《Python使用教程系列--高阶函数Map，Filter，Reduce》

**（带你一探Python中的map，filter，reduce函数）**

上次推文我们介绍了python中的logging模块的相关知识，这次推文我们将学习一下python中的高阶函数等相关的知识，这些高阶函数我们是非常常见的，比如我们经常使用的Map，Filter，Reduce。

1. **定义**

在学习python的基础知识的时候，我们可能会学到一个概念“函数式编程”，我们来看看百度百科的介绍：

**"函数式编程"是一种"编程范式"（programming paradigm），也就是如何编写程序的方法论。它属于"结构化编程"的一种，主要思想是把运算过程尽量写成一系列嵌套的函数调用。**

也就是说函数式编程有这样的特点：

（1）函数可作为对象可以赋值给变量。

（2）函数可作为参数传递给另一个函数。

（3）函数可作为一个参数返回。

那么高阶函数的定义是什么呢？简单的来说，只要满足一下的条件就可以认为这个函数是高阶函数：

**•** 函数可以作为参数传给另外一个函数；

**•** 函数的返回值为另外一个函数；

当然了，若一个函数的返回值为该函数本身的话，很显然就这就是我们常见的递归。

我们来看看几个简单的例子：

*# 例1 函数作为参数传给另外一个函数*

def print\_1():  
 print(**"打印1"**)  
def print\_2(print\_1):  
 *# 调用print\_1()函数* print\_1()  
 print(**"打印2"**)  
 print(**"打印结束"**)

*# 函数调用，参数为print\_1()函数*  
print\_2(print\_1)

*#例2*

*#返回值为一个函数*def print\_1():  
 print(**"我爱python"**)  
def print\_2():  
 print(**"我爱python知识学堂"**)  
 *# 返回值为一个函数* return(print\_1())  
*# 函数调用*  
print\_2()

1. **Map**

map()是python内置函数，它会根据所传递的函数对指定的序列（可迭代）做映射，原型如下：

map(func, \*iterables) --> map object

**参数function：**是一个函数，是自定义或者python内置的函数都可以。

**参数**\***iterable：**是可迭代的对象，比如我们常用的列表，元组等。

**返回值map object：**表示map函数的返回值是一个map对象。

简单的来说这个Map函数的功能就是使用func对传入的iterable的每一个元素(比如列表的每一个元素)进行处理，返回对象如<map object at 0x00000272BB702128>

注意到参数iterable是加了符号“\*”的，所以说这个参数的意思是可以接受多个可迭代的对象的。接下来我们看几个例子：

**Function为内置函数**

string\_1 = **'Python知识学堂'** *#字符串*string\_2 = [1,2,3,4,5,6] *#列表*  
string\_3 = {**'python'**:2,**'学习'**:3,1:4} *#字典*  
res1 = map(str,string\_1)  
res2 = map(str,string\_2)  
res3 = map(str,string\_3)  
  
print(string\_1)  
print(list(res1))  
print(list(res2))  
print(list(res3))

上述例子使用python的内置函数str()将字符串和列表的每个元素变成了str类型，根据map函数的说明我们知道res1和res2是一个Iterator，Iterator是惰性可迭代序列，因此通过list()函数将值求出来，注意，map不改变原list。

**关于迭代器的惰性计算，这里先引用网上的一句话：“迭代器的一个优点就是它不要求你事先准备好整个迭代过程中所有的元素。迭代器仅仅在迭代至某个元素时才计算该元素，而在这之前或之后，元素可以不存在或者被销毁。这个特点使得它特别适合用于遍历一些巨大的或是无限的集合，比如几个G的文件，或是斐波那契数列等等。这个特点被称为延迟计算或惰性求值(Lazy evaluation)。”**

**Function为自定义函数**

上述使用的是python的内置函数str()，实际的项目中我们更多的是传入自定义的函数：

def self\_test(x):  
 return x + 2  
string = [1,2,3,4,5,6]  
res = map(self\_test, string)  
print(list(res))

*#输出 [3, 4, 5, 6, 7, 8]*

上述代码中 向map函数中传入了自定义的self\_test函数，函数加对传入的元素加2的处理，理解起来还是很简单的。

**多个Iterable**

之前说过，参数\*iterable可以是多个可迭代对象的，我们来一探究竟：

def func(x,y,z):  
 return x\*\*2, y\*\*2, z\*\*2  
List1 = [1, 2]  
List2 = [1, 2, 3, 4]  
List3 = [1, 2, 3, 4, 5]  
res = map(func, List1, List2, List3)  
print(list(res))

