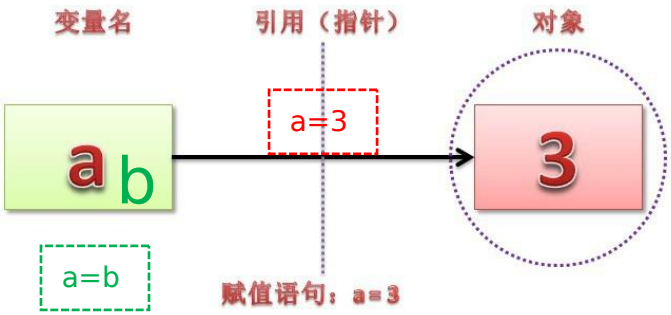
变量赋值初始化

在 python 中，所有的赋值操作都是引用操作。具体的实现：解释器先实例化一个类型或者类对象，再将该实例的引用赋给变量。执行 a=3 语句的，实际过程如下图所示。通过赋值=来完成其他变量赋值的方法，等效于其在将原变量的别名。例如执行语句 b=a，等效于给 a 再贴上 b 的标签。

更新模型	
分类	类型
可变类型	列表、字典
不可变类型	数字、字符串、元祖



对于所有类型的 python 内建类型，赋值初始化变量的过程以及通过变量初始化另外变量的过程都是引用及起别名的过程。表达式赋值方法，是一个变量赋值的新过程，实例化一个新类型或类来完成引用的过程。增量赋值的方法，对于不可变类型是新赋值过程，而可变类型则不生成新对象，直接原内存块后追加。

```
>>> a=(1,2,3)
>>> print a,id(a)
(1, 2, 3) 139686272735104
>>> b=a
>>> print b,id(b)
(1, 2, 3) 139686272735104
>>> a+=b
>>> print a,id(a)
(1, 2, 3, 1, 2, 3) 139686272701952
>>> a=[1,2,3]
>>> b=a
>>> print a,id(a)
[1, 2, 3] 139686272750440
>>> print b,id(b)
[1, 2, 3] 139686272750440
>>> a+=b
>>> print a,id(a)
[1, 2, 3, 1, 2, 3] 139686272750440
```

```
>>> a=[1,2,3]
>>> b=a
>>> print a,id(a)
[1, 2, 3] 139686272752528
>>> a=a+b
>>> print a,id(a)
[1, 2, 3, 1, 2, 3] 139686272752456
>>> a=[1,2,3]
>>> b=a
>>> print a,id(a)
[1, 2, 3] 139686272752672
>>> a+=b
>>> print a,id(a)
[1, 2, 3, 1, 2, 3] 139686272752672
```

表达式赋值

增量赋值

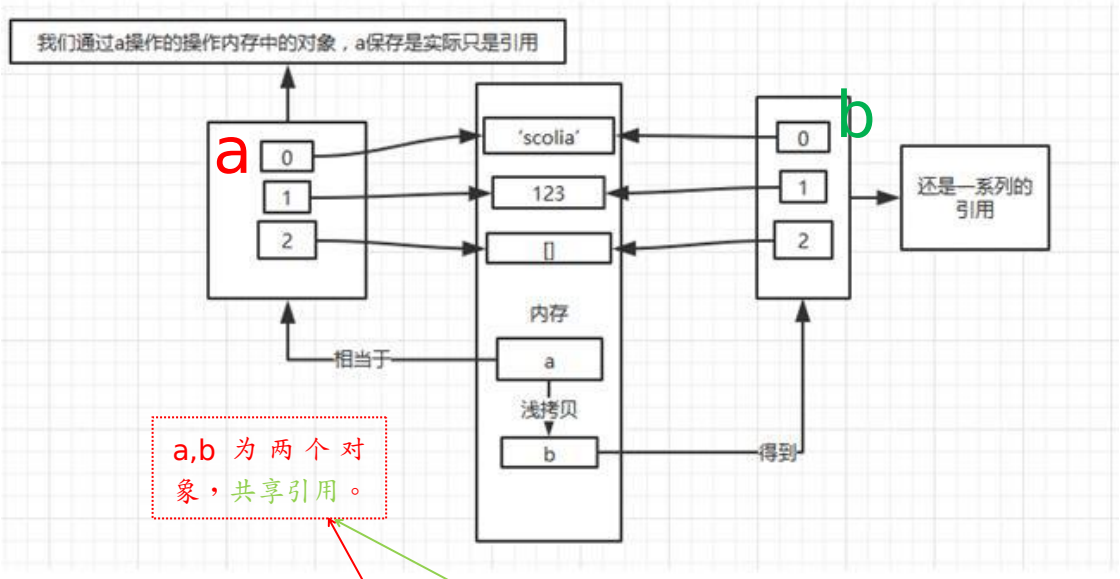
Idea1:对于可更改后续无用的变量，可以用增量赋值来减少内存的消耗。例如：a=a+b 与 a+=b,对于左边的式子是将 a 和 b 合并新件实例的过程，假如 a 变量后续不再使用需要出了作用域之外才会被回收;右边的式子是在 a 后面追加的过程。

