使用实时Java降低Java应用程序的易变性(1)Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022__E4_BD_BF_E7_94_A8_E5_AE_9E_E6_c104_644692.htm 一些Java应用程序未能提供适当的服务质量，尽管实现了其他性能目标，比如平均延迟或总吞吐量。通过引入不受应用程序控制的暂停或中断机制，Java 语言和运行时系统有时可能使应用程序无法满足服务性能指标。本文解释JVM中的延迟和中断的根源，介绍可用于减轻这些问题的技术，使您的应用程序能够交付更加一致的服务质量。 Java 应用程序中的易变性（通常是由暂停或延迟导致的，其发生时间无法预测）可能在整个软件栈中发生。延迟可由以下因素引起： ?硬件（缓存期间） ?固件（处理 CPU 温度数据等系统管理中断的过程中） ?操作系统（响应一个中断或执行定期调度的后台活动） ?在相同系统上运行的其他程序 ?JVM（垃圾收集、即时编译和类加载） ?Java 应用程序本身 很难在较高级别上补偿较低级别上的延迟，所以，如果您试图仅在应用程序级别解决易变性，您可能只是转移了 JVM 或 OS 延迟，并没有解决实际问题。幸运的是，较低级别的延迟可能比较高级别上的延迟相对短一些，所以只有在降低易变性的需求非常强烈时，才需要深入到比 JVM 或 OS 更低的级别上。如果需求不是那么强烈，您可以将精力集中在 JVM 级别上或应用程序中。 实时 Java 提供了必要的工具来堵截 JVM 和应用程序中的易变性源头，交付用户要求的服务质量。本文详细介绍 JVM 和应用程序级别上的易变性源头，介绍可用于减轻其影响的工具和技术。然后介绍一个简单的 Java 服务器应用程序来演示其中一些概念。 解决易变

性源头 JVM 中的易变性主要源自于 Java 语言的动态特性：？内存绝不会被应用程序显式释放，而是被垃圾收集器定期回收。？类在被应用程序首次使用时才进行解析。？本机代码在应用程序运行时由即时（JIT）编译器编译（而且可以重新编译），基于经常调用的类和方法。 在 Java 应用程序级别上，线程管理是与易变性相关的关键区域。 垃圾收集暂停 当垃圾收集器回收程序不再使用的内存时，它可以停止任何应用程序线程。（这种类型的收集器称为 Stop-the-world 或 STW 收集器）。或者它可以与应用程序同时执行自己的一些工作。无论是哪种情况，垃圾收集器需要的资源都不能供应用程序使用，所以，众所周知，垃圾收集（GC）是 Java 应用程序性能中的暂停和易变性的源头。尽管许多 GC 模型都具有自己的优缺点，但当应用程序的目标是缩短 GC 暂停时，两个主要的选择将是分代（generational）和实时 收集器。 分代收集器将堆组织为至少两个部分，这两个部分通常称为新和旧（有时称为保留）空间。新对象始终在新空间中分配。当新空间耗尽空闲内存时，将仅在该空间中进行垃圾收集。使用相对较小的新空间可能时 GC 周期更短。在多次新空间垃圾收集过程中存留下来的对象会被提升到旧空间中。旧空间垃圾收集发生的频率通常比新空间垃圾收集低得多，但是由于旧空间比新空间大得多，所以这些 GC 周期可能长得多。分代垃圾收集器提供了相对较短的平均 GC 暂停时间，但是旧空间收集的开销可能导致这些暂停时间的标准偏差非常大。对于活动数据集不会经常更改，但会产生大量垃圾的应用程序而言，分代收集器是最有效的。在这种场景中，旧空间收集极少发生，因此 GC 暂停时间取决于短的新空间收集时间。

与分代收集器相反，实时垃圾收集器会控制自身的行为，以显着缩短 GC 周期的长度（通过在应用程序空闲时执行周期）或减轻这些周期对应用程序性能的影响（通过基于与应用程序之间的一种“ 契约 ”，以更小的增量执行工作）。使用这类收集器，您可以预测完成特定任务的最遭情形。例如，IBM? WebSphere? Real-Time JVM 中的垃圾收集器将 GC 周期划分为较小的工作片段（称为 GC 限额），这些限额可以增量方式完成。对限额的调度对应用程序性能的影响极小，其延迟可低至几百微秒，通常小于 1 毫秒。为了达到这种延迟级别，垃圾收集器必须能够计划自己的工作，方法是引入应用程序利用契约 的概念。此契约管理允许 GC 中断应用程序执行工作的频率。例如，默认的利用契约为 70%，也就是在实时操作系统上运行时，仅允许 GC 使用每 10 毫秒中的至多 3 毫秒，典型的暂停时间大约为 500 微秒。 在实时垃圾收集器上运行应用程序时，堆大小和应用程序利用率是要考虑的重要调优选项。随着应用程序利用率的增加，垃圾收集器完成其工作的时间会更短，因此需要更大的堆来确保 GC 周期可以增量式地完成。如果垃圾收集器无法跟上分配速度，GC 将采用同步收集。 例如，与在使用分代垃圾收集器的 JVM 上（未提供利用契约）运行时相比，在 IBM WebSphere Real-Time JVM 上运行的应用程序（具有 70% 的默认应用程序利用契约）默认需要更大的堆。由于实时垃圾收集器控制着 GC 暂停时间的长度，所以增加堆大小会降低 GC 频率，不会延长各次暂停时间。另一方面，在非实时垃圾收集器中，增加堆大小通常会降低 GC 周期的频率，这会降低垃圾收集器的总体影响。当发生垃圾收集时，暂停时间通常会更长（因

为需要检查更大的堆）。 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