n):
   ....:     current = 0
   ....:     num1, num2 = 0, 1
   ....:     while current < n:
   ....:         yield num1
   ....:         num1, num2 = num2, num1+num2
   ....:         current += 1
   ....:     return 'done'
   ....:

In [31]: F = fib(5)

In [32]: next(F)
Out[32]: 1

In [33]: next(F)
Out[33]: 1

In [34]: next(F)
Out[34]: 2

In [35]: next(F)
Out[35]: 3

In [36]: next(F)
Out[36]: 5

In [37]: next(F)
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-37-8c2b02b4361a> in <module>()
----> 1 next(F)

StopIteration: done
```

千锋Python人工智能学院

在使用生成器实现的方式中，我们将原本在迭代器 `__next__` 方法中实现的基本逻辑放到一个函数中来实现，但是将每次迭代返回数值的return换成了yield，此时新定义的函数便不再是函数，而是一个**生成器**了。简单来说：**只要在def中有yield关键字的 就称为 生成器**

此时按照调用函数的方式(案例中为F = fib(5))使用生成器就不再是执行函数体了，而是会返回一个生成器对象（ 案例中为F ），然后就可以按照使用迭代器的方式来使用生成器了。

```
In [38]: for n in fib(5):
   ....:     print(n)
   ....:
1
1
2
3
5

In [39]:
```

但是用for循环调用generator时，发现拿不到generator的return语句的返回值。如果想要拿到返回值，必须捕获StopIteration错误，返回值包含在StopIteration的value中：

```
In [39]: g = fib(5)

In [40]: while True:
   ....:     try:
   ....:         x = next(g)
   ....:         print("value:%d"%x)
   ....:     except StopIteration as e:
   ....:         print("生成器返回值:%s"%e.value)
   ....:         break
   ....:
value:1
value:1
value:2
value:3
value:5
生成器返回值:done

In [41]:
```

千锋Python人工智能学院

总结

- 使用了yield关键字的函数不再是函数，而是生成器。（使用了yield的函数就是生成器）
- yield关键字有两点作用：
  - 保存当前运行状态（断点），然后暂停执行，即将生成器（函数）挂起
  - 将yield关键字后面表达式的值作为返回值返回，此时可以理解为起到了return的作用

- 可以使用next()函数让生成器从断点处继续执行，即唤醒生成器（函数）
- Python3中的生成器可以使用return返回最终运行的返回值，而Python2中的生成器不允许使用return返回一个返回值（即可以使用return从生成器中退出，但return后不能有任何表达式）。

4. 使用send唤醒

我们除了可以使用next()函数来唤醒生成器继续执行外，还可以使用send()函数来唤醒执行。使用send()函数的一个好处是可以在唤醒的同时向断点处传入一个附加数据。

例子：执行到yield时，gen函数作用暂时保存，返回i的值; temp接收下次c.send("python")，send发送过来的值，c.next()等价c.send(None)

```
In [10]: def gen():
   ....:     i = 0
   ....:     while i<5:
   ....:         temp = yield i
   ....:         print(temp)
   ....:         i+=1
   ....:
```

使用send

```
In [43]: f = gen()

In [44]: next(f)
Out[44]: 0

In [45]: f.send('haha')
haha
Out[45]: 1

In [46]: next(f)
None
Out[46]: 2

In [47]: f.send('haha')
haha
Out[47]: 3

In [48]:
```

千锋Python人工智能学院

使用next函数

```
In [11]: f = gen()

In [12]: next(f)
Out[12]: 0

In [13]: next(f)
None
Out[13]: 1

In [14]: next(f)
None
Out[14]: 2

In [15]: next(f)
None
Out[15]: 3

In [16]: next(f)
None
Out[16]: 4

In [17]: next(f)
None
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-17-468f0afdf1b9> in <module>()
----> 1 next(f)

StopIteration:
```

使用 `__next__()` 方法（不常使用）

千锋Python人工智能学院

```
In [18]: f = gen()

In [19]: f.__next__()
Out[19]: 0

In [20]: f.__next__()
None
Out[20]: 1

In [21]: f.__next__()
None
Out[21]: 2

In [22]: f.__next__()
None
Out[22]: 3

In [23]: f.__next__()
None
Out[23]: 4

In [24]: f.__next__()
None
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-24-39ec527346a9> in <module>()
----> 1 f.__next__()

StopIteration:
```

千锋Python人工智能学院