

02-深拷贝、浅拷贝

1. 浅拷贝

浅拷贝是对于一个对象的顶层拷贝。

通俗的了解就是：拷贝了引用并没有拷贝内容。

如：`a = 1` 这条赋值语句，数值是不会直接存放在变量中的，其中 `1` 的值存放在内存单元中，`a` 这个变量只是存放了 `1` 这个值内存单元的地址（引用）。在Python中所有的赋值语句基本上是引用。

变量可以看做是一个内存空间的名称（可能）。

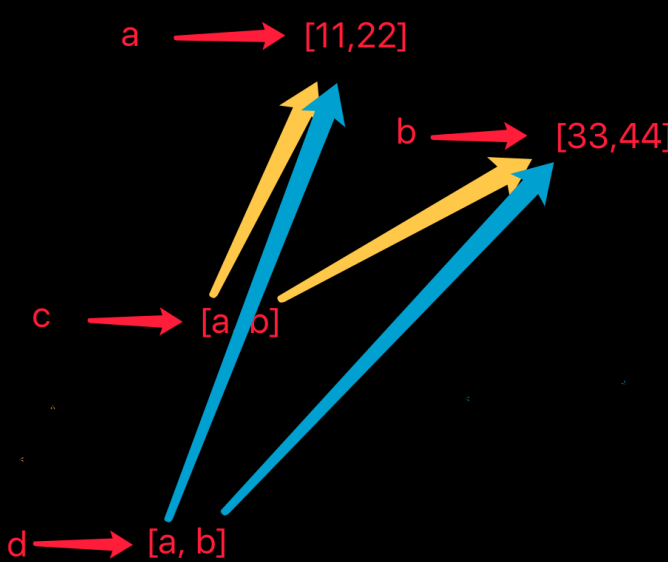
```
In [1]: a = [11, 22]
In [2]: b = a
In [3]: id(a) # 用来显示a指向的数据的内存地址
Out[3]: 4399944520
In [4]: id(b) # 用来显示b指向的数据的内存地址
Out[4]: 4399944520
In [5]: # 以上结果相同，说明了当给一个变量赋值时，其实就是将数据的引用复制了一份给了另外一个变量，这其实就是最简单的浅拷贝
In [6]: # 不仅列表是这样只要是 类似于 xx1 = xx2的这种基本都是 浅拷贝
In [7]:
In [7]: c = {"name": "laowang"}
In [8]: d = c
In [9]: id(c)
Out[9]: 4396398704
In [10]: id(d)
Out[10]: 4396398704
In [11]:
In [11]: c["passwd"] = "123456"
In [12]: c
Out[12]: {'name': 'laowang', 'passwd': '123456'}
In [13]: d
Out[13]: {'name': 'laowang', 'passwd': '123456'}
In [14]: # 因为都是浅拷贝，所以只要通过一个引用进行了修改，那么另外一个变量就看到的也就变化了
```



```
graph LR
    a --> list["[11, 22]"]
    b --> list
    c --> dict["{'name': 'laowang'}"]
    d --> dict
```

当一个变量赋值时，其实就是将数据的引用复制了一份给了另一个变量，这是最简单的浅拷贝。

```
In [9]: a = [11,22]
In [10]: b = [33,44]
In [11]: c = [a,b]
In [12]: id(a)
Out[12]: 4395286984
In [13]: id(b)
Out[13]: 4398005384
In [14]: id(c)
Out[14]: 4395392520
In [15]: import copy
In [16]: d = copy.copy(c)
In [17]: id(d)
Out[17]: 4395293000
In [18]: id(d[0])
Out[18]: 4395286984
In [19]: id(d[1])
Out[19]: 4398005384
In [20]: a.append(11)
In [21]: c
Out[21]: [[11, 22, 11], [33, 44]]
In [22]: d
Out[22]: [[11, 22, 11], [33, 44]]
```



