 Java 中使用 new、newarray、anewarray 和 multianewarray 指令来创建的对象，当这些对象不再使用时由垃圾收集来释放。那么 反序列化等都是间接使用了前面的某个指令, clone() 是个本地方法？ JVM 规范不需要任何特定的垃圾收集技术，甚至也没要求有垃圾收集机制。大概只是说不需要手工释放内存，具体怎么实现各 JVM 自行决定。 GC 除了释放不再被引用的对象，还要处理堆碎片，整理出连续的空闲空间才能放得下新的对象。不至于出现总的空闲空间足够，但碎片太多而报出 "Out of Memory" 的异常。 GC 有两个好处：一个是提高了生产率，不用埋头于 Memory Link 的有时甚至是逐行的检查；二，GC 也是 Java 安全策略的一部分，有了它不至于因错误的释放内存而导至 JVM 崩溃。但是 GC 的一个潜在缺陷影响了程序的性能，它需要一直在后台不时的做些事情，而且实时性也有所欠缺。垃圾收集算法 GC 算法要做两件基本的事情：1. 检测出垃圾对象；2. 回收垃圾对象，释放相应堆空间。垃圾检测一般是先建立一个根对象集合，其他对象要是从根对象起可触及就是活的，无法到达的就是垃圾。这里的根对象的认定就有些讲究的，不同的 JVM 的看法不完全一致，但总是会包含局部变量中的对象引用和栈帧的操作数栈(以及类变量中的引用)。 另一个根对象的来源是被加载的类的常量池中的对象引用。类的常量池中的字符串包括有类名、超类名、超接口名、字段名、字段特征签名、方法名、方法特征签名 还有一个来

源是传递到方法中的、没有被本地方法“ 释放 ”的对象引用(根据本地方法接口，本地方法可以通过简单的返回来释放引用，或者显式的调用一个回调函数来释放传递来的引用，或是这两者的结合)。 再一个潜在的根对象来源是，JVM 运行时数据区中从垃圾收集器的堆中分配的部分。某些实现中，方法区中的类数据本身可能被存放在使用垃圾收集器的堆中，以便使用和释放对象同样的垃圾收集算法来检测和卸载不再被引用的类。 在某些 JVM 实现中，像基本类型，如一个 int 如果被解释为一个本地指针，那它就是指向堆中的一个对象，但是保守的垃圾收集器对这种基本类型引用的堆中的对象不处理。 区别活动对象和垃圾的两个基本方法是引用计数和跟踪。 引用计数收集器 引用计数是垃圾收集的早期策略，这种方法时堆中的每个对象都有一个引用计数。当对象创建并赋给一个变量时，引用计数为 1。每次赋给别的变量时，引用计数加 1，当对象的引用超过了生存期或指向到了新值(如果引用置为 null)，对象的引用计减 1。这样对象的引用计数为 0 时就是垃圾，可清除的。引用计数对多个对象的循环引用无能为力，其实这些对象都是死的，但引用计数都不为 0，还有引用数的增减带来额外开销，基于这些缺陷，这种技术现在已经不为人所接受了。 跟踪收集器 跟踪收集追踪从根节点开始的对象引用图。基本的追踪算法叫作“ 标记并清除 ”，也就是垃圾收集的两个阶段。标记阶段，垃圾收集器遍历引用树，标记每一个遇到的对象。清除阶段，未被标记的对象被释放。可能在对象本身设置标记，要么就是用一个独立的位图来设置标记。 压缩收集器 垃圾收集同时要应对碎片整理的任务。标记和清除通常使用两种策略来消除堆碎片：

压缩和拷贝，这两种方法都是快速移动对象来减小碎片。 压缩收集我想应该是在标记清除之后来做的？压缩收集器把活动对象越过空闲区滑到堆的一堆，留下另一端的大的连续空闲块。被移动的对象的引用也被更新，指向新的位置。更新被移动对象的引用有时通过一个间接对象引用层来实现的，对象的引用不实际指向堆中对象，而是指向一个对象句柄表(由它完成对象引用到堆中对象的实际位置的映射)，对象被移动了，只需要更新对象句柄表的句柄值，这样程序中的对象引用不变。这种方法简化了消除堆碎片的工作，但是每次对象访问都要查一下映射表，带来了性能上的损失。 拷贝收集器 拷贝收集器把所有的活动对象移动到一个新的区域。这种方法在追踪对象过程中随着发现而被拷贝，不再有标记和清除的区分。一般的拷贝收集算法称为“ 停止并拷贝 ”。在这个方案中，堆被分为两个区域，任何时候只使用其中一个区域。对象在某一个区域中分配，直到这个区域被耗尽时，程序执行停止，遍历这个区域，活动对象移到另一个区域，完成后程序恢复执行，对象在新的区域分配。原来的区域剩下垃圾，全清除。直到新的区域耗尽时，程序停止，活动对象又往回移，循环工作。这种方法的代价就是堆内存只能使用到一半。 下面是“ 停止和拷贝 ”算法的垃圾收集过程中以时间为线索的 9 个快照： 看过这个图，应该不用多加解释，反正就是堆分成上下两个区域，哪部分满了，活动对象往另一部分跑，被移出的区域剩下的对象全是垃圾，可以哗一下全清空。来来回回，新对象总是在正用的那部份分配。想想你在运行 Java 程序的时候应该有过突然被中止不动的时候，可能就是 GC 在活动了。 分代收集器 简单的停止拷贝收集器的

缺点是，每次收集时，所有的活动对象都要移动来移动去。对于短生命的对象还好说，经常可以就地解决掉，可是对于长生命周期的对象就纯粹是个体力劳动了，把它挪来挪去除消耗大量的时间，没有产生任何效益。分代收集能直接让长生命周期的对象长时间的呆在一个地方按兵不动。GC 的精力可以更多的花在收集短命对象上。 这种方法里，堆被分成两个或更多的子堆，每一个堆为一“ 代 ”对象服务。最年幼的那一代进行最频繁的垃圾收集。因为多数对象是短命的，只有很小部分的年幼对象可以在经历第一次收集后还存活。如果一个最年幼的对象经历了好几次垃圾收集后仍是活着的，那这个对象就成为寿命更高的一代，它被转移到另外一个子堆中去。年龄更高一代的收集没有年轻一代来得频繁。每当对象在所属的年龄代中变得成熟(多次垃圾收集后仍幸存)之后，就可以转移到更高年龄的一代中去。 分代收集除了可应用于拷贝算法，也可以应用于标记清除算法。不管在哪种情况下，把堆按照对象年龄分组可以提高最基本的垃圾收集的性能。 