图解Python深拷贝和浅拷贝

Python中，对象的赋值，拷贝（深/浅拷贝）之间是有差异的，如果使用的时候不注意，就可能产生意外的结果。

下面本文就通过简单的例子介绍一下这些概念之间的差别。

对象赋值

直接看一段代码：

```
will = ["Will", 28, ["Python", "C#", "JavaScript"]]
wilber = will
print id(will)
print will
print [id(ele) for ele in will]
print id(wilber)
print wilber
print [id(ele) for ele in wilber]

will[0] = "Wilber"
will[2].append("CSS")
print id(will)
print will
print [id(ele) for ele in will]
print id(wilber)
print wilber
print [id(ele) for ele in wilber]
```

代码的输出为：

```
39737304
['Will', 28, ['Python', 'C#', 'JavaScript']]
[39413120, 36218340, 39766256]
39737304
['Will', 28, ['Python', 'C#', 'JavaScript']]
[39413120, 36218340, 39766256]
39737304
['Wilber', 28, ['Python', 'C#', 'JavaScript', 'CSS']]
[39758496, 36218340, 39766256]
39737304
['Wilber', 28, ['Python', 'C#', 'JavaScript', 'CSS']]
[39758496, 36218340, 39766256]
```

下面来分析一下这段代码：

- 首先，创建了一个名为will的变量，这个变量指向一个list对象，从第一张图中可以看到所有对象的地址（每次运行，结果可能不同）
- 然后，通过will变量对wilber变量进行赋值，那么wilber变量将指向will变量对应的对象（内存地址），也就是说"wilber is will"，"wilber[i] is will[i]"
  - 可以理解为，Python中，对象的赋值是
    存地址）传递
- 第三张图中，由于will和wilber指向同一个
  何修改都会体现在wilber上
  - 这里需要注意的一点是，str是不可变
    候会替换旧的对象，产生一个新的地址3975

2. Re:彻底理解JavaScript原型

看完了，感谢楼主。有个小疑问，遍历对象的属性时，为什么不会打印出_proto_这个属性。

--大兄弟竹子

3. Re:彻底理解JavaScript原型

zhege henguanjian！

--大兄弟竹子

4. Re:关于JavaScript继承的那些事

经过重写原型之后情况更加复杂了，下面就来看看重写原型之后的对象关系图：(有误)

--习习哈哈

5. Re:彻底理解JavaScript原型

非常好,学习了

--乘以乘

阅读排行榜

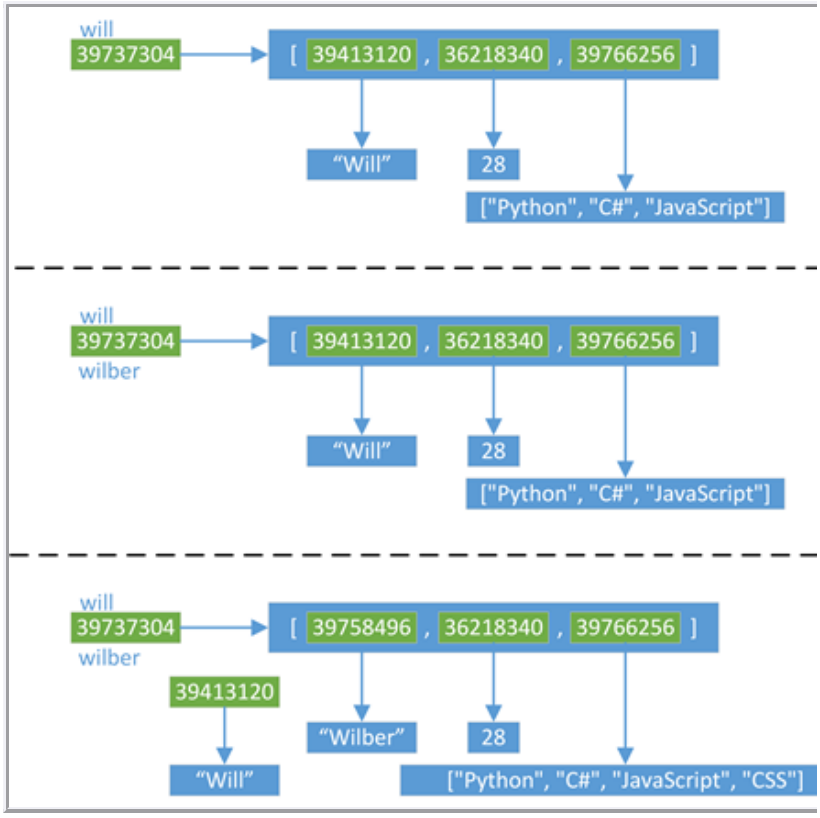
1. Python格式化字符串(16592)
2. 彻底理解JavaScript原型(8873)
3. 图解Python深拷贝和浅拷贝(4141)
4. Python之包管理工具(3901)
5. 动手学习TCP：数据传输(3622)

