Java实时应用程序中的内存管理 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/144/2021_2022_Java_E5_AE_9E_E6_97_B6_c104_144461.htm 使用Java的一个主要优点就是无需担心废弃对象，即，让Java运行时负责Java对象的内存管理。 这是通过让Java运行时对不再使用的Java对象进行垃圾收集而实现的。 垃圾收集是一个比较复杂的过程。通常，Java运行时会遍历堆，检查不再被其他对象引用、从而可以安全删除的对象，然而，由于垃圾收集占用CPU周期，所以它可能会影响应用程序代码的执行。即，如果在执行应用程序代码的过程中执行垃圾收集，则应用程序代码的响应时间可能延长。这会导致用户事务延迟的延长。更为糟糕的是，因为用户不知道何时会进行垃圾收集，因此延迟的延长是不可预知的。 实时应用程序有严格的时间要求，即，它们必须在确定的、已知的延迟条件下执行应用程序代码。因此垃圾收集所引起的不可预知的延迟延长就成为一个问题。 那么这个问题的解决方案是什么呢？一个明显的解决方案就是不要对实时应用程序使用Java.这是一个下下选。作为一种编程语言和一个开发平台，Java具有许多优点。我们应该能够解决Java中的这个问题。 另一种解决方案是在Java中使用另一种内存管理方法来代替垃圾收集程序。RTSJ （Real-Time Specification for Java）定义了immortal memory和scoped memory的概念。不朽内存（Immortal memory）是从不进行垃圾收集的内存，只要JVM存在，它就也存在。作用域内存（Scoped memory）则是按块分配和释放的内存。即，用户显式地创建一个内存区域来存放对象，该区域中的对象将在该区域被销毁时释放。

不管是不朽内存还是作用域内存，都不需要进行垃圾收集。不过，这也有一个缺点：管理内存的重担又落在了用户身上，就像C/C 应用程序一样。这个代价仍然很高。有没有更好的方法呢？ 我们来重新考虑一下垃圾收集。垃圾收集的主要问题是它所引起的不可预知的延迟峰值。能否避免这种不可预知的行为呢？或者，能否限制（即约束）这种不可预知的行为呢？通过更频繁地执行垃圾收集，我们就可以限制最大延迟时间。这就是WLRT所采用的方法。因此，垃圾收集成为一项可预知的任务，具有已知的代价，从而使实时开发人员可以根据需要进行考虑和建模。而且，更为重要的是，我们没有牺牲Java的易用性。 注：应该注意的是，虽然有垃圾收集和其他的内存管理技术，Java程序仍然可能存在内存泄漏。例如，在下述情况下：在不需要组件对象时，却没有将其从容器中移除。 