IBM WebSphere Real Time for Linux
V 3

用户指南

IBM WebSphere Real Time for Linux
V 3

用户指南

注意

在使用本资料及其支持的产品之前，请阅读第 73 页的『声明』中的信息。

第五版（2014 年 2 月）

本版本的用户指南适用于 IBM WebSphere Real Time for Linux V3 以及所有后续发行版和修订版，直到在新版本中另有声明为止。

目录

图

表

前言

本用户指南提供关于 IBM® WebSphere® Real Time for Linux 的常规信息。

第 1 章 简介

本文为您介绍 IBM WebSphere Real Time for Linux。

对该用户指南做出的任何新修改均通过更改左侧的竖线来指示。

本指南中未提供的有关 IBM WebSphere Real Time for Linux 的最新信息可以在以下位置中找到: http://www.ibm.com/support/docview.wss?uid=swg21501145

- 『WebSphere Real Time for Linux 概述』
- 『新增内容』
- 第 2 页的『优势』

WebSphere Real Time for Linux 概述

WebSphere Real Time for Linux 将实时功能与 IBM J9 虚拟机 (JVM) 捆绑在一起。

WebSphere Real Time for Linux 是一种 Java™ 运行时环境，且带有以实时功能扩展 IBM SDK for Java 的软件开发工具包 (SDK)。依赖于精确响应时间的应用程序可利用随 WebSphere Real Time for Linux（基于标准 Java 技术）提供的实时功能。

功能

实时应用程序需要的是稳定的运行时，而不是绝对速度。

使用传统 JVM 部署实时应用程序时的主要考虑事项如下所述:
- 由垃圾回收 (GC) 活动导致的不可预测（可能非常长）延迟。
- 出现即时 (JIT) 编译和重新编译时方法运行时出现延迟，且执行时间有变化。
- 任意的操作系统调度。

WebSphere Real Time for Linux 通过提供以下项除去这些障碍:
- Metronome 垃圾回收器，一种递增的确定性垃圾回收器，暂停时间极短。

新增内容

本主题介绍 IBM WebSphere Real Time for Linux 的更改内容。

WebSphere Real Time for Linux V3

WebSphere Real Time for Linux V3 是对 IBM SDK for Java V7 的扩展，以本发行版所提供的特性和功能为构建基础，包含实时功能。WebSphere Real Time for Linux 的先前版本均以 IBM SDK for Java 的先前发行版为基础。

要了解有关 IBM SDK for Java V7 中新增功能的更多信息，请参阅 IBM SDK for Java 7 信息中心中的新增内容。

对该用户指南做出的任何新修改均通过更改左侧的竖线来指示。

使用 Linux 调度策略来调度 Java 线程

从服务刷新 1 开始，您可以使用 **SCHED_RR** 调度策略来调度常规 Java 线程，以调优实时应用程序。有关更多信息，请参阅第 4 页的『线程调度』。

控制 Metronome 垃圾回收器的暂停时间

缺省情况下，Metronome 垃圾回收器在垃圾回收循环间暂停 3 毫秒。您可以使用新的命令行选项来更改该值以控制暂停时间。有关此选项的更多信息，请参阅第 20 页的『控制暂停时间』。

压缩引用

Metronome 垃圾回收器目前支持 64 位平台上的非压缩引用和压缩引用。有关任何性能含意，请参阅第 29 页的第 7 章，『性能』。

优势

实时环境的优点是 Java 应用程序能够以比使用标准 JVM 更高的可预测性运行，并提供一致的 Java 应用程序计时行为。会在指定的时间执行后台活动（如编译和垃圾回收），从而在运行应用程序时除去任何意外的后台活动峰值。

使用 Metronome 实时垃圾回收技术扩展 JVM 即可获得这些优势。

辅助功能选项

辅助功能选项功能部件帮助带有某些缺陷（如灵活性受限制或视力有限）的用户成功地使用信息技术。

IBM 力求为不同年龄或能力的所有人提供可访问的产品。

例如，可以在没有鼠标的情况下，通过只使用键盘来操作 WebSphere Real Time for Linux。

要读取有关影响底层 IBM SDK for Java V7 的辅助功能选项问题的信息，请参阅 IBM 信息中心。WebSphere Real Time for Linux 中没有影响独特功能部件和功能的辅助功能选项问题。

