Python程序设计

第七讲 面向对象程序设计

再看迭代器

张 华
WHU

回顾

可迭代对象：迭代器和生成器

- 可循环迭代的对象称之为可迭代对象，迭代器和生成器函数是可迭代对象，Python提供了定义迭代器和生成器的协议和方法。
- 相对于序列，可迭代对象仅在迭代时产生数据，故可节省内存空间。
- Python语言提供了若干内置可迭代对象：
  - range、map、filter、enumerate、zip；
  - 标准库itertools模块中包含各种迭代器。
- 这些迭代器非常高效，且内存消耗小。

```
>>> ite = iter(range(10,100))
>>> next(ite)
10
```

可迭代对象和迭代器

- 可迭代对象（Iterable）
  - 实现了__iter__()的对象是可迭代对象（iterable）。
  - collections模块中定义了抽象基类Iterable，使用内置的isinstance，可判断一个对象是否为可迭代对象。
  - 序列对象都是可迭代对象，生成器函数和生成器表达式也是可迭代对象。
- 迭代器（Iterator）
  - 实现了__next()__的对象是迭代器，可以使用内置函数next()，调用迭代器的__next__()方法，依次返回下一个值。
  - 使用迭代器可以实现对象的迭代循环，迭代器让程序更加通用、优雅、高效，更加Python化。

迭代器协议

自定义可迭代对象和迭代器

- 声明一个类，定义__iter__方法和__next()__。
- 创建该类的对象，即是可迭代对象，也是迭代器。

```
class MyFib:
    def __init__(self):
        self.a = 0
        self.b = 1

    def __next__(self):
        self.a, self.b = self.b, self.a+self.b
        return self.a

    def __iter__(self):
        return self
```

```
fibs = MyFib()
for f in fibs:
    if f<100:
        print(f, end=' ')
    else:
        break
```

```
1 1 2 3 5 8 13 21 34 55 89
```

生成器函数

生成器函数

- 生成器函数使用yield语句返回一个值，然后保存当前函数整个执行状态，等待下一次调用next()。
- 生成器函数是一个迭代器，是可迭代对象，支持迭代。

```
>>> def int_stream(n):
        for i in range(n):
            yield i


>>> type(int_stream)
<class 'function'>
>>> istream = int_stream(10)
>>> type(istream)
<class 'generator'>
>>> from collections.abc import Iterator
>>> isinstance(istream, Iterator)
True
```

```
>>> next(istream)
0
>>> istream.__next__()
1
>>> [v for v in istream]
[2, 3, 4, 5, 6, 7, 8, 9]
>>> next(istream)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    next(istream)
StopIteration
```

生成器函数

- 生成器函数和普通函数的执行流程不一样。
  - 普通函数是顺序执行，遇到return语句或者最后一行函数语句就返回。
  - 而改成生成器的函数，在每次调用next()的时候执行，遇到yield语句返回，再次执行时从上次返回的yield语句处继续执行。

```
def odd():
    print('step 1')
    yield 1
    print('step 2')
    yield 3
    print('step 3')
    yield 5
```

```
>>> o = odd()
>>> next(o)
step 1
1
>>> next(o)
step 2
3
>>> next(o)
step 3
5
```

生成器函数

练习：

- 利用生成器函数创建Fibonacci数列。

```
def genFibs():
    a,b = 0,1
    while True:
        a,b = b,a+b
        yield a

for x in genFibs():
    if x<100:
        print(x, end=' ')
    else:
        break
```

```
1 1 2 3 5 8 13 21 34 55 89
```