**Python之内存管理机制**

Python使用gc模块处理python对象以及python垃圾回收器的工作，完整的gc模块文档参考[这里](https://docs.python.org/2/library/gc.html)。举gc模块的几个例子看一下：

* gc.enable()——可自动进行垃圾回收；
* gc.disable()——不可自动进行垃圾回收；
* gc.set\_threshold()——设置python垃圾回收的阈值；
* gc.set\_debug()——设置垃圾回收的调试标记，调试信息会被写入std.err；
* gc.get\_objects()——返回收集器跟踪的所有对象的列表，不包括返回的列表。
* ......

Python的内存管理机制主要包括三个方面：引用计数机制、垃圾回收机制、内存池机制

**一、引用计数机制**

Python内部使用引用计数来保持追踪内存中的对象，所有对象都有引用计数。

引用**计数增加**的情况：

（1）对象被创建：x=1；

（2）被对象已有引用赋值：y=x；

（3）被作为参数传递给函数：f（x）；

（4）将其放入一个人容器中，如列表、元组或字典。

引用**计数减少**的情况：

（1）一个本地引用离开了它的作用域。比如上面的f（x）函数结束时，x指向的对象引用就会减1；

（2）使用del语句对对象别名显式的销毁：del x；

（3）引用被重新赋值：x=2；

（4）对象从一个窗口对象中移除：myList.remove(x)；

（5）窗口对象本身被销毁：del myList，或窗口对象本身离开了作用域：f（myList）

使用**sys.getrefcount()**函数可以获得对象的当前引用计数。大多数情况下，引用计数比你猜测得要大得多。

对于**不可变数据**（如数字和字符串），解释器会在程序的不同部分共享内存，以便节约内存。

**二、垃圾回收**

（1）引用计数

引用计数也是一种垃圾收集机制，而且是一种最直观、最简单的垃圾收集技术。党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，该对象就成为要回收的垃圾，被回收掉。但是有一个例外，循环引用是对象之间的相互引用，会使得一组对象的引用计数不为0，然后这些对象实际上没有被任何外部对象所引用，这些对象就会占内存永远不会被释放掉。

因此Python又引入了其他的垃圾回收机制来弥补引用计数的缺陷：“标记-清理”，“分代回收”。

（2）标记清理

对于循环引用，当两个对象相互引用时，del语句可以减少a和b的引用计数，并销毁用于引用底层对象的名称。然而由于每个对象都包含对其他对象的引用，因此引用计数不会归零，对象也不会销毁；从而导致内存泄漏。为了解决这一问题，解释器会定期执行一个循环检测器，搜索不可访问的对象的循环并删除它们。

在实际操作中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周期的维护。）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。

（3）分代回收

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。  
举个例子来说明：   
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。

**三、内存池机制**

Python提供了对内存的垃圾收集机制，但是它是将不用的内存放到内存池，而不是返回给系统。

python中内存机制呈现出金字塔形状，-1、-2层主要由操作系统进行操作。第0层是c中的malloc，free等内存分配和释放函数进行操作。

（1）第1层和第2层是**内存池**，有Python的接口函数PyMem\_Malloc函数实现。Python 中所有小于256个字节的对象都使用pymalloc实现的分配器，而大的对象则使用系统的malloc。不过，通过修改Python源代码，我们可以改变这个默认值，从而改变Python的默认内存管理行为。

Python引入了一个内存池机制，是为了加速Python的执行效率，用于管理对小块内存的申请和释放。

（3）对于python对象，如整数、浮点数、List，都有其独立的私有内存池，对象间不共享它们的内存池。也就是说，如果你分配又释放了大量的整数，用于缓存这些整数的内存就不能再分配给浮点数。（值语义）