*#输出 [(1, 1, 1), (4, 4, 4)]*

可以看出，map()函数中传入了多个iterabl。输出的结果中列表的长度为2。这是为什么？很简单，这里有一个知识点：输出的list的长度取决于\*iterable中传入的litrable的最小长度，比如在上述中List1、List2和List3中最短的为List1，长度为2。

**For循环取内容**

我们知道，map()函数的返回为一个map对象，如<map object at 0x00000272BB702128>

我们之前的操作是使用list输出里面的内容的，实际上我们也可以使用for循环遍历的方式取值：

*#for循环来取出内容*def add\_test(x):  
 return x + 2  
string = [1,2,3,4,5,6]  
res = map(add\_test, string)  
res\_ls=[]  
for i in res:  
 res\_ls.append(i)  
print(res\_ls)

输出的结果跟上述一样，就不多分析了，有的小伙伴学习到这，就在想我们自己该怎么实现

map()函数的功能呢？其实也不太难，我们接着学：

**自实现Map()功能**

def add\_test(x):  
 return x + 2  
*# 实现map()函数功能*def self\_map(function,iterable):  
 str\_1=[]  
 for each in iterable:  
 each\_num = function(each)  
 str\_1.append(each\_num)return str\_1.\_\_iter\_\_()  
  
string = [1,2,3,4,5,6]  
result = self\_map(add\_test,string)  
print(list(result))

上述代码的输出为：[3, 4, 5, 6, 7, 8]，且完成了map函数的功能了。小伙伴们可能对代码代码str\_1.\_\_iter\_\_()比较好奇，这个代码的功能就是将str\_1转换为迭代器对象。

map()函数我们就介绍到这，接下来来看另一个高阶函数filter()

1. **Filter**

看到这个函数的名字大家就知道其功能大概就是来过滤一些什么的，那具体是怎样的呢？

filter()是python内置函数，它会根据所传递的函数对指定的对象（可迭代）做过滤，原型如下：

filter(function or None, iterable) --> filter object

可以看出该函数接收两个参数，第一个为函数，第二个为可迭代对象，序列的每个元素作为参数传递给函数进行判断，然后返回 True 或 False，最后将返回 True 的元素放到新列表（或迭代器）中。

**Filter()实例**

我们来看实例，注意filter()函数的返回时filter可迭代的对象，如：<filter object at 0x0000023D19128E48>，所以在输出的时候需要使用list()函数转换一下，当然了也可以使用for循环来取值。

def select\_element(string):  
 if **'p'** in string:  
 return True  
 *# return 1* else:  
 return False  
 *# return 0*res\_1 = filter(select\_element, [**'1'**,**'2'**,**'3'**,**'python'**])  
res\_2 = filter(select\_element, {**'python'**:1,**'知识'**:2,**'学堂'**:3})  
print(list(res\_1))  
print(list(res\_2))

*# 输出均为  
# ['python']  
# ['python']*

**自实现Filter()功能**

跟map()一样，我们自己来实现filter()的功能：

*# 自实现filte()函数功能*def select\_element(string):  
 if **'p'** in string:  
 return True  
 *# return 1* else:  
 return False  
 *# return 0*def filter\_test(function,iterable):  
 str\_1=[]  
 for each in iterable:  
 if function(each):  
 str\_1.append(each)  
 return str\_1  
  
string = [**'1'**,**'2'**,**'3'**,**'python'**]  
res1 = filter\_test(select\_element,string)  
res2 = filter\_test(lambda x:x==**'python'**,string)  
print(res1)  
print(res2)

输出结果跟之前的一样，其中res2 = filter\_test(lambda x:x==**'python'**,string)中使用了lambda函数，我们以后再具体介绍这类函数的使用。filter()函数我们就介绍到这，接下来来看另一个高阶函数reduce ()。

1. **Reduce**

跟map()、filter()类似，reduce()是一个以函数以及sequence为参数的高阶函数，**其返回值为一个value值而不是迭代器对象。**其原型如下：

