闭包

函数只是一段可执行代码，编译后就“固化”了，每个函数在内存中只有一份实例，得到函数的入口点便可以执行函数了。函数还可以嵌套定义，即在一个函数内部可以定义另一个函数，有了嵌套函数这种结构，便会产生闭包问题。

函数嵌套

在函数里面还可以定义函数，可以嵌套多层，执行需要被调用。

```
def outer():
    print('outer----hello')
    def inner():  # inner这个函数是在outer函数内部定义的
        print('inner----hello')
    inner()  # inner函数只在outer函数内部可见

outer()
# inner()  这里会报错，在outer函数外部无法访问到inner函数
```

什么是闭包

闭包是由函数及其相关的引用环境组合而成的实体(即：**闭包=函数块+引用环境**)。

```
def outer(n):
    num = n
    def inner():
        return num+1
    return inner

print(outer(3)())  # 4
print(outer(5)())  # 5
```

在这段程序中，函数 inner 是函数 outer 的内嵌函数，并且 inner 函数是outer函数的返回值。我们注意到一个问题：内嵌函数 inner 中引用到外层函数中的局部变量num，Python解释器会这么处理这个问题呢？ 先让我们来看看这段代码的运行结果，当我们调用分别由不同的参数调用 outer 函数得到的函数时，得到的结果是隔离的(相互不影响)，也就是说每次调用outer函数后都将生成并保存一个新的局部变量num,这里outer函数返回的就是闭包。 **如果在一个内部函数里，对在外部作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包(closure).**

修改外部变量的值

闭包里默认不能修改外部变量。

```
def outer(n):
    num = n
    def inner():
        num = num + 1
        return num
    return inner

print(outer(1)())
```

上述代码运行时会报错!

`UnboundLocalError: local variable 'num' referenced before assignment`

原因分析

在python里，只要看到了赋值语句，就会认为赋值语句的左边是一个局部变量。`num = num + 1` 这段代码里，`num` 在 `=` 的左边，python解析器会认为我们要修改 `inner` 函数里 `num` 这个局部变量，而这个变量使用之前是未声明的，所以会报错。

解决方案

我们分析过，报错的原因在于当我们在闭包内修改外部变量时，会被python解析器误会为内部函数的局部变量。所以，解决方案就在于，我们需要想办法，让解析器知道我们不是要修改局部变量，而是要修改外部变量。

- 解决方法：使用 nonlocal 关键字

```
def outer(n):
    num = n
    def inner():
        nonlocal num  # 修改前使用nonlocal关键字对 num 变量进行说明
        num = num + 1
        return num
    return inner

print(outer(2)())
```