评论排行榜

1. 彻底理解JavaScript原型(32)
2. Git Step by Step – (1) Git 简介(21)
3. 给博客添加一个的目录(9)
4. JavaScript中的this(9)
5. 动手学习TCP: 环境搭建(9)

推荐排行榜

1. 彻底理解JavaScript原型(207)
2. JavaScript的执行上下文(38)
3. 动手学习TCP：数据传输(27)
4. JavaScript中的this(22)
5. 理解JavaScript的作用域链(21)



浅拷贝

下面就来看看浅拷贝的结果：

```
import copy

will = ["Will", 28, ["Python", "C#", "JavaScript"]]
wilber = copy.copy(will)

print id(will)
print will
print [id(ele) for ele in will]
print id(wilber)
print wilber
print [id(ele) for ele in wilber]

will[0] = "Wilber"
will[2].append("CSS")
print id(will)
print will
print [id(ele) for ele in will]
print id(wilber)
print wilber
print [id(ele) for ele in wilber]
```

代码结果为：

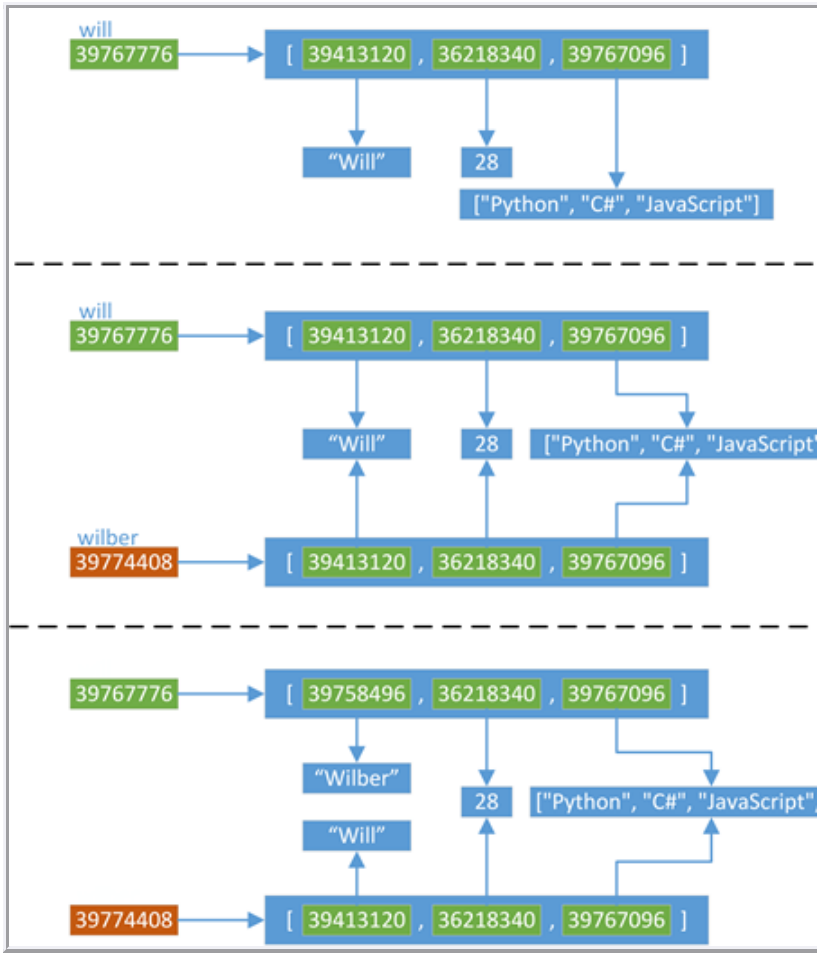
```
39767776
['Will', 28, ['Python', 'C#', 'JavaScri
[39413120, 36218340, 39767096]
39774408
['Will', 28, ['Python', 'C#', 'JavaScri
[39413120, 36218340, 39767096]
39767776
['Wilber', 28, ['Python', 'C#', 'JavaSc
[39758496, 36218340, 39767096]
39774408
['Will', 28, ['Python', 'C#', 'JavaScript',
[39413120, 36218340, 39767096]
```


