**Python中的闭包怎么理解**

上次的文件中我们学习了装饰器的基础知识，具体可以才能够以下链接进行查看：

**装饰器系列(3)-基于类的装饰器实现**

由于在后台有小伙伴问到了闭包的相关知识，这里我就稍微简单的讲述一下闭包的相关知识，大家一起复习或学习一下。

1. **什么是闭包**

首先需要解答的第一个问题就是，什么是闭包。我们先引用百度百科中的解释，尽管你可能不是很同意百度的说法：**闭包就是能够读取其他函数内部变量的函数。例如在javascript中，只有函数内部的子函数才能读取[局部变量](https://baike.baidu.com/item/%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F/9844788?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%97%AD%E5%8C%85/_blank)，所以闭包可以理解成“定义在一个[函数](https://baike.baidu.com/item/%E5%87%BD%E6%95%B0/301912?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E9%97%AD%E5%8C%85/_blank)内部的函数“。在本质上，闭包是将函数内部和函数外部连接起来的桥梁。**

**大家也可以参考维基百科的定义哈。**

总结一下就是：如果在外部函数中定义了一个内部函数，并且内部函数能够读取到外部函数内的变量，这个定义在内部的函数我们就称为闭包，仔细想一些这不就是装饰器么，我们先解释道这，举一个例子：

def fun\_1(x):

y = 100

def inner\_fun():

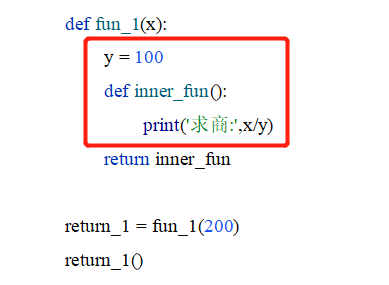
print('求商:',x/y)

return inner\_fun

return\_1 = fun\_1(200)

return\_1()

输出的结果就是2.0，这里我们稍微解释一下：上述案例中inner\_fun就被包括在函数fun\_1中，那么我们可以肯定的是fun\_1内部定义的变量(如y=100)，对于inner\_fun函数来说就就是可以调用的，反之则不成立。这就好比父类的资产子类是可以使用的那样，可能不是很恰当，但是就是这个意思。那么闭包就是下述标记部分：



其中y=100,我们就叫其“自由变量”

1. **闭包的理解**

对闭包的理解看来大家都清楚了，那么我们来进一步理解一下闭包：

我们先看这样的一个例子：

def fun\_1(x):

def fun\_2(y):

return x/y

return fun\_2

result = fun\_1(100)

print(result(200))

print(result(300))

可以看出输出了两个不同的值，这两个值都是基于一个100的变量进行计算的，而我们的这个100一直没有改变，只是传入了不同的参数。试想一下，如果fun\_1()是一个不常变动的接口，当我们对内部函数使用不同的参数就可以完成不同的对象的初始化，是不是很有效。

我们来打印一些输出：

print(fun\_1.\_\_closure\_\_)  
print(result.\_\_closure\_\_)  
print(type(result.\_\_closure\_\_))  
print(result.\_\_closure\_\_[0].cell\_contents)

print(result.\_\_code\_\_.co\_freevars)

输出为：

None

(<cell at 0x00000200D3CB5FD0: int object at 0x00000200AA5955D0>,)

<class 'tuple'>

100

('x',)

这个输出是什么意思呢？每一个函数都有一个\_\_closure\_\_属性，如果输出是一个cell对象的元组（tuple），那么这个函数就是一个闭包，使用闭包的cell\_cntents属性可以获取相应的相应自由变量的值，这里是100。另外我们可以通过\_\_code\_\_属性（编译后的函数定义体）来查看保存的局部变量和自由变量的名称，这里打印出来的是('x',)。

我来看一个变量作用域的问题,给出下面的例子：

def Test():  
 var = 10  
 def my\_add(x):  
 var += 1  
 return var + x  
 return my\_add  
my\_add = Test()  
res = my\_add(100)  
print(res)

这个代码会报“UnboundLocalError: local variable 'var' referenced before assignment”的错误，分析一下原因就是：在运行到语句var + =1 的时候，实际是一个赋值语句var = var + 1这会将var变成局部变量(这种赋值语句会创建局部变量count)，那么对变量var进行赋值由于var是外部函数Test的私有变量，对于内部函数my\_add来说是是不可见的，由于my\_add函数中没有这个变量因此会报错，我们可以这样进行修改：

def Test():  
 var = 10  
 def my\_add(x):  
 nonlocal var  
 var += 1  
 return var + x  
 return my\_add  
my\_add = Test()  
res = my\_add(100)  
print(res)

这里使用Python中的关键字nonlocal 将变量var标记为自由变量，这样使得局部变量var=10在函数外部访问就变成了可能，也通过如此完成了计算。或者改写一下，使用global函数也是可以的：

var = 10  
def Test():  
 def my\_add(x):  
 global var  
 var += 1  
 return var + x  
 return my\_add  
  
my\_add = Test()  
res = my\_add(100)  
print(res)

最后我们来看一个代码：

def func\_out(n):  
 i = 1  
 def func\_in():  
 nonlocal i  
 i = i + n  
 print(i)  
 return func\_in  
  
loop = 0  
start = func\_out(3)  
while loop<3:  
 start()  
 loop += 1

输出是这样的：

4

7

10

这个怎么理解呢？我总结一下就是：函数执行后,变量的改变会一直保存在内存流中,再次调用时是改变的值。

那么如果我们想讲这个局部变量能够长久的保存在内存中，那么就可以用闭包来实现这个功能，但是也要注意的是，这种做法会在一定的程度上造成内存的溢出。

1. **参考文献**

（1）《流畅的Python》-Luciano Ramalho

（2）https://stackoverflow.com/questions/36636/what-is-a-closure