浅copy, 只会复制最顶层的那个列表

如上图所示，浅拷贝只是拷贝了存储于内存中的变量值的地址，或者说引用，所以使用 `id()` 函数查看变量中值的引用地址时查到的地址值与原变量是一样的。

其中 `copy()` 方法是浅拷贝，仅仅复制最顶层的那个列表。

```
1 a = [11,22]
2 b = [33,44]
3 c = [a,b]
4 d = c
5 e = copy.copy(c)
6 # 其中 copy() 方法是浅拷贝，仅仅复制最顶层的那个列表，即 copy() 再新建一个列表存放列表
  c 中的变量 a,b ，而变量 a,b 存放的是数据的引用。
7 # 因此也可以预见以下代码的结果
8 x = [[a,b]]
9 y = copy.copy(x)
10 id(x[0]) == id(y[0]) # True
```

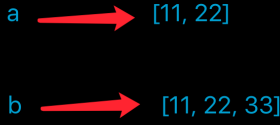
2. 深拷贝

深拷贝是对于一个对象所有层次的拷贝（递归）。

```

In [16]: import copy
In [17]: a = [11, 22]
In [18]: b = copy.deepcopy(a) # 对a指向的列表进行深copy
In [19]: a
Out[19]: [11, 22]
In [20]: b
Out[20]: [11, 22]
In [21]: id(a)
Out[21]: 4399904456
In [22]: id(b)
Out[22]: 4400396296
In [23]: # 以上结果说明了通过deepcopy确实将a列表中所有的数据的引用copy了，而不是只copy了a指向的列表的引用
In [24]:
In [24]: a.append(33)
In [25]: a
Out[25]: [11, 22, 33]
In [26]: b
Out[26]: [11, 22]
In [27]:

```

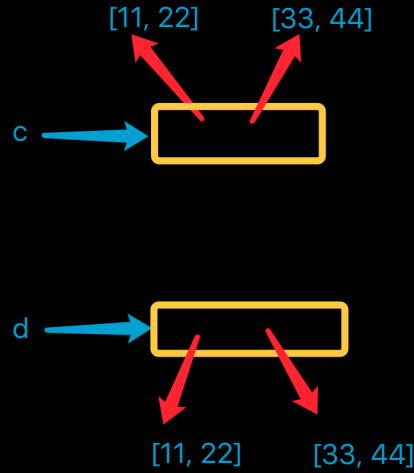


进一步了解深拷贝。

```

In [32]: import copy
In [33]: a = [11, 22]
In [34]: b = [33, 44]
In [35]: c = [a, b]
In [36]: d = copy.deepcopy(c)
In [37]: id(a)
Out[37]: 4400357768
In [38]: id(b)
Out[38]: 4399676872
In [39]: id(c)
Out[39]: 4400148872
In [40]: id(c[0])
Out[40]: 4400357768
In [41]: id(c[1])
Out[41]: 4399676872

```



```

In [42]:
In [42]: id(d[0])
Out[42]: 4400003144

In [43]: id(d[1])
Out[43]: 4400139016

In [44]: id(d)
Out[44]: 4400089544

In [45]:
In [45]: c[0].append(55)

In [46]: c
Out[46]: [[11, 22, 55], [33, 44]]

In [47]: d
Out[47]: [[11, 22], [33, 44]]

```

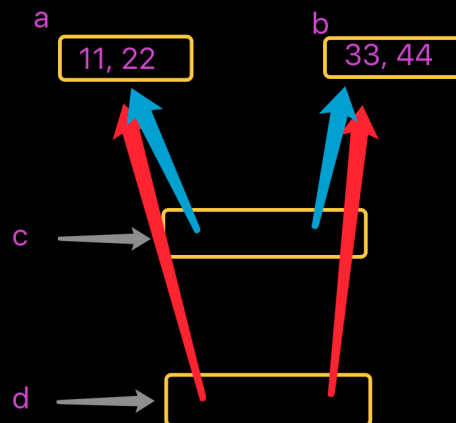
3. 拷贝的其他方式

- 分片表达式可以赋值一个序列

```

In [13]: a = [11, 22]
In [14]: b = [33, 44]
In [15]: c = [a, b]
In [16]: d = c[:]
In [17]: id(c)
Out[17]: 4500666120
In [18]: id(d)
Out[18]: 4499927112
In [19]: id(c[0])
Out[19]: 4499881672
In [20]: id(d[0])
Out[20]: 4499881672
In [21]: a
Out[21]: [11, 22]
In [22]: a.append(33)
In [23]: c
Out[23]: [[11, 22, 33], [33, 44]]
In [24]: d
Out[24]: [[11, 22, 33], [33, 44]]
In [25]:

