PYTHON
编程基础

迭代器

概述

01 可直接使用for循环遍历的对象统称为可迭代对象（Iterable）。

02 迭代器（Iterator）是指有__iter__( )方法和__next__( )方法的对象，可以通过next函数不断获取下一个值，并不是所有的可迭代对象都是迭代器。

03 可以使用isinstance方法判断一个对象是否是可迭代对象或迭代器。

概述


```
from collections.abc import Iterable,Iterator #导入Iterable和Iterator类型
ls=[1,2,3,4,5] #创建一个列表对象
g=(x for x in range(1,6)) #创建一个生成器
print('ls是可迭代对象：',isinstance(ls,Iterable)) #True
print('g是可迭代对象：',isinstance(g,Iterable)) #True
print('ls是迭代器：',isinstance(ls,Iterator)) #False
print('g是迭代器：',isinstance(g,Iterator)) #True
```

iter函数

01 ▶ 对于可迭代对象，可以通过iter函数得到迭代器。

例：

```
from collections.abc import Iterator #导入Iterator类型
ls=[1,2,3,4,5] #创建一个列表对象
it=iter(ls) #利用iter函数获取ls的迭代器
print('it是迭代器：',isinstance(it,Iterator))
```

```
it是迭代器： True
```

next函数

01

对于迭代器，则可以使用next函数不断获取下一个元素，当所有元素都获取完毕后再调用next函数，就会引发StopIteration异常。

例：

```
g=(x for x in range(1,3)) #创建一个生成器
print('第1个元素：',next(g))
print('第2个元素：',next(g))
#print('第3个元素：',next(g)) #取消前面的注释则会引发StopIteration异常
```

```
第1个元素： 1
第2个元素： 2
```

自定义迭代器

01

如果我们要自己定义一个类，使得该类的对象是迭代器，则需要在类中实现__next__和__iter__这两个方法。

02

例：

```
from collections.abc import Iterator
class Faclist: #定义Faclist类
    def __init__(self): #定义构造方法
        self.n=1
        self.fac=1
```

自定义迭代器

```
    def __next__(self): #定义next方法
        self.fac*=self.n
        self.n+=1
        return self.fac
    def __iter__(self): #定义__iter__方法
        return self
if __name__=='__main__':
    facs=Faclist()
    print('facs是迭代器：',isinstance(facs,Iterator))
    for i in range(1,6): #i在1至5范围依次取值
        print('第%d个元素：'%i,next(facs))
```

```
facs是迭代器： True
第1个元素： 1
第2个元素： 2
第3个元素： 6
第4个元素： 24
第5个元素： 120
```