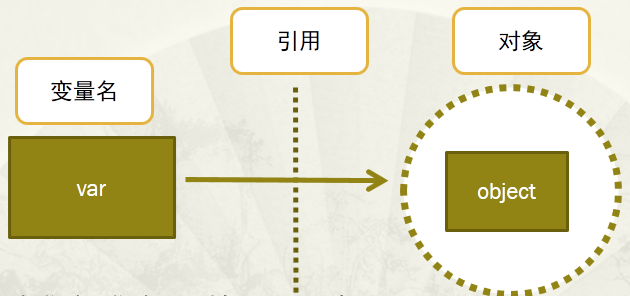
## [Python内存管理机制](https://www.cnblogs.com/geaozhang/p/7111961.html)

简介：

Python的GC模块主要运用了“引用计数”（reference counting）来跟踪和回收垃圾。在引用计数的基础上，还可以通过“标记-清除”（mark and sweep）解决容器对象可能产生的循环引用的问题。通过“分代回收”（generation collection）以空间换取时间来进一步提高垃圾回收的效率。

**一、变量与对象**

关系图如下：



1、变量，通过变量指针引用对象

　　变量指针指向具体对象的内存空间，取对象的值。

2、对象，类型已知，每个对象都包含一个头部信息（头部信息：类型标识符和引用计数器）

注意：

　　变量名没有类型，类型属于对象（因为变量引用对象，所以类型随对象），变量引用什么类型的对象，变量就是什么类型的。

[复制代码](javascript:void(0);)

In [32]: var1=object

In [33]: var2=var1

In [34]: id(var1)

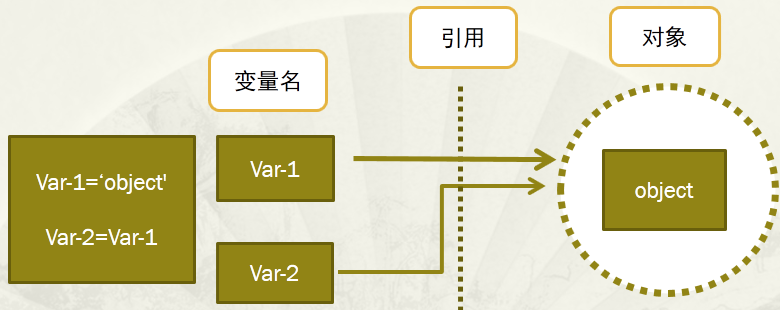
Out[34]: 139697863383968

In [35]: id(var2)

Out[35]: 139697863383968

[复制代码](javascript:void(0);)

PS：id()是python的内置函数，用于返回对象的身份，即对象的内存地址。



[复制代码](javascript:void(0);)

In [39]: a=123

In [40]: b=a

In [41]: id(a)

Out[41]: 23242832

In [42]: id(b)

Out[42]: 23242832

In [43]: a=456

In [44]: id(a)

Out[44]: 33166408

In [45]: id(b)

Out[45]: 23242832

[复制代码](javascript:void(0);)

3、引用所指判断

　　通过is进行引用所指判断，is是用来判断两个引用所指的对象是否相同。

整数

In [46]: a=1

In [47]: b=1

In [48]: print(a is b)

True

短字符串

In [49]: c="good"

In [50]: d="good"

In [51]: print(c is d)

True

长字符串

In [52]: e="very good"

In [53]: f="very good"

In [54]: print(e is f)

False

列表

In [55]: g=[]

In [56]: h=[]

In [57]: print(g is h)

False

由运行结果可知：

　　1、Python缓存了整数和短字符串，因此每个对象在内存中只存有一份，引用所指对象就是相同的，即使使用赋值语句，也只是创造新的引用，而不是对象本身；

　　2、Python没有缓存长字符串、列表及其他对象，可以由多个相同的对象，可以使用赋值语句创建出新的对象。

**二、引用计数**

　　在Python中，每个对象都有指向该对象的引用总数---引用计数

　　查看对象的引用计数：sys.getrefcount()

1、普通引用

[复制代码](javascript:void(0);)

In [2]: import sys

In [3]: a=[1,2,3]

In [4]: getrefcount(a)

Out[4]: 2

In [5]: b=a

In [6]: getrefcount(a)

Out[6]: 3

In [7]: getrefcount(b)

Out[7]: 3

[复制代码](javascript:void(0);)

注意：

　　当使用某个引用作为参数，传递给getrefcount()时，参数实际上创建了一个临时的引用。因此，getrefcount()所得到的结果，会比期望的多1。

2、容器对象

　　Python的一个容器对象(比如：表、词典等)，可以包含多个对象。

[复制代码](javascript:void(0);)

In [12]: a=[1,2,3,4,5]

In [13]: b=a

In [14]: a is b

Out[14]: True

In [15]: a[0]=6

In [16]: a

Out[16]: [6, 2, 3, 4, 5]

