9.21 避免重复的属性方法¶

问题¶

你在类中需要重复的定义一些执行相同逻辑的属性方法，比如进行类型检查，怎样去简化这些重复代码呢？

解决方案¶

考虑下一个简单的类，它的属性由属性方法包装：

```
class Person:
    def __init__(self, name ,age):
        self.name = name
        self.age = age

    @property
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        if not isinstance(value, str):
            raise TypeError('name must be a string')
        self._name = value

    @property
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if not isinstance(value, int):
            raise TypeError('age must be an int')
        self._age = value
```

可以看到，为了实现属性值的类型检查我们写了很多的重复代码。 只要你以后看到类似这样的代码，你都应该想办法去简化它。 一个可行的方法是创建一个函数用来定义属性并返回它。例如：

```
def typed_property(name, expected_type):
    storage_name = '_' + name

    @property
    def prop(self):
        return getattr(self, storage_name)

    @prop.setter
    def prop(self, value):
        if not isinstance(value, expected_type):
            raise TypeError('{} must be a {}'.format(name, expected_type))
        setattr(self, storage_name, value)

    return prop

# Example use
class Person:
    name = typed_property('name', str)
    age = typed_property('age', int)

    def __init__(self, name, age):
        self.name = name
        self.age = age
```

讨论¶

本节我们演示内部函数或者闭包的一个重要特性，它们很像一个宏。例子中的函数 `typed_property()` 看上去有点难理

解，其实它所做的仅仅就是为你生成属性并返回这个属性对象。 因此，当在一个类中使用它的时候，效果跟将它里面的代码放到类定义中去是一样的。 尽管属性的 `getter` 和 `setter` 方法访问了本地变量如 `name`，`expected_type` 以及 `storate_name`，这个很正常，这些变量的值会保存在闭包当中。

我们还可以使用 `functools.partial()` 来稍稍改变下这个例子，很有趣。例如，你可以像下面这样：

```
from functools import partial

String = partial(typed_property, expected_type=str)
Integer = partial(typed_property, expected_type=int)

# Example:
class Person:
    name = String('name')
    age = Integer('age')

    def __init__(self, name, age):
        self.name = name
        self.age = age
```

其实你可以发现，这里的代码跟8.13小节中的类型系统描述器代码有些相似。