**6.20 \*拷贝Python对象、浅拷贝和深拷贝**

2008-06-17 13:37 [美] Wesley J. Chun著/宋吉广译 人民邮电出版社 字号：

《Python核心编程》（中文第二版）第6章主要讲的是序列：字符串、列表和元组,接下来我们要研究这样一些Python的类型，它们的成员有序排列的，并且可以通过下标偏移量访问到它的一个或者几个成员，这类Python类型统称为序列，包括字符串（普通字符串和unicode字符串）、列表和元组类型,本节为大家介绍\*拷贝Python对象、浅拷贝和深拷贝 。

**6.20  \*拷贝Python对象、浅拷贝和深拷贝**

在前面的3.5节里面我们讲过，对象赋值实际上是简单的对象引用。也就是说，当你创建一个对象，然后把它赋给另一个变量的时候，Python并没有拷贝这个对象，而是拷贝了这个对象的引用。

比如，假设你想创建一对小夫妻的通用档案，名为person。然后你分别为他俩拷贝一份。在下面的例子中，我们展示了两种拷贝对象的方式，一种使用了切片操作，另一种用了工厂方法，为了区分出3个不同的对象，我们使用id()内建函数来显示每个对象的标识符。（我们还可以用is操作符来做相同的事情）。

|  |
| --- |
| >>> person = ['name', ['savings', 100.00]] >>> hubby = person[:]  # slice copy >>> wifey = list(person) # fac func copy >>> [id(x) for x in person, hubby, wifey] [11826320, 12223552, 11850936] |

为他们创建了初始有$100的个人存款账户，用户名改为定制的名字。但是，当丈夫取走$50后，他的行为影响到了他妻子的账户，虽然我们进行了分开的拷贝操作（当然，前提是我们希望他们每个人都拥有自己单独的账号，而不是一个单一的联合账号。）为什么会这样呢？

|  |
| --- |
| >>> hubby[0] = 'joe' >>> wifey[0] = 'jane' >>> hubby, wifey (['joe', ['savings', 100.0]], ['jane', ['savings', 100.0]]) >>> hubby[1][1] = 50.00 >>> hubby, wifey (['joe', ['savings', 50.0]], ['jane', ['savings', 50.0]]) |

原因是我们仅仅做了一个浅拷贝。对一个对象进行浅拷贝其实是新创建了一个类型跟原对象一样，其内容是原来对象元素的引用，换句话说，这个拷贝的对象本身是新的，但是它的内容不是。

序列类型对象的浅拷贝是默认类型拷贝，并可以以下几种方式实施  
 (1)完全切片操作[:]；  
 (2)利用工厂函数，比如list()、dict()等；  
 (3)使用copy模块的copy函数。

你的下一个问题可能是，当妻子的名字被赋值，为什么丈夫的名字没有受到影响？难道它们的名字现在不应该都是'jane'了吗？为什么名字没有变成一样的呢？怎么会是这样呢？这是因为在这两个列表的两个对象中。第1个对象是不可变的（是个字符串类型），而第2个是可变的（一个列表）。正因为如此，当进行浅拷贝时，字符串被显式的拷贝，并新创建了一个字符串对象，而列表元素只是把它的引用复制了一下，并不是它的成员。所以改变名字没有任何问题，但是更改他们银行账号的任何信息都会引发问题。现在，让我们分别看一下每个列表的元素的对象ID值。注意，银行账号对象是同一个对象，这也是为什么对一个对象进行修改会影响到另一个的原因。注意在我们改变他们的名字后，新的名字字符串是如何替换原有‘名字’字符串的。

改变前

|  |
| --- |
| >>> [id(x) for x in hubby] [9919616, 11826320] >>> [id(x) for x in wifey] [9919616, 11826320] |

改变后

|  |
| --- |
| >>> [id(x) for x in hubby] [12092832, 11826320] >>> [id(x) for x in wifey] [12191712, 11826320] |

假设我们要给这对夫妻创建一个联合账户，那这是一个非常棒的方案。但是，如果需要的是两个分离账户，就需要作些改动了。要得到一个完全拷贝或者说深拷贝——创建一个新的容器对象，包含原有对象元素（引用）全新拷贝的引用——需要copy.deepcopy()函数。我们使用深拷贝来重写整个例子。

|  |
| --- |
| >>> person = ['name', ['savings', 100.00]] >>> hubby = person >>> import copy >>> wifey = copy.deepcopy(person) >>> [id(x) for x in person, hubby, wifey] [12242056, 12242056, 12224232] >>> hubby[0] = 'joe' >>> wifey[0] = 'jane' >>> hubby, wifey (['joe', ['savings', 100.0]], ['jane', ['savings', 100.0]]) >>> hubby[1][1] = 50.00 >>> hubby, wifey (['joe', ['savings', 50.0]], ['jane', ['savings', 100.0]]) |

这就是我们想要的方式。作为验证，让我们确认一下所有四个对象都是不同的。

|  |
| --- |
| >>> [id(x) for x in hubby] [12191712, 11826280] >>> [id(x) for x in wifey] [12114080, 12224792] |

以下有几点关于拷贝操作的警告。第一，非容器类型（比如数字、字符串和其他“原子”类型的对象，像代码、类型和xrange对象等）没有被拷贝一说，浅拷贝是用完全切片操作来完成的。第二，如果元组变量只包含原子类型对象，对它的深拷贝将不会进行。如果我们把账户信息改成元组类型，那么即便按我们的要求使用深拷贝操作也只能得到一个浅拷贝。

|  |
| --- |
| >>> person = ['name', ('savings', 100.00)] >>> newPerson = copy.deepcopy(person) >>> [id(x) for x in person, newPerson] [12225352, 12226112] >>> [id(x) for x in person] [9919616, 11800088] >>> [id(x) for x in newPerson] [9919616, 11800088] |

核心模块：copy  
  
我们刚才描述的浅拷贝和深拷贝操作都可以在copy模块中找到。

其实copy模块中只有两个函数可用：copy()进行浅拷贝操作，而deepcopy()进行深拷贝操作。