```



$d = c[:]$ 与 $d = \text{copy.copy}(c)$ 一样 属于浅copy

- 字典的 copy 方法可以拷贝一个字典

```

In [62]: d = dict(name="zhangsan", age=27)
In [63]: co = d.copy()
In [64]: id(d)
Out[64]: 4397681184
In [65]: id(co)
Out[65]: 4378467208
In [66]: d
Out[66]: {'age': 27, 'name': 'zhangsan'}
In [67]: co
Out[67]: {'age': 27, 'name': 'zhangsan'}
In [68]:
In [68]: d = dict(name="zhangsan", age=27, children_ages = [11, 22])
In [69]: co = d.copy()
In [70]: d["children_ages"].append(9)
In [71]: d
Out[71]: {'age': 27, 'children_ages': [11, 22, 9], 'name': 'zhangsan'}
In [72]: co
Out[72]: {'age': 27, 'children_ages': [11, 22, 9], 'name': 'zhangsan'}
In [73]:

```

4. 注意点

浅拷贝对不可变类型和可变类型的copy不同

1. copy.copy对于可变类型，会进行浅拷贝
2. copy.copy对于不可变类型，不会拷贝，仅仅是指向

```

1 In [88]: a = [11,22,33]
2 In [89]: b = copy.copy(a)
3 In [90]: id(a)
4 Out[90]: 59275144
5 In [91]: id(b)
6 Out[91]: 59525600
7 In [92]: a.append(44)
8 In [93]: a
9 Out[93]: [11, 22, 33, 44]
10 In [94]: b
11 Out[94]: [11, 22, 33]
12
13
14 In [95]: a = (11,22,33)
15 In [96]: b = copy.copy(a)
16 In [97]: id(a)
17 Out[97]: 58890680
18 In [98]: id(b)
19 Out[98]: 58890680

```

```
In [24]: a = [11,22]
In [25]: b = [33,44]
In [26]: c = (a,b)
In [27]: d = copy.copy(c)
In [28]: id(c)
Out[28]: 4398226568
In [29]: id(d)
Out[29]: 4398226568
In [30]: a.append(33)
In [31]: c
Out[31]: ([11, 22, 33], [33, 44])
In [32]: d
Out[32]: ([11, 22, 33], [33, 44])
In [33]:
In [33]: e = copy.deepcopy(c)
In [34]: id(c)
Out[34]: 4398226568
In [35]: id(e)
Out[35]: 4399065544
In [36]: a.append(44)
In [37]: c
Out[37]: ([11, 22, 33, 44], [33, 44])
In [38]: e
Out[38]: ([11, 22, 33], [33, 44])
In [39]:
```

如果c是元组，那么 copy时会，仅仅是元组的引用copy
而deepcopy依然是深copy，即递归copy所有

copy.copy和copy.deepcopy的区别

copy.copy

```
In [81]: a = [11, 22]
```

```
In [82]: b = (a, )
```

```
In [83]: c = [b,]
```

```
In [84]:
```

```
In [84]: d = copy.copy(c)
```

```
In [85]:
```

```
In [85]: c
```

```
Out[85]: [[11, 22],,]
```

```
In [86]: d
```

```
Out[86]: [[11, 22],,]
```

```
In [87]: a.append(33)
```

```
In [88]: c
```

```
Out[88]: [[11, 22, 33],,]
```

```
In [89]: d
```

```
Out[89]: [[11, 22, 33],,]
```

```
In [90]: id(c)
```

```
Out[90]: 4400371720
```

```
In [91]: id(d)
```

