SCJP考试辅导：Java的垃圾收集机制 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/145/2021_2022_SCJP_E8_80_83_E8_AF_95_c104_145541.htm 通常，我们把分配出去后，却无法回收的内存空间称为"内存渗漏体（Memory Leaks）"。以上这种程序设计的潜在危险性在Java这样以严谨、安全著称的语言中是不允许的。但是Java语言既不能限制程序员编写程序的自由性，又不能把声明对象的部分去除（否则就不是面向对象的程序语言了），那么最好的解决办法就是从Java程序语言本身的特性入手。于是，Java技术提供了一个系统级的线程（Thread），即垃圾收集器线程（Garbage Collection Thread），来跟踪每一块分配出去的内存空间，当Java 虚拟机（Java Virtual Machine）处于空闲循环时，垃圾收集器线程会自动检查每一快分配出去的内存空间，然后自动回收每一快可以回收的无用的内存块。 垃圾收集器线程是一种低优先级的线程，在一个Java程序的生命周期中，它只有在内存空闲的时候才有机会运行。它有效地防止了内存渗漏体的出现，并极大可能地节省了宝贵的内存资源。但是，通过Java虚拟机来执行垃圾收集器的方案可以是多种多样的。 下面介绍垃圾收集器的特点和它的执行机制： 垃圾收集器系统有自己的一套方案来判断哪个内存块是应该被回收的，哪个是不符合要求暂不回收的。垃圾收集器在一个Java程序中的执行是自动的，不能强制执行，即使程序员能明确地判断出有一块内存已经无用了，是应该回收的，程序员也不能强制垃圾收集器回收该内存块。程序员唯一能做的就是通过调用System. gc 方法来"建议"执行垃圾收集器，但其是否可以执行，什么时候执行却都是

不可知的。这也是垃圾收集器的最主要的缺点。当然相对于它给程序员带来的巨大方便性而言，这个缺点是瑕不掩瑜的。 垃圾收集器的主要特点 1．垃圾收集器的工作目标是回收已经无用的对象的内存空间，从而避免内存渗漏体的产生，节省内存资源，避免程序代码的崩溃。 2．垃圾收集器判断一个对象的内存空间是否无用的标准是：如果该对象不能再被程序中任何一个"活动的部分"所引用，此时我们就说，该对象的内存空间已经无用。所谓"活动的部分"，是指程序中某部分参与程序的调用，正在执行过程中，尚未执行完毕。 3．垃圾收集器线程虽然是作为低优先级的线程运行，但在系统可用内存量过低的时候，它可能会突发地执行来挽救内存资源。当然其执行与否也是不可预知的。 4．垃圾收集器不可以被强制执行，但程序员可以通过调用System. gc方法来建议执行垃圾收集器。 5．不能保证一个无用的对象一定会被垃圾收集器收集，也不能保证垃圾收集器在一段Java语言代码中一定会执行。因此在程序执行过程中被分配出去的内存空间可能会一直保留到该程序执行完毕，除非该空间被重新分配或被其他方法回收。由此可见，完全彻底地根绝内存渗漏体的产生也是不可能的。但是请不要忘记，Java的垃圾收集器毕竟使程序员从手工回收内存空间的繁重工作中解脱了出来。设想一个程序员要用C或C 来编写一段10万行语句的代码，那么他一定会充分体会到Java的垃圾收集器的优点！ 6．同样没有办法预知在一组均符合垃圾收集器收集标准的对象中，哪一个会被首先收集。 7．循环引用对象不会影响其被垃圾收集器收集。 8．可以通过将对象的引用变量（reference variables，即句柄handles）初始化为null值，来暗示垃圾收集

器来收集该对象。但此时，如果该对象连接有事件监听器（典型的 AWT组件），那它还是不可以被收集。所以在设一个引用变量为null值之前，应注意该引用变量指向的对象是否被监听，若有，要首先除去监听器，然后才可以赋空值。 9. 每一个对象都有一个finalize( )方法，这个方法是从Object类继承来的。 10. finalize( )方法用来回收内存以外的系统资源，就像是文件处理器和网络连接器。该方法的调用顺序和用来调用该方法的对象的创建顺序是无关的。换句话说，书写程序时该方法的顺序和方法的实际调用顺序是不相干的。请注意这只是finalize( )方法的特点。 11. 每个对象只能调用finalize( )方法一次。如果在finalize( )方法执行时产生异常（exception），则该对象仍可以被垃圾收集器收集。 12. 垃圾收集器跟踪每一个对象，收集那些不可到达的对象（即该对象没有被程序的任何"活的部分"所调用），回收其占有的内存空间。但在进行垃圾收集的时候，垃圾收集器会调用finalize( )方法，通过让其他对象知道它的存在，而使不可到达的对象再次"复苏"为可到达的对象。既然每个对象只能调用一次finalize( )方法，所以每个对象也只可能"复苏"一次。 13. finalize( )方法可以明确地被调用，但它却不能进行垃圾收集。 14. finalize( )方法可以被重载（overload），但只有具备初始的finalize( )方法特点的方法才可以被垃圾收集器调用。 15. 子类的finalize( )方法可以明确地调用父类的finalize( )方法，作为该子类对象的最后一次适当的操作。但Java编译器却不认为这是一次覆盖操作（overriding），所以也不会对其调用进行检查。 16. 当finalize( )方法尚未被调用时，System.runFinalization( )方法可以用来调用finalize( )方法，并实现相同

的效果，对无用对象进行垃圾收集。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com