Copy 与 deepcopy

浅拷贝,赋值时赋值原对象的引用，也即新的对象旧的内容。执行下面代码得到的结果：及起别名

a=['scolia',123,[],]

b=a[:]



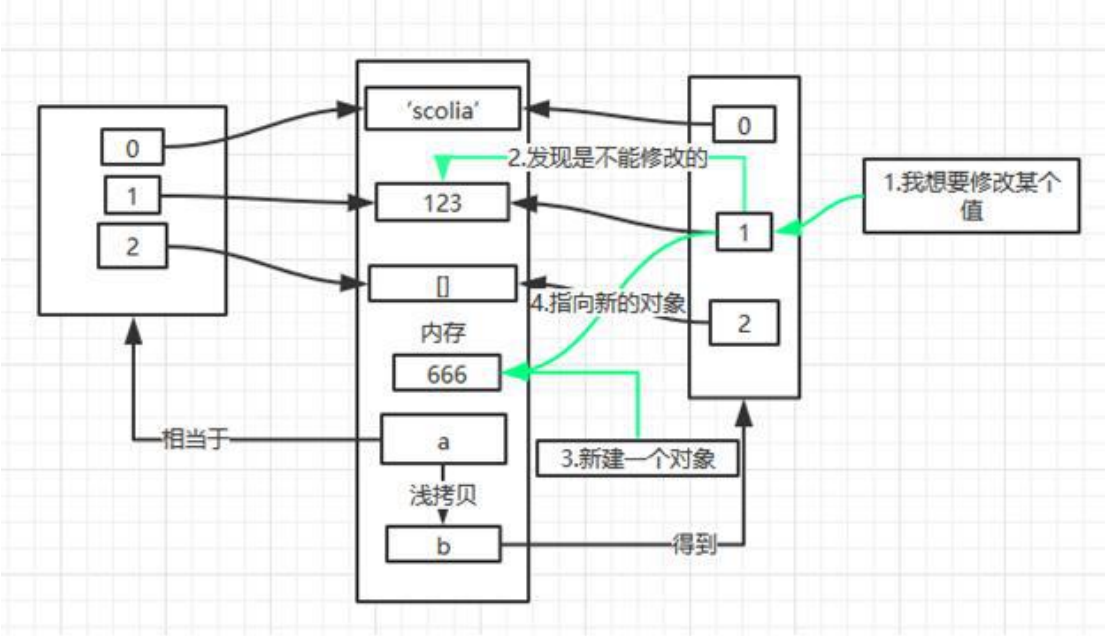

```
>>> a=['scolia',123,[],]
>>> b=a[:]
>>> print a,id(a),id(a[0]),id(a[1]),id(a[2])
['scolia', 123, []] 139686272831360 139686272786224 19524928 139686272856648
>>> print b,id(b),id(b[0]),id(b[1]),id(b[2])
['scolia', 123, []] 139686272752744 139686272786224 19524928 139686272856648
>>>
```

由此可见，a、b 两个对象共享引用。当更新其中的不可变类型变量时，是一个重新赋值的过程，需要更新引用;当更新可变类型变量时，并不需要更新引用，直接修改。执行下述代码时：

a=['scolia',123,[]共享引用，因为元组不可更改性；,]

b=a[:]

b[1]='666'



