对象引用机制、垃圾回收机制、内存池机制

1. 对象引用机制

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)一个对象分配一个新名称

2)将其放入一个容器中（如列表、元组或字典）

引用计数减少的情况：

1，使用del语句对对象别名显示的销毁

2，引用超出作用域或被重新赋值

sys.getrefcount( )函数可以获得对象的当前引用计数

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。

1. 垃圾回收

1，当一个对象的引用计数归零时，它将被垃圾收集机制处理掉。

2，当两个对象a和b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一个对其他对象的应用，因此引用计数不会归零，对象也不会销毁。（从而导致内存泄露）。为解决这一问题，解释器会定期执行一个循环检测器，搜索不可访问对象的循环并删除它们。

3. 内存池机制

Python提供了对内存的垃圾收集机制，但是它将不用的内存放到内存池而不是返回给操作系统。

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

2，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。

3，对于Python对象，如整数，浮点数和List，都有其独立的私有内存池，对象间不共享他们的内存池。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。

<!-- Python垃圾回收机制 -->

Python GC主要使用引用计数（reference counting）来跟踪和回收垃圾。在引用计数的基础上，通过“标记-清除”（mark and sweep）解决容器对象可能产生的循环引用问题，通过“分代回收”（generation collection）以空间换时间的方法提高垃圾回收效率。

1 引用计数

PyObject是每个对象必有的内容，其中ob\_refcnt就是做为引用计数。当一个对象有新的引用时，它的ob\_refcnt就会增加，当引用它的对象被删除，它的ob\_refcnt就会减少.引用计数为0时，该对象生命就结束了。

优点:

1.简单

2.实时性

缺点:

1.维护引用计数消耗资源

2.循环引用

2 标记-清除机制

基本思路是先按需分配，等到没有空闲内存的时候从寄存器和程序栈上的引用出发，遍历以对象为节点、以引用为边构成的图，把所有可以访问到的对象打上标记，然后清扫一遍内存空间，把所有没标记的对象释放。

3 分代技术

分代回收的整体思想是：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，每个集合就成为一个“代”，垃圾收集频率随着“代”的存活时间的增大而减小，存活时间通常利用经过几次垃圾回收来度量。

Python默认定义了三代对象集合，索引数越大，对象存活时间越长。

举例：

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。

1. http状态码
   1. 401-未授权，需要进行身份认证或者已提供身份认证但是没有通过
   2. 403-拒绝访问，没有权限，表示资源不可用，服务器理解客户的请求，但拒绝处理它，通常由于服务器上文件或目录的权限设置导致的 WEB 访问错误
2. TCP协议
   1. 三次握手
      1. 客户端请求服务端，发送包SYN(seq=j)
      2. 服务端接收客户端的SYN包，发送一个SYN(seq=k)，一个确认包ACK(seq=j+1)
      3. 客户端收到服务端的包，向服务端发送确认包ACK(seq=k+1)，客户端与服务端连接成功，完成三次握手
   2. 四次挥手
      1. 客户端需要释放链接，向客户端发送TCP报文，标志位为FIN(seq=n)，客户端进入FIN-WAIT-1阶段，即半关闭，因为客户端发送数据完成，服务端可能存在数据没有发送完成
      2. 服务端接收到报文，先向客户端发送ACK=n+1，告诉客户端，已收到请求，请客户端继续等待消息，这时候客户端进入FIN-WAIT-2阶段，继续等待服务器的FIN报文
      3. 服务端发送完所有数据后，向客户端发送报文FIN(seq=m)，告诉客户端，服务端已准备好关闭，服务器进入LAST\_ACK状态
      4. 客户端收到FIN(seq=m)后，知道可以关闭连接了，但是它为防止网络延迟，影响服务端接收消息进行关闭的时间，向服务端发送ACk(seq=m+1)，进入TIME\_WAIT状态，如果服务端没有收到ACK可以重传，服务端收到ACK后，进行关闭连接，客户端等待一段时间发现服务端没有回复，知道服务端已关闭，此时，客户端进行关闭连接，完成四次握手
3. Http请求中get和post的区别
   1. GET 请求是通过 URL 直接请求数据，数据信息可以在 URL 中直接看到，比如浏览器访问；而 POST 请求是放在请求头中的，我们是无法直接看到的。
   2. GET 提交有数据大小的限制，一般是不超过 1024 个字节，而这种说法也不完全准确，HTTP 协议并没有设定 URL 字节长度的上限，而是浏览器做了些处理，所以长度依据浏览器的不同所有不同；POST 请求在 HTTP 协议中也没有做说明，一般来说是没有设置限制的，但是实际上浏览器也有默认值。总体而言，少量的数据使用 GET，大量的数据使用 POST。
   3. GET 请求因为数据参数是暴露在 URL 中的，所以安全性比较低，比如密码是不能暴露的，就不能请求使用 GET；POST 请求中，请求参数信息是放在请求头的，所以安全性较高，可以使用。在实际中，涉及到登陆操作的时候，尽量使用 HTTP 请求，安全性更好。
4. 多线程和多进程
   1. 进程：
      1. 1. 操作系统进行资源分配和调度的基本单位，多个进程之间相互独立；
      2. 2. 稳定性好，如果一个进程崩溃，不影响其他进程，但是进程消耗资源大，开启的进程数量有限
   2. 线程：
      1. CPU 进行资源分配和调度的基本单位，线程是进程的一部分，是比进程更小的、能独立运行的基本单位，一个进程下的多个线程可以共享该进程的所有资源。
      2. 如果 IO 操作密集，则可以多线程运行效率高，缺点是如果一个线程崩溃，都会造成进程的崩溃。
   3. 运用：
      1. IO 密集的用途多线程，在用户输入 sleep 时候，可以切换到其他线程执行，减少等待的时间；
      2. CPU 密集的用多进程，因为加入 IO 操作少，用多线程的话，因为线程共享一个全局解释器锁，当前运行的线程会霸占 GIL，其他线程没有 GIL，就不能充分利用多核 CPU 的优势。