第118天：Python 之对象的比较与拷贝

原创 豆豆 Python技术 2月3日

众所周知，Python 是一门面向对象语言，在 Python 的世界一切皆对象，那么我们如何判断两个对象是否是同一个对象呢。

== 操作符和 is

相信大家对于这两个操作符都不陌生。具体来说就是 `==` 操作符比较的是两个对象的值是否相等，而 is 操作符的含义则是二者到底是否是同一个对象，换言之，即两个对象是否指向同一块内存地址。

上面我们说过，Python 中一切皆是对象，对象包含 id（唯一身份标识），type（类型） 和 value（值） 三个要素。id 可以通过函数 `id(obj)` 来获取。因此 is 操作符就相当于比较两个对象的 id 是否相同，而 `==` 操作符则相当于比较两个对象的 value 是否相同。

下面我们来看几个例子。

```
>>> a = 'red'
>>> b = 'red'
>>> a == b
True
```

这里，我们声明了两个对象 a 和 b，其内容都是字符串 ‘red’，毋庸置疑 `==` 操作符应该返回 True，很好理解。

```
>>> a = 256
>>> b = 256
>>> a == b
True
>>> a is b
True

>>> a = 257
>>> b = 257
>>> a == b
True
>>> a is b
False
```

同样，对于 `==` 操作符，无论是 a，b 的值是 256 还是 257 二者都是相等的。奇怪的是同样是 is 操作，数值的大小居然对结果有影响。

事实上 a is b 为 True 的结论只适用于 -5 到 256 的数值，因为出于性能的考虑，Python 对这个范围内的数值进行了缓存。当你为整数对象赋值时（-5 到 256）并不会生成新的对象，而是使用事先创建好的缓存对象。如果超过了缓存范围，那么就会申请两块不同的内存地址，is 操作当然会返回 False。

不信我们可以把它们的 id 拿出来看看。

```
>>> a = 256
>>> b = 256
>>> id(a)
4525792016
>>> id(b)
4525792016

>>> a = 257
>>> b = 257
>>> id(a)
4528947760
>>> id(b)
4528947856
```

但是，当你把同样的代码放到编辑器中去去执行时，你会惊奇的发现程序的执行结果跟我们刚才所说的缓存机制竟然是相冲突的。

不是说 -5 到 256 范围内的整数才会被缓存么。为啥这么大的数 is 操作也返回 True 了呢。

从结果来看 a 和 b 两个大数的内存地址肯定是一样的，不然 is 操作符也不会返回 True。这是因为在交互模式（就是那个黑窗口了）下每一条命令就是一个代码块，Python 逐行编译执行；在编辑器中，一个函数，一个类或者一个文件才是一个代码块。Python 会整体编译执行，因此相同值的变量只会初始化一次，第二次初始化相同值的变量时会重用旧值。

> 上面所说的编译是指 CPython 将源代码编译为字节码的过程。

只有当对象的值是数值或者字符串型且在缓存范围内时，a is b 才返回 True，否则当 a 和 b 是 int，str，list，tuple，set 或 dict 类型时，a is b 均返回 False。

事实上，经过测试，我发现对于有空格的字符串 Python 并不会去缓存。找了好久，终于在 stringobject.h 中找到了解释，Python 解释器采取 intern 机制来提高字符串操作效率，当内存中有相同值的字符串时就会重用，而不是生成一个新的相同值的字符串对象。但也不是说对所有的字符串均采取该 intern 机制。只有看起来像 Python 标识符的字符串才会被缓存。

> This is generally restricted to strings that "look like" Python identifiers, although the intern() builtin can be used to force interning of any string.

另外，比较操作符 is 的效率要优于 `==`，因为 is 操作符无法被重载，执行 is 操作只是对比对象的 id 而已。而 `==` 操作

符则会递归地遍历对象的所有值，并逐一比较。

题外话：如果你了解 Java 就会发现，Python 中的 `==` 类似于 Java 中的 equals，而 is 则类似与 Java 中的 `==` 比较符。

对象的拷贝

对象的拷贝其实就是创建新的对象的过程，在 Python 中共有两种拷贝模式，浅拷贝和深拷贝。

当顶层对象和它的子元素对象全都是不可变对象时，不存在被拷贝，因为没有产生新对象。浅拷贝和深拷贝的区别就是浅拷贝只拷贝顶层对象，而不会去拷贝内部的子元素对象。深拷贝则会递归地拷贝顶层对象内部的子元素对象。

