9.19 在定义的时候初始化类的成员¶

问题¶

你想在类被定义的时候就初始化一部分类的成员，而不是要等到实例被创建后。

解决方案¶

在类定义时就执行初始化或设置操作是元类的一个典型应用场景。本质上讲，一个元类会在定义时被触发， 这时候你可以执行一些额外的操作。

下面是一个例子，利用这个思路来创建类似于 `collections` 模块中的命名元组的类：

```
import operator

class StructTupleMeta(type):
    def __init__(cls, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for n, name in enumerate(cls._fields):
            setattr(cls, name, property(operator.itemgetter(n)))

class StructTuple(tuple, metaclass=StructTupleMeta):
    _fields = []
    def __new__(cls, *args):
        if len(args) != len(cls._fields):
            raise ValueError('{} arguments required'.format(len(cls._fields)))
        return super().__new__(cls,args)
```

这段代码可以用来定义简单的基于元组的数据结构，如下所示：

```
class Stock(StructTuple):
    _fields = ['name', 'shares', 'price']

class Point(StructTuple):
    _fields = ['x', 'y']
```

下面演示它如何工作：

```
>>> s = Stock('ACME', 50, 91.1)
>>> s
('ACME', 50, 91.1)
>>> s[0]
'ACME'
>>> s.name
'ACME'
>>> s.shares * s.price
4555.0
>>> s.shares = 23
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
>>>
```

讨论¶

这一小节中，类 `StructTupleMeta` 获取到类属性 `_fields` 中的属性名字列表， 然后将它们转换成相应的可访问特定元组槽的方法。函数 `operator.itemgetter()` 创建一个访问器函数， 然后 `property()` 函数将其转换成一个属性。

本节最难懂的部分是知道不同的初始化步骤是什么时候发生的。 `StructTupleMeta` 中的 `__init__()` 方法只在每个类被定义时被调用一次。 `cls` 参数就是那个被定义的类。实际上，上述代码使用了 `_fields` 类变量来保存新的被定义的类， 然后给它再添加一点新的东西。

`StructTuple` 类作为一个普通的基类，供其他使用者来继承。 这个类中的 `__new__()` 方法用来构造新的实例。 这里使

用 `__new__()` 并不是很常见，主要是因为我们要修改元组的调用签名， 使得我们可以像普通的实例调用那样创建实例。就像下面这样：

```
s = Stock('ACME', 50, 91.1) # OK
s = Stock(('ACME', 50, 91.1)) # Error
```

跟 `__init__()` 不同的是，`__new__()` 方法在实例被创建之前被触发。 由于元组是不可修改的，所以一旦它们被创建了就不可能对它做任何改变。而 `__init__()` 会在实例创建的最后被触发， 这样的话我们就可以做我们想做的了。这也是为什么 `__new__()` 方法已经被定义了。

尽管本节很短，还是需要你能仔细研读，深入思考Python类是如何被定义的，实例是如何被创建的， 还有就是元类和类的各个不同的方法究竟在什么时候被调用。

PEP 422 提供了一个解决本节问题的另外一种方法。 但是，截止到我写这本书的时候，它还没被采纳和接受。 尽管如此，如果你使用的是Python 3.3或更高的版本，那么还是值得去看一下的。