一、python内存管理

这个问题需要从三个方面来说：

- 1）对象的引用计数机制（四增五减）
- 2）垃圾回收机制（手动自动，分代回收）
- 3）内存池机制（大m小p）

1）对象的引用计数机制

要保持追踪内存中的对象，Python使用了引用计数这一简单的技术。sys.getrefcount(a)可以查看a对象的引用计数，但是比正常计数大1，因为调用函数的时候传入a，这会让a的引用计数+1

a）增加引用计数

对象被创建：x = 3.14

另外的别名被创建：y = x

对象被作为参数传递给函数（新的本地引用）：foobar(x)

对象成为容器对象的一个元素：myList = [123, x, ‘xyz’]

b）减少引用计数

对象的一个别名被赋值给其他对象：x = 123

对象的别名被显式销毁：del y

一个本地引用离开了其作用范围。如fooc()函数结束时，func函数中的局部变量（全局变量不会）

对象被从一个窗口对象中移除：myList.remove(x)

窗口对象本身被销毁：del myList

c）引用计数例子，加深理解

id()获取对象的内存地址

在Python中，整数和短小的字符，Python都会缓存这些对象，以便重复使用。当我们创建多个等于1的引用时，实际上是让所有这些引用指向同一个对象

```python
#is用于判断两个引用所指的对象是否相同。
a = 1
b = 1

a is b
#True

print(id(a))
#505348560

print(id(b))
#505348560

a = 'good'
b = 'good'
a is b
#True
```


让我们来看看较长的字符串：

```python
a = "very good morning"
b = "very good morning"

id(a)
#57960680
id(b)
#57960968

a is b
#False
```


sys.getrefcount()来获取对象的引用计数：

```python
import sys
a = [1 ,2, 3]

print( sys.getrefcount(a) )
#2

b = a
print( sys.getrefcount(a) )
#3
```

2）垃圾回收机制

吃太多，总会变胖，Python也是这样。当Python中的对象越来越多，它们将占据越来越大的内存。不过你不用太担心Python的体形，它会在适当的时候“减肥”，启动垃圾回收(garbage collection)，将没用的对象清除

从基本原理上，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了

比如某个新建对象，它被分配给某个引用，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。比如下面的表

```
a = [1, 2, 3]

sys.getrefcount(a)
#2
del a
```


del a后，已经没有任何引用指向之前建立的[1, 2, 3]这个表。这个对象如果继续待在内存里，就成了不健康的脂肪。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

然而，减肥是个昂贵而费力的事情。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。如果内存中的对象不多，就没有必要总启动垃圾回收

所以，Python只会在特定条件下，自动启动垃圾回收。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

我们可以通过gc模块的get_threshold()方法，查看该阈值：

```
import gc
gc.get_threshold()

#(700, 10, 10)
```


返回(700, 10, 10)，后面的两个10是与分代回收相关的阈值，后面可以看到。700即是垃圾回收启动的阈值。可以通过gc中的set_threshold()方法重新设置。

我们也可以手动启动垃圾回收，即使用gc.collect()

分代回收：

Python同时采用了分代(generation)回收的策略。这一策略的基本假设是，存活时间越久的对象，越不可能在后面的程序中变成垃圾。

我们的程序往往会产生大量的对象，许多对象很快产生和消失，但也有一些对象长期被使用。出于信任和效率，对于这样一些“长寿”对象，我们相信它们的用处，所以减少在垃圾回收中扫描它们的频率

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，那么它就被归入下一代对象。垃圾回收启动时，一定会扫描所有的0代对象

如果0代经过一定次数垃圾回收，那么就启动对0代和1代的扫描清理。当1代也经历了一定次数的垃圾回收后，那么会启动对0，1，2，即对所有对象进行扫描

这两个次数即上面get_threshold()返回的(700, 10, 10)返回的两个10。也就是说，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

同样可以用set_threshold()来调整，比如对2代对象进行更频繁的扫描

```
import gc
gc.set_threshold(700, 10, 5)
```

3）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

python中的内存管理机制都有两套实现，一套是针对小对象，就是大小小于256K时，pymalloc会在内存池中申请内存空间；当大于256K时，则会直接执行系统的malloc的行为来申请内存空间。