1. gc要完成哪些事情

（1）哪些内存需要被回收

（2）什么时候回收

（3）如何回收

java内存运行时区域的各个部分，其中程序计数器、虚拟机栈、本地方法栈3个区域随线程创建而生，随线程结束而亡。栈的栈帧跟着方法进入和退出而有条不紊的执行着出栈和入栈。每一个栈帧中分配多少内存基本上是类结构确定下来就已经知道的。因此这几个区域的内存分配与回收都是具有去欸你高兴的，在这几个区域与是不需要过多的考虑垃圾回收问题，因为方法结束或线程结束的时候，内存自然就回收了，而java堆和方法区则不一样，一个接口中的多少个是西安类需要的内存可能不一样，一个方法中多少分支需要多少内存也是不一样的，只有运行时才知道哪些对象需要创建，内存分配时动态的。

2. 如何判断哪些对象需要回收

即如何判断哪些对象还“活着“

（1）引用计数器

当一个对象被引用的时候，即引用计数器加1，当失效的时候减1，所以当引用为0的时候，即这个对象不会再被引用，即可被回收

在python， actionscript3中的flashplayer中被使用，效率比较高，但是主流的jvm并没有采用这种方式，因为当对象出现循环引用的时候，如果采用这个算法，即使这两个对象不被引用，也没法被有效的回收（使用-XX:+PrintGCDetails）可以查询gc情况

（2）可达性分析算法

基本思路就是通过一些列的gc roots对象作为起始点、从这些节点的开始向下搜索，搜索所走过的路径成为引用链，当一个对象到gc roots没有任何引用链相连时，则证明这个对象是不可用的，因此判断为可回收对象

哪些对象可作为gc roots：

（1）虚拟机栈（栈帧中本地变量表）中引用的对象

（2）方法区中类静态属性引用的对象

（3）方法区中常量引用的对象

（4）本地方法区中JNI（即一般说的native方法）引用的对象

java引用：
（1）强引用（StrongReference）：在java程序代码中普遍存在，即 new obj(); 这类引用只要强引用存在，垃圾收集器永远不会回收掉被引用对象
（2）软引用（SoftReference）：描述一些还有用但是非必须的对象，在系统将要发生内存溢出之前，将对这些对象列进行第二次回收，如果这次回收还没有走狗的内存，就会抛出内存溢出
（3）弱引用（WeakReference）：非必须的对象，但是它的强度要比弱引用更弱一些，被弱引用关联的对象只能生存到下一次垃圾回收之前
（4）虚引用（PhantomReference）：虚引用也称幽灵引用或则幻影引用，它是最弱的一种引用关系。一个对象是否有虚引用的存在，完全不会对其生存时间构成影响，也没法通过虚引用获得一个对象实例。为一个对象设置虚引用的唯一目的就是能在这个对象被垃圾回收之前收到一个系统通知

对象是生存还是死亡
及时一个对象可达性分析算法中不可达，也并非是非死不可的，这个时候只是暂时处于缓刑，而真正宣告死亡需要经历两个标记过程
（1）如果当进行可达性分析发现没有与gc roots相连接的引用链，那么它将会被第一次编辑并惊醒一次筛选，筛选的条件是此对象是否有必要执行finalize()方法。当对象没有覆盖finalize()方法，或者finalize方法已经被调用，就会被视为没有必要执行
（2）如果被认定为可以执行finalize方法，那么这个对象将会放置到一个叫做F-Queue的队列中，并在稍后会由一个虚拟机自动创建的、低优先级的Finalizer线程去执行它，这里所谓的执行就是触发finalize方法，而不是等待这个方法结束，因为有可能方法里执行时间过长以及死循环等。finalize方法是对象逃离死亡的最后一次机会，稍后gc会对F-Queue中的对象进行第二次小规模的标记、如果对象进行了自救（只需要重新与引用链上的任何一个对象建立联系即可），如果自救了，那么它就会被移除“即将回收”的队列中，否则就真的被回收了

finalize方法有且只会被调用一次，但是并不鼓励在程序中重写这个方法，它的运行代价昂贵、不确定性大、无法保证各个对象的调用顺序