5

Fork me on GitHub

分析一下这段代码：

- 首先，依然使用一个will变量，指向一个list类型的对象
- 然后，通过copy模块里面的浅拷贝函数copy()，对will指向的对象进行浅拷贝，然后浅拷贝生成的新对象赋值给wilber变量
  - 浅拷贝会创建一个新的对象，这个例子中**"wilber is not will"**
  - 但是，对于对象中的元素，浅拷贝就只会使用原始元素的引用（内存地址），**也就是说"wilber[i] is will[i]"**
- 当对will进行修改的时候
  - 由于list的第一个元素是不可变类型，所以will对应的list的第一个元素会使用一个新的对象39758496
  - 但是list的第三个元素是一个可变类型，修改操作不会产生新的对象，所以will的修改结果会相应的反应到wilber上



总结一下，当我们使用下面的操作的时候，会产生浅拷贝的效果：

- 使用切片[:]操作
- 使用工厂函数（如list/dir/set）
- 使用copy模块中的copy()函数

深拷贝

最后来看看深拷贝：

```
import copy

will = ["Will", 28, ["Python", "C#", "JavaScript"]]
wilber = copy.deepcopy(will)

print id(will)
print will
```

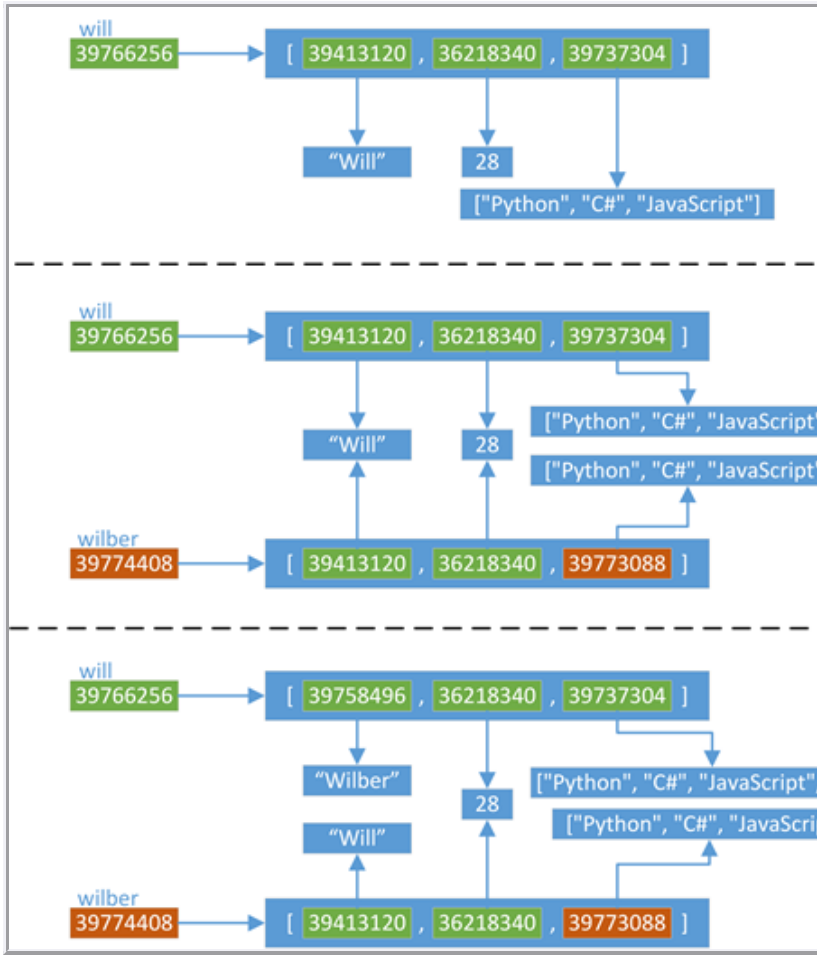
```
print [id(ele) for ele in will]
print id(wilber)
print wilber
print [id(ele) for ele in wilber]

will[0] = "Wilber"
will[2].append("CSS")
print id(will)
print will
print [id(ele) for ele in will]
print id(wilber)
print wilber
print [id(ele) for ele in wilber]
```