共享引用，因为元组不可更改性；

不可变类型更新过程

```
>>> a=['scolia',123,[],]
>>> b=a[:]
>>> print a,id(a),id(a[0]),id(a[1]),id(a[2])
['scolia', 123, []] 139686272831360 139686272786224 19524928 139686272856648
>>> print b,id(b),id(b[0]),id(b[1]),id(b[2])
['scolia', 123, []] 139686272752744 139686272786224 19524928 139686272856648
>>> b[1]=666
>>> print a,id(a),id(a[0]),id(a[1]),id(a[2])
['scolia', 123, []] 139686272831360 139686272786224 19524928 139686272856648
>>> print b,id(b),id(b[0]),id(b[1]),id(b[2])
['scolia', 666, []] 139686272752744 139686272786224 19877576 139686272856648
```

对象更新引用;当元素为可变对象时，则马上新建对象更新引用。

可变类型更新过程

```
>>> a=['scolia',123,[],]
>>> b=a[:]
>>> print a,id(a),id(a[0]),id(a[1]),id(a[2])
['scolia', 123, []] 139686272752672 139686272786224 19524928 139686272829488
>>> print b,id(b),id(b[0]),id(b[1]),id(b[2])
['scolia', 123, []] 139686272831360 139686272786224 19524928 139686272829488
>>> b[2].append('add')
>>> print a,id(a),id(a[0]),id(a[1]),id(a[2])
['scolia', 123, ['add']] 139686272752672 139686272786224 19524928 13968627282948
8
>>> print b,id(b),id(b[0]),id(b[1]),id(b[2])
['scolia', 123, ['add']] 139686272831360 139686272786224 19524928 13968627282948
8
```

创建浅拷贝的方式:

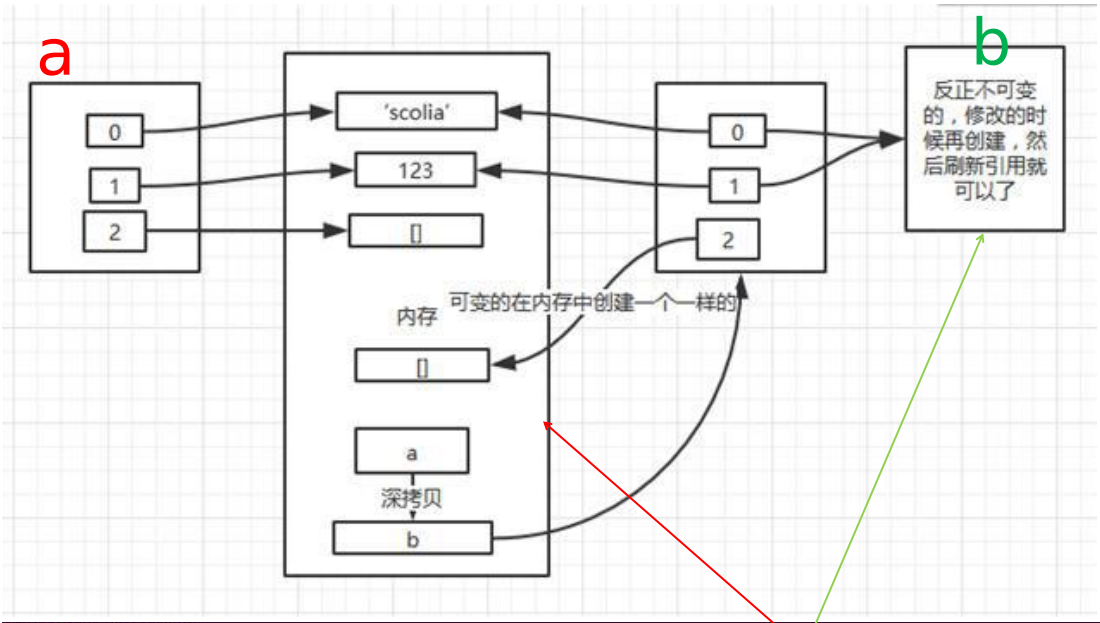
1.切片操作;