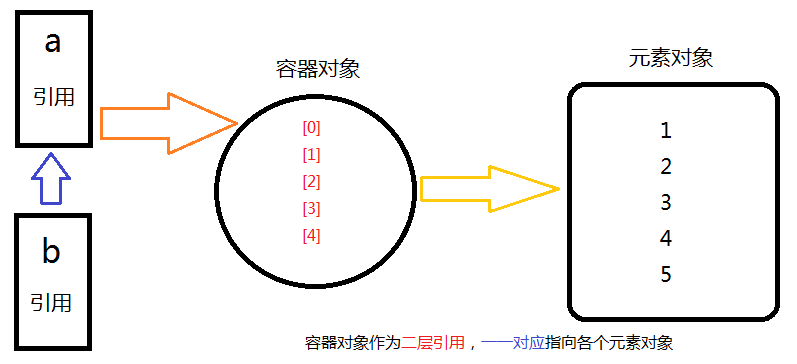
In [17]: a is b

Out[17]: True

In [18]: b

Out[18]: [6, 2, 3, 4, 5]

[复制代码](javascript:void(0);)



由上可见，实际上，容器对象中包含的并不是元素对象本身，是指向各个元素对象的引用。

3、引用计数增加

　　1、对象被创建

[复制代码](javascript:void(0);)

In [39]: getrefcount(123)

Out[39]: 6

In [40]: n=123

In [41]: getrefcount(123)

Out[41]: 7

[复制代码](javascript:void(0);)

　　2、另外的别人被创建

In [42]: m=n

In [43]: getrefcount(123)

Out[43]: 8

　　3、作为容器对象的一个元素

In [44]: a=[1,12,123]

In [45]: getrefcount(123)

Out[45]: 9

　　4、被作为参数传递给函数：foo(x)

4、引用计数减少

　　1、对象的别名被显式的销毁

In [46]: del m

In [47]: getrefcount(123)

Out[47]: 8

　　2、对象的一个别名被赋值给其他对象

In [48]: n=456

In [49]: getrefcount(123)

Out[49]: 7

　　3、对象从一个窗口对象中移除，或，窗口对象本身被销毁

[复制代码](javascript:void(0);)

In [50]: a.remove(123)

In [51]: a

Out[51]: [1, 12]

In [52]: getrefcount(123)

Out[52]: 6

[复制代码](javascript:void(0);)

　　4、一个本地引用离开了它的作用域，比如上面的foo(x)函数结束时，x指向的对象引用减1。

**三、垃圾回收**

　　当Python中的对象越来越多，占据越来越大的内存，启动垃圾回收(garbage collection)，将没用的对象清除。

1、原理

　　当Python的某个对象的引用计数降为0时，说明没有任何引用指向该对象，该对象就成为要被回收的垃圾。比如某个新建对象，被分配给某个引用，对象的引用计数变为1。如果引用被删除，对象的引用计数为0，那么该对象就可以被垃圾回收。

In [74]: a=[321,123]

In [75]: del a

2、解析del

　　del a后，已经没有任何引用指向之前建立的[321,123]，该表引用计数变为0，用户不可能通过任何方式接触或者动用这个对象，当垃圾回收启动时，Python扫描到这个引用计数为0的对象，就将它所占据的内存清空。

3、注意

　　1、垃圾回收时，Python不能进行其它的任务，频繁的垃圾回收将大大降低Python的工作效率；

　　2、Python只会在特定条件下，自动启动垃圾回收（垃圾对象少就没必要回收）

　　3、当Python运行时，会记录其中分配对象(object allocation)和取消分配对象(object deallocation)的次数。当两者的差值高于某个阈值时，垃圾回收才会启动。

In [93]: import gc

In [94]: gc.get\_threshold()　　#gc模块中查看阈值的方法

Out[94]: (700, 10, 10)

阈值分析：

　　700即是垃圾回收启动的阈值；

　　每10次0代垃圾回收，会配合1次1代的垃圾回收；而每10次1代的垃圾回收，才会有1次的2代垃圾回收；

当然也是可以手动启动垃圾回收：

In [95]: gc.collect() #手动启动垃圾回收

Out[95]: 2

4、何为分代回收

　　Python将所有的对象分为0，1，2三代；

　　所有的新建对象都是0代对象；

　　当某一代对象经历过垃圾回收，依然存活，就被归入下一代对象。

**四、内存池机制**

　　Python中有分为大内存和小内存：（256K为界限分大小内存）

1、大内存使用malloc进行分配

2、小内存使用内存池进行分配

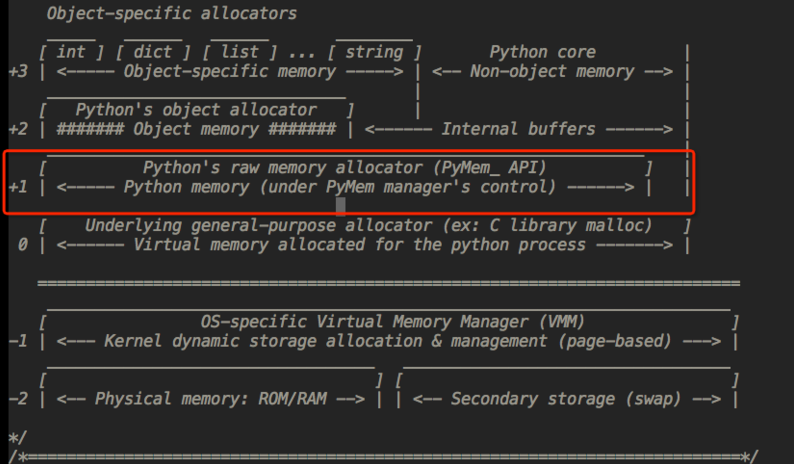
3、Python的内存池(金字塔)