代码的结果为：

```
39766256
['Will', 28, ['Python', 'C#', 'JavaScript']]
[39413120, 36218340, 39737304]
39767776
['Will', 28, ['Python', 'C#', 'JavaScript']]
[39413120, 36218340, 39773088]
39766256
['Wilber', 28, ['Python', 'C#', 'JavaScript', 'CSS']]
[39758496, 36218340, 39737304]
39767776
['Will', 28, ['Python', 'C#', 'JavaScript']]
[39413120, 36218340, 39773088]
```

分析一下这段代码：

- 首先，同样使用一个will变量，指向一个list类型的对象
- 然后，通过copy模块里面的深拷贝函数deepcopy()，对will指向的对象进行深拷贝，然后深拷贝生成的新对象赋值给wilber变量
  - 跟浅拷贝类似，深拷贝也会创建一个新的对象，这个例子中**"wilber is not will"**
  - 但是，对于对象中的元素，深拷贝都会重新生成一份（有特殊情况，下面会说明），而不是简单的使用原始元素的引用（内存地址）
    - 例子中will的第三个元素指向39737304，而wilber的第三个元素是一个全新的对象39773088，也就是说，**"wilber[2] is not will[2]"**
- 当对will进行修改的时候
  - 由于list的第一个元素是不可变类型，所以will对应的list的第一个元素会使用一个新的对象39758496
  - 但是list的第三个元素是一个可不类型，修改操作不会产生新的对象，但是由于"wilber[2] is not will[2]"，所以will的修改不会影响wilber



拷贝的特殊情况

其实，对于拷贝有一些特殊情况：

- 对于非容器类型（如数字、字符串、和其他'原子'类型的对象）没有拷贝这一说
  - 也就是说，对于这些类型，"obj is copy.copy(obj)" 、"obj is copy.deepcopy(obj)"
- 如果元祖变量只包含原子类型对象，则不能深拷贝，看下面的例子

```
1  import copy
2
3  books = ("Python", "C#", "JavaScript")
4  copies = copy.deepcopy(books)
5  print books is copies
6
7  books = ("Python", "C#", "JavaScript", [])
8  copies = copy.deepcopy(books)
9  print books is copies
```

```
PAUSE
True
False
Press any key to contin
```

总结

本文介绍了对象的赋值和拷贝，以及它们之间

- Python中对象的赋值都是进行对象引用（
- 使用copy.copy()，可以进行对象的浅拷贝于对象中的元素，依然使用原始的引用.
- 如果需要复制一个容器对象，以及它里面的所有的子元素），可以使用copy.deepcopy()进行深拷

5

- 对于非容器类型（如数字、字符串、和其他'原子'类型的对象）没有被拷贝一说
- 如果元祖变量只包含原子类型对象，则不能深拷贝，看下面的例子

分类: 我的Python笔记

« 上一篇：Python格式化字符串
» 下一篇：Python迭代器和生成器

posted @ 2017-05-05 10:21 田小计划 阅读(4140) 评论(...) 编辑 收藏