2.使用 list()工厂函数新建对象;（b = list(a)）

3.copy 模块中的 copy 方法。对象更新引用;当元素为可变对象时，则马上新建对象更新引用。

深拷贝，即是将原对象的所有内容拷贝新建一份，更新新对象的引用内容。执行下面代码的过程：

```
from copy import deepcopy
a=['scolia',123,[],]
b=deepcopy(a)
```




```
>>> import copy
>>> a=['scolia',123,[],]
>>> b=copy.deepcopy(a)
>>> print a,id(a),id(a[0]),id(a[1]),id(a[2])
['scolia', 123, []] 139686272752600 139686272786224 19524928 139686272752672
>>> print b,id(b),id(b[0]),id(b[1]),id(b[2])
['scolia', 123, []] 139686272856648 139686272786224 19524928 139686272856288
>>> b[2].append('add')
>>> print a,id(a),id(a[0]),id(a[1]),id(a[2])
['scolia', 123, []] 139686272752600 139686272786224 19524928 139686272752672
>>> print b,id(b),id(b[0]),id(b[1]),id(b[2])
['scolia', 123, ['add']] 139686272856648 139686272786224 19524928 13968627285628
8
```

创建浅拷贝的方式:

copy 模块中的 copy 方法。

分类	类型	浅拷贝	深拷贝
不可变	数值	起别名	起别名
	字符串	起别名	起别名
	简单元组	起别名	起别名
	嵌套元组	起别名	共享引用
可变	列表	共享引用	递归复制
	字典	共享引用	递归复制

嵌套元祖浅拷贝为上层为别名，因为其类型时不可更改的;嵌套元祖深拷贝为递归复制的过程。其内部的可变类型是一个赋值新建对象的过层。

```
>>> a=(1,2,3)
>>> b=copy.copy(a)
>>> c=copy.deepcopy(a)
>>> print a,id(a)
(1, 2, 3) 139686272837984
>>> print b,id(b)
(1, 2, 3) 139686272837984
>>> print c,id(c)
(1, 2, 3) 139686272837984
```

```
>>> a=(1,2,[3,4])
>>> b=copy.copy(a)
>>> c=copy.deepcopy(a)
>>> print a,id(a)
(1, 2, [3, 4]) 139686272735104
>>> print b,id(b)
(1, 2, [3, 4]) 139686272735104
>>> print c,id(c)
(1, 2, [3, 4]) 139686272736224
```

```
>>> b[2][1]=64
>>> print a,id(a)
(1, 2, [3, 64]) 139686272735104
>>> print b,id(b)
(1, 2, [3, 64]) 139686272735104
>>> print c,id(c)
(1, 2, [3, 4]) 139686272736224
```

列表和字典的浅拷贝都是两个对象共享引用的过程。深拷贝则是递归复制的过程。在拷贝时，起元素为不可变类型对象则先共享引用，待需要更新时再新建对象更新引用;当元素为可变对象时，则马上新建对象更新引用。

```
>>> a=[1,2,3]
>>> b=copy.copy(a)
>>> c=copy.deepcopy(a)
>>> print a,id(a)
[1, 2, 3] 139686272752744
>>> print b,id(b)
[1, 2, 3] 139686272856576
>>> print c,id(c)
[1, 2, 3] 139686272752672
```

```
>>> a=[1,2,[3,4]]
>>> b=copy.copy(a)
>>> c=copy.deepcopy(a)
>>> print a,id(a)
[1, 2, [3, 4]] 139686272829488
>>> print b,id(b)
[1, 2, [3, 4]] 139686272856648
>>> print c,id(c)
[1, 2, [3, 4]] 139686272856576
```

对象更新引用;当元素为可变对象时，则马上新建对象更新引用。

```
>>> b[2][1]=64
>>> print a,id(a)
[1, 2, [3, 64]] 139686272829488
>>> print b,id(b)
[1, 2, [3, 64]] 139686272856648
>>> print c,id(c)
[1, 2, [3, 4]] 139686272856576
>>> print a[2],id(a[2])
[3, 64] 139686272831360
>>> print b[2],id(a[2])
[3, 64] 139686272831360
>>> print c[2],id(c[2])
[3, 4] 139686272829128
>>>
```

Idea1:1.当有些信息为不可更改的属性，有些可更改时，可以再最上层使用

元组;全部可更改时用可变类型。2.对于可更改的信息为共享的信息时，则可以用浅拷贝生成;单人的则用深拷贝生成。