1.18 映射名称到序列元素¶

问题¶

你有一段通过下标访问列表或者元组中元素的代码，但是这样有时候会使得你的代码难以阅读， 于是你想通过名称来访问元素。

解决方案¶

`collections.namedtuple()` 函数通过使用一个普通的元组对象来帮你解决这个问题。 这个函数实际上是一个返回 Python 中标准元组类型子类的一个工厂方法。 你需要传递一个类型名和你需要的字段给它，然后它就会返回一个类，你可以初始化这个类，为你定义的字段传递值等。 代码示例：

```
>>> from collections import namedtuple
>>> Subscriber = namedtuple('Subscriber', ['addr', 'joined'])
>>> sub = Subscriber('jonesy@example.com', '2012-10-19')
>>> sub
Subscriber(addr='jonesy@example.com', joined='2012-10-19')
>>> sub.addr
'jonesy@example.com'
>>> sub.joined
'2012-10-19'
>>>
```

尽管 `namedtuple` 的实例看起来像一个普通的类实例，但是它跟元组类型是可交换的，支持所有的普通元组操作，比如索引和解压。 比如：

```
>>> len(sub)
2
>>> addr, joined = sub
>>> addr
'jonesy@example.com'
>>> joined
'2012-10-19'
>>>
```

命名元组的一个主要用途是将你的代码从下标操作中解脱出来。 因此，如果你从数据库调用中返回了一个很大的元组列表，通过下标去操作其中的元素， 当你在表中添加了新的列的时候你的代码可能就会出错了。但是如果你使用了命名元组，那么就不会有这样的顾虑。

为了说明清楚，下面是使用普通元组的代码：

```
def compute_cost(records):
    total = 0.0
    for rec in records:
        total += rec[1] * rec[2]
    return total
```

下标操作通常会让代码表意不清晰，并且非常依赖记录的结构。 下面是使用命名元组的版本：

```
from collections import namedtuple

Stock = namedtuple('Stock', ['name', 'shares', 'price'])
```

```
Stock = namedtuple('Stock', ['name', 'shares', 'price'])
def compute_cost(records):
    total = 0.0
    for rec in records:
        s = Stock(*rec)
        total += s.shares * s.price
    return total
```

讨论¶

命名元组另一个用途就是作为字典的替代，因为字典存储需要更多的内存空间。 如果你需要构建一个非常大的包含字典的数据结构，那么使用命名元组会更加高效。 但是需要注意的是，不像字典那样，一个命名元组是不可更改的。比如：

```
>>> s = Stock('ACME', 100, 123.45)
>>> s
Stock(name='ACME', shares=100, price=123.45)
>>> s.shares = 75
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
>>>
```

如果你真的需要改变属性的值，那么可以使用命名元组实例的 `_replace()` 方法， 它会创建一个全新的命名元组并将对应的字段用新的值取代。比如：

```
>>> s = s._replace(shares=75)
>>> s
Stock(name='ACME', shares=75, price=123.45)
>>>
```

`_replace()` 方法还有一个很有用的特性就是当你的命名元组拥有可选或者缺失字段时候， 它是一个非常方便的填充数据的方法。 你可以先创建一个包含缺省值的原型元组，然后使用 `_replace()` 方法创建新的值被更新过的实例。比如：

```
from collections import namedtuple

Stock = namedtuple('Stock', ['name', 'shares', 'price', 'date', 'time'])

# Create a prototype instance
stock_prototype = Stock('', 0, 0.0, None, None)

# Function to convert a dictionary to a Stock
def dict_to_stock(s):
    return stock_prototype._replace(**s)
```

下面是它的使用方法：

```
>>> a = {'name': 'ACME', 'shares': 100, 'price': 123.45}
>>> dict_to_stock(a)
Stock(name='ACME', shares=100, price=123.45, date=None, time=None)
>>> b = {'name': 'ACME', 'shares': 100, 'price': 123.45, 'date': '12/17/2012'}
>>> dict_to_stock(b)
Stock(name='ACME', shares=100, price=123.45, date='12/17/2012', time=None)
>>>
```

最后要说的是，如果你的目标是定义一个需要更新很多实例属性的高效数据结构，那么命名元组并不是你的最佳选择。这时候你应该考虑定义一个包含 `__slots__` 方法的类（参考8.4小节）。