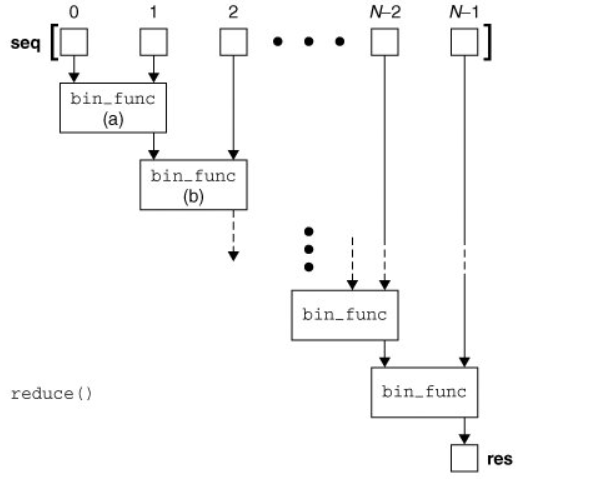
reduce(function, sequence[, initial]) -> value

但reduce()传入的函数必须接收两个参数，比如func (x,y)满足条件，但是func(x,y,z)并不满足条件。

**Initial为初始化的一个参数，可不填。（如果没有指定Initial的值那么其为sequence的第一个元素的值）。**

**Reduce()实例**

在看reduce()的函数的实例之前，我们来看一个图解释一下reduce的执行过程：



看图很简单：reduce每一次迭代，都将上一次的迭代结果与下一个元素一起传入function函数中，取得值以后将其与下一个元素一起继续下传…找到使用到sequence中的最后一个元素。

我们来看一个使用reduce()函数的实例：

*# 需要functools从导入*

from functools import reduce  
def func(x, y):  
 return x + y  
string = [1, 3, 5, 7, 9]  
result\_1 = reduce(func,string)  
result\_2 = reduce(func,string,100)  
print(result\_1)  
print(result\_2)

上述代码输出为：25和125，result\_2使用initial = 100。

其中25的值可以这样理解：25 = ((((1+3)+5)+7)+9)

其中125的值可以这样理解：125 = (((((100+1)+3)+5)+57)+9)

其实上述reduce的计算过程还是比较好理解的，其实在Python中还有一个函数sum()，可以直接求职的，具体的求值语句：res = sum(string)。

**自实现Reduce()功能**

Reduce函数自定义实现也比较简单，我们来看看这个实现怎么写：

def func(x, y):  
 return x + y

def self\_reduce(function, string, initializer=None):  
 it = iter(string)  
 if initializer is None:  
 value = next(it)  
 else:  
 value = initializer  
 for each\_ele in it:  
 value = function(value, each\_ele)  
 return value  
  
string = {1, 3, 5, 7, 9}  
*# string = [1, 3, 5, 7, 9]*result\_1 = self\_reduce(func,string)  
result\_2 = self\_reduce(func,string,100)  
  
print(result\_1)  
print(result\_2)

输出结果跟上述一样，这里的string = {1, 3, 5, 7, 9}是一个集合，即传入的是一个集合，当然了string = [1, 3, 5, 7, 9]也是可以的。

或者使用以下的方法实现Reduce()方法的功能也是可以的：

def func(x, y):  
 return x + y  
  
def self\_reduce(function, iterable, initializer=None):  
 if initializer is None:  
 value =iterable.pop(0)  
 else:  
 value=initializer  
 for each\_ele in iterable:  
 value=function(value,each\_ele)  
 return value  
  
string = {1, 3, 5, 7, 9}  
result\_1 = self\_reduce(func,string,100)  
  
print(result\_1)

输出结果为：125

1. **总结**

以上就是本次推文的所有内容啦！主要讲的是Python中的高阶函数，具体的内容涉及到高阶函数的定义和概念，具体的讲解了三个常用的高阶函数：map,filter,reduce的使用方法，同时也讲解了如何自己实现这三者的功能，这个自己实现三者功能的方法大家可以好好研究下。

在实际的工程项目中，运用好这些方法，不仅会减少代码量，而且在很大的程度上还能加速代码的运行速度。

最后，感谢大家的关注，如果觉得作者写得还不错的话，欢迎三连点赞分享再看支持，您的支持是我写作的最大动力，谢谢！