9.23 在局部变量域中执行代码¶

问题¶

你想在使用范围内执行某个代码片段，并且希望在执行后所有的结果都不可见。

解决方案¶

为了理解这个问题，先试试一个简单场景。首先，在全局命名空间内执行一个代码片段：

```
>>> a = 13
>>> exec('b = a + 1')
>>> print(b)
14
>>>
```

然后，再在一个函数中执行同样的代码：

```
>>> def test():
...     a = 13
...     exec('b = a + 1')
...     print(b)
...
>>> test()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in test
NameError: global name 'b' is not defined
>>>
```

可以看出，最后抛出了一个NameError异常，就跟在 `exec()` 语句从没执行过一样。 要是你想在后面的计算中使用到 `exec()` 执行结果的话就会有问题了。

为了修正这样的错误，你需要在调用 `exec()` 之前使用 `locals()` 函数来得到一个局部变量字典。 之后你就能从局部字典中获取修改过后的变量值了。例如：

```
>>> def test():
...     a = 13
...     loc = locals()
...     exec('b = a + 1')
...     b = loc['b']
...     print(b)
...
>>> test()
14
>>>
```

讨论¶

实际上对于 `exec()` 的正确使用是比较难的。大多数情况下当你要考虑使用 `exec()` 的时候， 还有另外更好的解决方案（比如装饰器、闭包、元类等等）。

然而，如果你仍然要使用 `exec()` ，本节列出了一些如何正确使用它的方法。 默认情况下， `exec()` 会在调用者局部和全局范围内执行代码。然而，在函数里面， 传递给 `exec()` 的局部范围是拷贝实际局部变量组成的一个字典。 因此，如果 `exec()` 如果执行了修改操作，这种修改后的结果对实际局部变量值是没有影响的。 下面是另外一个演示它的例子：

```
>>> def test1():
...     x = 0
...     exec('x += 1')
...     print(x)
...
>>> test1()
0
>>>
```

上面代码里，当你调用 `locals()` 获取局部变量时，你获得的是传递给 `exec()` 的局部变量的一个拷贝。 通过在代码执行后审查这个字典的值，那就能获取修改后的值了。下面是一个演示例子：

```
>>> def test2():
...     x = 0
...     loc = locals()
...     print('before:', loc)
...     exec('x += 1')
...     print('after:', loc)
...     print('x =', x)
...
>>> test2()
before: {'x': 0}
after: {'loc': {...}, 'x': 1}
x = 0
>>>
```

仔细观察最后一步的输出，除非你将 `loc` 中被修改后的值手动赋值给x，否则x变量值是不会变的。

在使用 `locals()` 的时候，你需要注意操作顺序。每次它被调用的时候， `locals()` 会获取局部变量值中的值并覆盖字典中相应的变量。 请注意观察下下面这个试验的输出结果：

```
>>> def test3():
...     x = 0
...     loc = locals()
...     print(loc)
...     exec('x += 1')
...     print(loc)
...     locals()
...     print(loc)
...
>>> test3()
{'x': 0}
{'loc': {...}, 'x': 1}
{'loc': {...}, 'x': 0}
>>>
```

注意最后一次调用 `locals()` 的时候x的值是如何被覆盖掉的。

作为 `locals()` 的一个替代方案，你可以使用你自己的字典，并将它传递给 `exec()` 。例如：

```
>>> def test4():
...     a = 13
...     loc = { 'a' : a }
...     glb = { }
...     exec('b = a + 1', glb, loc)
...     b = loc['b']
...     print(b)
...
>>> test4()
14
>>>
```

大部分情况下，这种方式是使用 `exec()` 的最佳实践。 你只需要保证全局和局部字典在后面代码访问时已经被初始化。

还有一点，在使用 `exec()` 之前，你可能需要问下自己是否有其他更好的替代方案。 大多数情况下当你要考虑使用 `exec()` 的时候， 还有另外更好的解决方案，比如装饰器、闭包、元类，或其他一些元编程特性。