8.5 在类中封装属性名¶

问题¶

你想封装类的实例上面的“私有”数据，但是Python语言并没有访问控制。

解决方案¶

Python程序员不去依赖语言特性去封装数据，而是通过遵循一定的属性和方法命名规约来达到这个效果。 第一个约定是任何以单下划线_开头的名字都应该是内部实现。比如：

```
class A:
    def __init__(self):
        self._internal = 0 # An internal attribute
        self.public = 1 # A public attribute

    def public_method(self):
        '''
        A public method
        '''
        pass

    def _internal_method(self):
        pass
```

Python并不会真的阻止别人访问内部名称。但是如果你这么做肯定是不好的，可能会导致脆弱的代码。 同时还要注意到，使用下划线开头的约定同样适用于模块名和模块级别函数。 例如，如果你看到某个模块名以单下划线开头(比如_socket)，那它就是内部实现。 类似的，模块级别函数比如 `sys._getframe()` 在使用的时候就得加倍小心了。

你还可能会遇到在类定义中使用两个下划线(__)开头的命名。比如：

```
class B:
    def __init__(self):
        self.__private = 0

    def __private_method(self):
        pass

    def public_method(self):
        pass
        self.__private_method()
```

使用双下划线开始会导致访问名称变成其他形式。 比如，在前面的类B中，私有属性会被分别重命名为 `_B__private` 和 `_B__private_method` 。 这时候你可能会问这样重命名的目的是什么，答案就是继承——这种属性通过继承是无法被覆盖的。比如：

```
class C(B):
    def __init__(self):
        super().__init__()
        self.__private = 1 # Does not override B.__private

    # Does not override B.__private_method()
    def __private_method(self):
```

```
    pass
```

这里，私有名称 `__private` 和 `__private_method` 被重命名为 `_C__private` 和 `_C__private_method` ，这个跟父类B中的名称是完全不同的。

讨论¶

上面提到有两种不同的编码约定(单下划线和双下划线)来命名私有属性，那么问题就来了：到底哪种方式好呢？ 大多数而言，你应该让你的非公共名称以单下划线开头。但是，如果你清楚你的代码会涉及到子类， 并且有些内部属性应该在子类中隐藏起来，那么才考虑使用双下划线方案。

还有一点要注意的是，有时候你定义的一个变量和某个保留关键字冲突，这时候可以使用单下划线作为后缀，例如：

```
lambda_ = 2.0 # Trailing _ to avoid clash with lambda keyword
```

这里我们并不使用单下划线前缀的原因是它避免误解它的使用初衷 (如使用单下划线前缀的目的是为了防止命名冲突而不是指明这个属性是私有的)。 通过使用单下划线后缀可以解决这个问题。