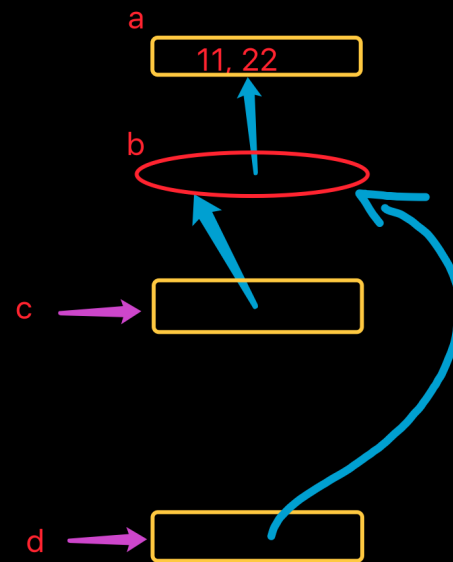
```
Out[91]: 4400343688
```

```
In [92]: id(c[0])
```

```
Out[92]: 4400563256
```

```
In [93]: id(d[0])
```

```
Out[93]: 4400563256
```



`d = c` # 让d这个变量指向c指向的空间
`d = copy.copy(c)` # 复制所有c指向的数据到一个新空间，但是不会递归copy

```
In [98]: a = [11, 22]
```

```
In [99]: b = [a]
```

```
In [100]: c = [b]
```

```
In [101]:
```

```
In [101]: d = copy.copy(c)
```

```
In [102]:
```

```
In [102]: c
```

```
Out[102]: [[11, 22]]
```

```
In [103]: d
```

```
Out[103]: [[11, 22]]
```

```
In [104]: id(c)
```

```
Out[104]: 4395608584
```

```
In [105]: id(d)
```

```
Out[105]: 4400474632
```

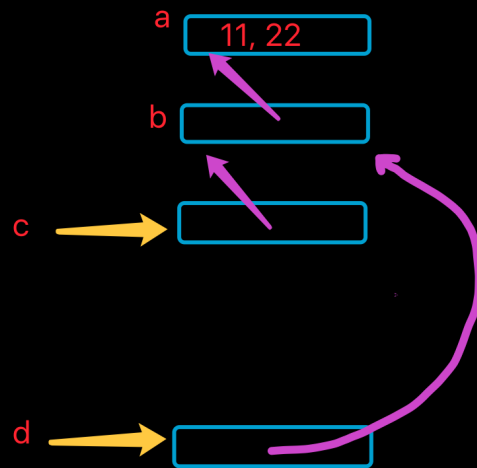
```
In [106]: id(c[0])
```

```
Out[106]: 4400474824
```

```
In [107]: id(d[0])
```

```
Out[107]: 4400474824
```

```
In [108]:
```



copy.deepcopy


```
In [121]: a = [11, 22]

In [122]: b = [a]

In [123]: c = [b]

In [124]:

In [124]: d = copy.deepcopy(c)

In [125]:

In [125]: c
Out[125]: [[[11, 22]]]

In [126]: d
Out[126]: [[[11, 22]]]

In [127]: id(c)
Out[127]: 4399971144

In [128]: id(d)
Out[128]: 4399853064

In [129]: id(c[0])
Out[129]: 4400242312

In [130]: id(d[0])
Out[130]: 4400473736
```

```
In [132]: a
Out[132]: [11, 22]

In [133]: a.append(33)
Out[133]: [11, 22, 33]

In [134]: a
Out[134]: [11, 22, 33]

In [135]: c
Out[135]: [[[11, 22, 33]]]

In [136]: d
Out[136]: [[[11, 22]]]

In [137]: []
```

`d = copy.deepcopy(c)`
会将c指向的空间进行递归copy

```
In [138]: a = [11, 22]
```

```
In [139]: b = (a,)
```

```
In [140]: c = [b]
```

```
In [141]:
```

```
In [141]: d = copy.deepcopy(c)
```

```
In [142]: c
```

```
Out[142]: [[11, 22],]
```

```
In [143]: d
```

```
Out[143]: [[11, 22],]
```

```
In [144]: id(c)
```

```
Out[144]: 4399912328
```

```
In [145]: id(d)
```

```
Out[145]: 4400358024
```

```
In [146]: id(c[0])
```

```
Out[146]: 4399871536
```

```
In [147]: id(d[0])
```

```
Out[147]: 4399936344
```

```
In [148]: a.append(33)
```

```
In [149]: c
```

```
Out[149]: [[11, 22, 33],]
```

```
In [150]: d
```

```
Out[150]: [[11, 22],]
```