我们可以使用对象类型本身的构造器，切片 以及 copy 函数来实现浅拷贝。

```
a = [1, 'hello', [1,2]]
b = list(a)

a[0] = 100
a[2][0] = 100
print(a)
print(b)

## 输出结果
[100, 'hello', [100, 2]]
[1, 'hello', [100, 2]]
```

对于顶层可变的对象，如果其子对象不可变，那当你修改子对象时，实际上是将引用指向了另外一个新的对象而已。类比上边的例子就是并不是把 `a[0]` 的值从 1 修改为 100，而是将 `a[0]` 指向 100。

如果子对象可变，比如对于 `a[2]` 来说，由于是浅拷贝，所以实际上 `a[2]` 和 `b[2]` 指向的都是同一个列表对象。修改 `a[2][0]` 为 100之后， `b[2][0]` 也一并会修改。

通过切片来实现浅拷贝。

```
>>> a = [1, 2, 3]
>>> b = a[:]
>>> a == b
True
>>> a is b
False
```

但是对于顶层不可变的对象，不存在对象的拷贝，因为都是指向同一个对象，没有新的对象产生。

```
>>> a = (1,2,3)
```

```
>>> b = tuple(a)
>>> a == b
True
>>> a is b
True
```

如你所见，关于浅拷贝如果元素不可变的还好，没什么副作用；如果元素可变，那就要小心其副作用了。

深拷贝递归拷贝顶层对象以及它内部的子对象，因此，新对象和原来的旧对象，没有任何关联。Python 中使用 copy.deepcopy() 函数来实现对对象的深拷贝。

```
import copy

a = [1, 'hello', [1,2]]
b = copy.deepcopy(a)

a[0] = 100
a[2][0] = 100
print(a)
print(b)

## 输出结果
[100, 'hello', [100, 2]]
[1, 'hello', [1, 2]]
```

深拷贝即拷贝了顶层对象，同时还拷贝了子对象，所以 `a[2]` 和 `b[2]` 指向的是两个不同的列表。修改 `a[2][0]` 后，重新指向了新的整数，但是这并不会影响到 `b[2]` 。

```
>>> import copy
>>> a = (1,2,3)
>>> b = copy.deepcopy(a)
>>> a is b
True

>>> a = (1,2,[1,2])
>>> b = copy.deepcopy(a)
>>> a is b
False
>>> a[2][0] = 100
>>> a
(1, 2, [100, 2])
>>> b
(1, 2, [1, 2])
```

对于不可变对象来说，如果其子对象全部不可变，那么深拷贝就和浅拷贝是一样的效果，都是指向同一个内存地址。

但是如果子对象中包含可变对象，那么深拷贝之后的对象就不再是原来的对象了，因为可变对象被重新拷贝了一份，放到例子中就是 `a[2]` 和 `b[2]` 指向的不再是同一个列表。因此，修改 `a[2]` 并不会影响 `b[2]` 。

如果深拷贝的对象中存在指向自身的引用，那么会不会无限递循环呢。

答案是不会，深拷贝函数内部维护了一个字典，该字典记录了已经拷贝的对象与其 id。拷贝过程中，如果字典里已经存储了将要拷贝的对象，则会从字典直接返回，不再进行递归。

总结

这篇文章介绍了对象的比较和拷贝。is 比较两个对象的 id 值是否相等，`==` 比较的是两个对象的值是否相等，小整数对象 [-5, 256] 会被缓存起来重复使用，is 的效率要优于 `==` 操作。浅拷贝是对原对象中子对象的引用，有可能会产生副作用，深拷贝会创建新的对象，不会和原对象干扰。相信你学完之后会对 Python 中对象的理解会更深一些。

代码地址

> 示例代码：https://github.com/JustDoPython/python-100-day/tree/master/day-118

系列文章

第117天：机器学习算法之 K 近邻
第116天：机器学习算法之朴素贝叶斯理论
第115天：Python 到底是值传递还是引用传递
第 114 天：三木板模型算法项目实战
第 113 天：Python XGBoost 算法项目实战
第 112 天：机器学习算法之蒙特卡洛
第 111 天：Python 垃圾回收机制
从 0 学习 Python 0 - 110 大合集总结

PS： 公号内回复 ：Python，即可进入Python 新手学习交流群，一起**100天计划！**

-END-

Python 技术

关于 Python 都在这里