　　第3层：最上层，用户对Python对象的直接操作

　　第1层和第2层：内存池，有Python的接口函数PyMem\_Malloc实现-----若请求分配的内存在1~256字节之间就使用内存池管理系统进行分配，调用malloc函数分配内存，但是每次只会分配一块大小为256K的大块内存，不会调用free函数释放内存，将该内存块留在内存池中以便下次使用。

　　第0层：大内存-----若请求分配的内存大于256K，malloc函数分配内存，free函数释放内存。

　　第-1，-2层：操作系统进行操作



**五、标记-清除**

“标记-清除”是为了解决循环引用的问题。可以包含其他对象引用的容器对象（比如：list，set，dict，class，instance）都可能产生循环引用。  
我们必须承认一个事实，如果两个对象的引用计数都为1，但是仅仅存在他们之间的循环引用，那么这两个对象都是需要被回收的，也就是说，它们的引用计数虽然表现为非0，但实际上有效的引用计数为0。我们必须先将循环引用摘掉，那么这两个对象的有效计数就现身了。假设两个对象为A、B，我们从A出发，因为它有一个对B的引用，则将B的引用计数减1；然后顺着引用达到B，因为B有一个对A的引用，同样将A的引用减1，这样，就完成了循环引用对象间环摘除。  
但是这样就有一个问题，假设对象A有一个对象引用C，而C没有引用A，如果将C计数引用减1，而最后A并没有被回收，显然，我们错误的将C的引用计数减1，这将导致在未来的某个时刻出现一个对C的悬空引用。这就要求我们必须在A没有被删除的情况下复原C的引用计数，如果采用这样的方案，那么维护引用计数的复杂度将成倍增加。

原理：“标记-清除”采用了更好的做法，我们并不改动真实的引用计数，而是将集合中对象的引用计数复制一份副本，改动该对象引用的副本。对于副本做任何的改动，都不会影响到对象生命走起的维护。  
这个计数副本的唯一作用是寻找root object集合（该集合中的对象是不能被回收的）。当成功寻找到root object集合之后，首先将现在的内存链表一分为二，一条链表中维护root object集合，成为root链表，而另外一条链表中维护剩下的对象，成为unreachable链表。之所以要剖成两个链表，是基于这样的一种考虑：现在的unreachable可能存在被root链表中的对象，直接或间接引用的对象，这些对象是不能被回收的，一旦在标记的过程中，发现这样的对象，就将其从unreachable链表中移到root链表中；当完成标记后，unreachable链表中剩下的所有对象就是名副其实的垃圾对象了，接下来的垃圾回收只需限制在unreachable链表中即可。

**六、分代回收**

背景：分代的垃圾收集技术是在上个世纪80年代初发展起来的一种垃圾收集机制，一系列的研究表明：无论使用何种语言开发，无论开发的是何种类型，何种规模的程序，都存在这样一点相同之处。即：一定比例的内存块的生存周期都比较短，通常是几百万条机器指令的时间，而剩下的内存块，起生存周期比较长，甚至会从程序开始一直持续到程序结束。  
从前面“标记-清除”这样的垃圾收集机制来看，**这种垃圾收集机制所带来的额外操作实际上与系统中总的内存块的数量是相关的，当需要回收的内存块越多时，垃圾检测带来的额外操作就越多，而垃圾回收带来的额外操作就越少；反之，当需回收的内存块越少时，垃圾检测就将比垃圾回收带来更少的额外操作**。为了提高垃圾收集的效率，采用“空间换时间的策略”。

原理：将系统中的所有内存块根据其存活时间划分为不同的集合，每一个集合就成为一个“代”，垃圾收集的频率随着“代”的存活时间的增大而减小。也就是说，活得越长的对象，就越不可能是垃圾，就应该减少对它的垃圾收集频率。那么如何来衡量这个存活时间：通常是利用几次垃圾收集动作来衡量，如果一个对象经过的垃圾收集次数越多，可以得出：该对象存活时间就越长。

举例说明：

当某些内存块M经过了3次垃圾收集的清洗之后还存活时，我们就将内存块M划到一个集合A中去，而新分配的内存都划分到集合B中去。当垃圾收集开始工作时，大多数情况都只对集合B进行垃圾回收，而对集合A进行垃圾回收要隔相当长一段时间后才进行，这就使得垃圾收集机制需要处理的内存少了，效率自然就提高了。在这个过程中，集合B中的某些内存块由于存活时间长而会被转移到集合A中，当然，集合A中实际上也存在一些垃圾，这些垃圾的回收会因为这种分代的机制而被延迟。  
在Python中，总共有3“代”，也就是Python实际上维护了3条链表。具体可以查看Python源码详细了解。