

Python基础DAY11

一、迭代器

- 可迭代对象:有__iter__方法的对象就是可迭代对象,可重复取值
- 迭代器:有__next__方法的对象就是迭代器,最典型的迭代器就是for循环和文件句柄
- 迭代器也是一个可迭代对象
- __iter__():将一个可迭代对象转换成一个迭代器
- __next__():迭代器具有惰性机制,只有使用__next__()方法才能读取下一个元素
- 用__next__()获取元素时,不能超过迭代器的范围,否则会报错
- 应用场景:当内存较小,数据量大的时候选择迭代器,内存够大,数据量相对不大的时候选择可迭代对象
- 迭代器的优缺点:
 - 优点:节省内存
 - 缺点:
 - 1.不能灵活操作,无法直接查看元素,无法直接使用时间
 - 2.操作是一次性的
 - 3.无法逆向操作
 - 4.相对来说更加消耗时间
- for循环的本质:

```
s = "abcdefg"
s = s.__iter__()
while True:
    try:
        print(s.__next__())
    except:StopIteration:
        break
```

- 将可迭代对象转换成迭代器的两种方法:
 - 1. l = l.__iter__()
 l.__next__()
 python2中没有这种方法,python3才有
 - 2. l = iter(l)
 next(l)
 推荐这种用法,因为python2和python3中都有这种写法

二、函数构造的生成器

- 生成器的本质也是一个迭代器,与迭代器不同的是,迭代器是python自带的,生成器是程序员写的
- 生成器可以在函数中构造,也可以在表达式中构造
- 简单的函数构造的生成器:

```
def func()
```

- print(1)
- yield 5

```
g = func()  
print(g)
```

该函数只会打印func()的内存地址,且里面的函数体不会执行,同时会创建一个生成器对象

- 函数中return改成yield就会让该函数创建一个生成器
- yield和return的异同:
 - 相同:都会返回值
 - 不同:return会结束函数,后面的代码不会执行.yield只会暂时退出,后续若还有next方法就会继续执行后面的代码,所以可以用多个yield
- python的检查异常分为两种
 - 1.语法分析,即只检查语法是否符合规范,不管数据是否合理或已存在
 - 2.词法分析,会检查数据的合理性
- 上述简单的生成器不会执行print(1)也是基于以上原理
- 每次执行func()都会生成一个新的生成器,所以会有以下情况

```
392 # print(g)  
393 # print(next(g))  
394 1 def func(): 3  
395 4  
396 4 yield 3 10 12  
397 6 yield 2  
398 8 yield 1  
399 2 g = func()  
400 3 print(next(g))  
401 3 print(next(g))  
402 2 print(next(g))  
403 1 print(next(func()))  
404 1 print(next(func()))  
405 3
```

Run: hanshu x
D:\2019python\Scripts\python.exe C:/Users/W0/PycharmProjects/Python26/day09/hanshu.py
3
2
1
3
3
Process finished with exit code 0

- yield总结:
 - yield 能返回多个值,元组形式存储
 - 能够返回多种类型的数据
 - 一个函数里可以写多个yield并且都能执行
 - 能够记录执行位置,这也是yield最重要的作用
 - 后面如果不写内容,默认返回None
- yield后面无论是什么数据类型,都是一次性返回,例如下图:输出的就是整个列表

```
def func():  
    yield [1, 2, 3, 4]  
g = func()  
print(next(g))
```

- yield from:后面跟一个可迭代对象,这样就不会一次性全部返回,而是一次next返回其中的一个元素
- 区分迭代器和生成器:
 - 1.查看是否有send方法,有send方法就是生成器
 - 2.查看内存地址,直接print这个变量就是查看他的地址,不需要id(推荐)
 - 生成器一定是一个迭代器,迭代器不一定是生成器