键盘导航

本产品使用标准的 Microsoft Windows 导航键。

对于需要使用键盘导航的用户，可在 Swing密钥绑定处找到 Swing 应用程序的有用击键的描述。

IBM 和辅助功能选项

请参阅 IBM Human Ability and Accessibility Center 以获取有关 IBM 对辅助功能选项的承诺的更多信息。

第 2 章 了解 IBM WebSphere Real Time for Linux

本部分介绍 IBM WebSphere Real Time for Linux 的关键组件。

- 『Metronome 垃圾回收器简介』

Metronome 垃圾回收器简介

Metronome 垃圾回收器用于取代 WebSphere Real Time for Linux 中的标准垃圾回收器。

Metronome 垃圾回收与标准垃圾回收之间的关键差异在于 Metronome 垃圾回收发生在较小的可中断步骤中，但标准垃圾回收会在标记和回收垃圾的过程中停止应用程序。

例如:

```
java -Xgcpolicy:metronome -Xgc:targetUtilization=80 yourApplication
```

该示例指定应用程序在每 60 毫秒中运行 80% 的时间。其余 20% 的时间可用于垃圾回收（如果有垃圾要回收）。如果给予 Metronome 垃圾回收器足够的资源，那么它可保证达到利用率级别。当堆中的可用空间量降低到动态确定的阈值以下时，会开始进行垃圾回收。

Metronome 垃圾回收和类卸载

Metronome 通过与标准 Java 开发者套件相同的方式来支持类卸载。但是，由于其中涉及的工作，在卸载类时，垃圾回收活动期间可能会出现暂停时间界外值。

Metronome 垃圾回收器线程

Metronome 垃圾回收器由两种类型的线程构成: 单个警报线程以及若干个回收 (GC) 线程。缺省情况下，GC 对每个可用于操作系统的逻辑活动处理器都使用一个线程。这可实现 GC 周期内的最高效并行处理。GC 周期表示触发 GC 与垃圾释放完成之间的时间。根据 Java 堆大小，一个完整 GC 周期的耗用时间可以为若干秒。一个 GC 周期通常包含数百个 GC 定量。这些定量是对应用程序编码的非常短的暂停，通常持续 3 毫秒。使用 **-verbose:gc** 可获取周期和定量的摘要报告。有关更多信息，请参阅: 第 60 页的『使用 verbose:gc 信息』。您可以使用 **-Xgcthreads** 选项来设置 JVM 的 GC 线程数。

将 **-Xgcthreads** 增大到缺省值以上并无任何益处。减小 **-Xgcthreads** 可以降低 GC 周期内的整体 CPU 负载，尽管 GC 周期将被延长。

注: GC 定量持续时间目标保持为常量 3 毫秒。

您不能更改 JVM 的警报线程数。

Metronome 垃圾回收器定期检查 JVM 以查看堆内存是否具有足够的可用空间。当可用空间量降低至限制值以下时，Metronome 垃圾回收器将促使 JVM 开始垃圾回收。

警报线程

单个警报线程可保证使用最少资源。它定期“唤醒”，并执行以下检查:

- 堆内存中的可用空间量

- 垃圾回收当前是否正在进行

  如果没有足够的可用空间，并且未在执行垃圾回收，那么警报线程将触发回收线程以开始垃圾回收。警报线程在其检查 JVM 的下一安排时间之前不会执行任何操作。

回收线程

回收线程执行垃圾回收。

垃圾回收周期完成后，Metronome 垃圾回收器将检查可用堆空间量。如果仍没有足够的可用堆空间，那么将使用同一触发器标识来启动另一个垃圾回收周期。如果有足够的可用堆空间，那么触发器将结束，而垃圾回收线程将停止。警报线程会继续监控可用堆空间，并将在需要时触发另一个垃圾回收周期。

有关使用 Metronome 垃圾回收器的更多信息，请参阅第 20 页的『使用 Metronome 垃圾回收器』。

线程调度

Linux 调度策略可以与常规 Java 线程一起使用以调优实时应用程序。

对于 WebSphere Real Time for Linux，您可以使用 SCHED_RR 调度策略运行常规 Java 线程。使用 SCHED_RR 策略可以使您更好地控制应用程序，这可改善 Java 线程的实时性能。当使用 SCHED_RR 策略启动 Java 时，JVM 会检测主线程的优先级和策略。JVM 会相应地变更优先级和策略映射。有关变更常规 Java 线程优先级和策略的更多信息，请参阅第 17 页的『线程调度和分派』。

Linux 调度策略包括:

SCHED_OTHER

大多数线程使用的缺省通用时间共享调度策略。必须为这些线程分配优先级零。

SCHED_OTHER 使用时间分片，这意味着每个线程只运行一段有限的时间，该时间过后将允许下一个线程运行。

SCHED_FIFO

只能用于大于零的优先级。当 SCHED_FIFO 线程变为可用时，该线程具有高于任何普通 SCHED_OTHER 线程的优先级。

如果具有较高优先级的 SCHED_FIFO 线程变为可用，那么该线程具有高于具有较低优先级的现有 SCHED_FIFO 线程的优先级。这样，该线程将因其优先级而被保持在队列顶部。

无时间分片。

注: WebSphere Real Time for Linux 不使用 SCHED_FIFO。

SCHED_RR

是对 SCHED_FIFO 的增强。差异在于每个线程只允许运行一段有限的时间。如果线程超过该时间，那么将按其优先级返回到列表。SCHED_RR 可以供 WebSphere Real Time for Linux V3 使用。

有关这些 Linux 调度策略的更多详细信息，请参阅 **sched_setscheduler** 的联机帮助页。

有关将 Linux 调度策略与 WebSphere Real Time for Linux 结合使用的更多信息，请参阅第 17 页的『线程调度和分派』。

第 3 章 规划

在安装 WebSphere Real Time for Linux 之前请先阅读本部分。

-
- 『受支持的环境』
-
- 第 8 页的『注意事项』

迁移

您可以在不进行修改的情况下在 WebSphere Real Time for Linux 上运行标准 Java 应用程序。

受支持的环境

IBM WebSphere Real Time for Linux 在某些硬件平台和操作系统上是受支持的。

IBM WebSphere Real Time for Linux

支持以下平台体系结构:

- Intel 体系结构, 32 位 (IA-32)
  - Pentium 4
  - Pentium Xeon
  - Pentium M
  - Pentium D 和等效处理器
- AMD64/EM64T
- IBM POWER® 32
- IBM POWER 64

注: 不再支持 Pentium 3 硬件。

支持以下操作系统:

表 1. 经过测试的 Linux 环境

硬件	IA-32 32 位	AMD64/EM64T 64 位	
SDK 地址空间	**32 位**	**32 位**	**64 位**
RHEL 5 Update 7	是	是	是
RHEL 6 Update 1	是	是	是
SLES 11 Service Pack 2	是	是	是
Ubuntu 8.04	是	是	是
Ubuntu 10.04	是	是	是

注: 不支持 SLES 9、SLES 10 和 RHEL 4。

注意事项

使用 WebSphere Real Time for Linux 时有许多因素需要注意。

- 可能的情况下，请勿在同一个系统上运行多个实时 JVM。原因是这样做会导致有多个垃圾回收器。每个 JVM 不知道其他 JVM 的内存区域。影响之一是在 JVM 之间无法协调 GC 循环和暂停时间，即意为某个 JVM 可能对其他 JVM 的 GC 性能产生不利影响。 如果您必须使用多个 JVM，请确保使用 **taskset** 命令将每个 JVM 绑定到处理器的特定子集。
- 先前 WebSphere Real Time for Linux 发行版用于存储预编译代码和类的共享高速缓存与本发行版的 WebSphere Real Time for Linux 使用的高速缓存不兼容。您必须重新生成先前高速缓存的内容。
- 在使用共享类高速缓存时，高速缓存名称不能超出 53 个字符。

第 4 章 安装 WebSphere Real Time for Linux

请按照以下步骤来安装产品。

- 『安装文件』
- 『从 InstallAnywhere 程序包进行安装』
  - 第 10 页的『完成有人照管安装』
  - 第 11 页的『完成无人照管安装』
  - 第 12 页的『已知问题和限制』
- 第 13 页的『设置路径』
- 第 14 页的『设置类路径』
- 第 14 页的『测试安装』
- 第 15 页的『卸载 WebSphere Real Time for Linux』

安装文件

您需要这些安装文件。

IBM WebSphere Real Time for Linux 通过两种类型的 InstallAnywhere 程序包提供。

可安装的程序包

可安装程序包用于配置系统。例如，程序可能会设置环境变量。

- `wrt-3.0-0.0-linux-<arch>-sdk.bin`
- `wrt-3.0-0.0-linux-<arch>-jre.bin`

归档程序包

这些程序包将文件解压缩到系统，但不会执行任何配置。

- `wrt-3.0-0.0-linux-<arch>-sdk-archive.bin`
- `wrt-3.0-0.0-linux-<arch>-jre-archive.bin`

注: <arch> 是平台体系结构: x86_32 或 x86_64。

从 InstallAnywhere 程序包进行安装

这些程序包提供了一个交互式程序，它指导您完成安装选项。您可以通过图形用户界面或系统控制台运行该程序。

开始之前

您的系统必须具有以下两个共享库:

- GNU C 库 V2.3 (glibc)
- `libstdc++.so.5`

如果您没有 `libstdc++.so.5` 共享库，那么在安装时可能会看到 Java 核心转储，并包含以下错误:

```
JVMJ9VM011W Unable to load j9dmp24: libstdc++.so.5: cannot open shared object file:
No such file or directory
JVMJ9VM011W Unable to load j9gc24: libstdc++.so.5: cannot open shared object file:
No such file or directory
JVMJ9VM011W Unable to load j9vrb24: libstdc++.so.5: cannot open shared object file:
No such file or directory
```

如果您在安装可安装的程序包，那么必须在系统上安装 rpm 构建工具，否则安装程序无法在 RPM 数据库中注册此新程序包。要确定是否已安装 rpm 构建工具，请输入以下命令:

```
rpm -q rpm-build
```

关于此任务

InstallAnywhere 程序包具有 .bin 文件扩展名。

有两种类型的程序包:

可安装程序包
安装这些程序包也会配置您的系统，例如，通过设置环境变量进行配置。

归档 安装这些程序包会将文件解抽取到系统，但不会执行任何配置。

过程

- 要以交互方式安装程序包，请完成有人照管安装。
- 要在无任何额外用户交互的情况下安装程序包，请完成无人照管安装。 如果要安装许多系统，那么可以选择该选项。
- 安装过程完成后，按照本部分中的配置步骤进行操作，例如设置路径和类路径环境变量。

结果

产品已安装。

注: 请勿中断安装过程，例如通过按 Ctrl+C 来中断。如果中断该过程，那么可能必须重新安装产品。有关更多信息，请参阅第 12 页的『中断的安装』。

如果您在使用可安装程序包，那么可能会看到一些消息，告知您发现了问题。安装归档程序包不会产生任何消息。以下列表中显示了安装可安装程序包时可能会看到的一些消息:

The installer cannot run on your configuration. It will now quit.
如果未授权您的用户标识运行安装过程，那么会出现该错误消息。安装过程因为无法继续，所以会结束。要解决该问题，请使用具有 root 用户权限的用户标识来重新开始安装。

An RPM package is already installed. Uninstall the package before proceeding. 该消息指示 RPM 程序包已安装。安装过程因为无法继续，所以会结束。要解决该问题，请先卸载 RPM 程序包，然后再继续。

完成有人照管安装

以交互方式从 InstallAnywhere 程序包安装产品。

开始之前

在开始安装过程之前，请先确保满足以下条件:

- 如果之前已从 RPM 程序包安装了 WebSphere Real Time for Linux，那么必须先卸载该程序包，然后再继续。
- 您必须拥有一个具备 root 用户权限的用户标识。

过程

1. 将安装包文件下载到临时目录。
2. 切换到临时目录。
3. 通过在 shell 提示符处输入 *./package*（其中 *package* 是要安装的程序包的名称）来开始安装过程。
4. 从安装程序窗口中显示的列表中选择语言，然后单击**下一步**。可用语言的列表取决于系统的语言环境设置。
5. 阅读许可协议，并使用滚动条到达许可文本末尾。要继续安装，必须接受许可协议的条款。要接受这些条款，请选择相应单选按钮，然后单击**确定**。

   注: 您必须首先阅读到许可文本末尾，然后才能选择用于接受许可协议的单选按钮。
6. 将要求您选择安装的目标目录。如果您不希望安装到缺省目录，请单击**选择**以通过使用浏览器窗口来选择其他目录。选择了安装目录后，请单击**下一步**以继续。
7. 将要求您复查所做选择。要更改所做的选择，请单击**上一步**。如果选择正确，请单击**安装**以继续安装。
8. 安装过程完成后，请单击**完成**以完成安装。

完成无人照管安装

如果您要安装多个系统，并且已知道要使用的安装选项，那么可能希望使用无人照管安装过程。您通过使用有人照管安装过程安装一次，然后使用生成的响应文件来完成以后的安装而无需任何额外的用户交互。

过程

1. 通过完成有人照管安装来创建响应文件。使用以下选项之一:
   - 使用 GUI 并指定安装程序创建响应文件。该响应文件名为 `installer.properties`，并创建于安装目录中。
   - 使用命令行并向有人照管安装命令附加 -r 选项，同时指定响应文件的完整路径。例如:

     ```
     ./package -r /path/installer.properties
     ```

   示例响应文件内容:

   ```
   INSTALLER_UI=silent
   USER_INSTALL_DIR=/my_directory
   ```

   在该示例中，*/my_directory* 是您为安装而选择的目标安装目录。
2. 可选: 如有需要，可编辑响应文件以更改选项。

   注: 归档程序包具有以下已知问题: 即使您在响应文件中更改了目录，使用响应文件的安装仍会使用缺省目录。如果缺省目录中存在先前安装，那么会予以覆盖。

如果您要创建多个响应文件，并且每个都具有不同的安装选项，请为每个响应文件都指定格式为 *myfile*.properties 的唯一名称。

3. 可选： 生成日志文件。 因为要进行静默安装，所以安装过程结束时将不会显示任何状态消息。要生成包含安装状态的日志文件，请完成以下步骤:
   a. 通过使用以下命令来设置所需的系统属性。

      ```
      export _JAVA_OPTIONS="-Dlax.debug.level=3 -Dlax.debug.all=true"
      ```

   b. 设置以下环境变量以将日志输出发送到控制台。

      ```
      export LAX_DEBUG=1
      ```

4. 通过 **-i** 静默选项以及用于指定响应文件的 **-f** 选项运行程序包安装程序，以开始无人照管安装。 例如:

   ```
   ./package -i silent -f /path/installer.properties 1>console.txt 2>&1
   ./package -i silent -f /path/myfile.properties 1>console.txt 2>&1
   ```

   您可以使用属性文件的标准路径或相对路径。在这些示例中，字符串 1>console.txt 2>&1 将安装过程信息从 stderr 和 stdout 流重定向到当前目录中的 console.txt 日志文件。如果您认为安装存在问题，请复查该日志文件。

   注: 如果您的安装目录包含多个响应文件，那么将使用缺省响应文件 installer.properties。

中断的安装

如果在安装期间程序包安装程序意外停止（例如，如果您按了 Ctrl+C），那么安装会中断并且您无法卸载或重新安装产品。如果您尝试进行卸载或重新安装，那么可能会看到消息 Fatal Application Error。

关于此任务

要解决该问题，请删除文件并重新安装，如以下步骤中所述。

过程

1. 删除 /var/.com.zerog.registry.xml 注册表文件。
2. 如果创建了包含所安装内容的目录，请将其删除。例如 /opt/IBM/javawrt3[_64]/。
3. 重新运行安装程序。

已知问题和限制

InstallAnywhere 程序包具有一些已知问题和限制。

- 如果您的系统中没有 libstdc++.so.5 共享库，那么安装将失败，并产生 Java 核心转储。有关更多信息，请参阅第 9 页的『从 InstallAnywhere 程序包进行安装』。
- 安装包 GUI 不支持 Orca 屏幕朗读程序。您可以使用无人照管安装方式作为 GUI 的替代方式。
- 如果安装后，您输入 *./package* 以再次启动安装程序，那么安装程序将显示以下消息:

  ```
  ENTER THE NUMBER OF THE DESIRED CHOICE, OR PRESS <ENTER> TO ACCEPT THE DEFAULT:
  ```

  如果您按 Enter 键以接受缺省值，那么安装程序将没有响应。请输入一个数字，然后按 Enter 键。

- 如果您安装程序包，然后尝试以其他方式（例如控制台或静默方式）重新安装，那么可能会看到以下错误消息:

  ```
  Invocation of this Java Application has caused an InvocationTargetException.
  This application will now exit
  ```

  如果您已通过使用 GUI 方式安装并要以控制台方式重新运行安装程序，那么不应该看到此消息。如果您看到该错误，并在运行程序以选择卸载选项（仅限可安装程序包），请改用 **./_uninstall/uninstall** 命令，如第 15 页的『卸载 WebSphere Real Time for Linux』中所述。

仅限可安装程序包

- 您无法通过使用 InstallAnywhere 程序包来升级现有安装。要升级 WebSphere Real Time for Linux，必须首先卸载任何先前版本。
- 即使您使用不同安装目录，也不能同时在同一系统上安装同一版本的 WebSphere Real Time for Linux 的两个不同实例。例如，您不能同时在 `/previous` 目录中有 WebSphere Real Time for Linux V3 并在 `/current` 目录中有 WebSphere Real Time for Linux 服务刷新安装。安装程序会检查版本号。如果安装程序发现具有同一版本号的现有程序包，那么会要求您卸载该现有程序包。
- 如果程序包已安装，并且您通过使用 GUI 再次运行程序包安装程序，那么可以选择卸载该程序包。该卸载选项在无人照管方式下不可用。如果您以无人照管方式再次运行程序包安装程序，那么该程序会运行但不会执行任何操作。

仅限归档程序包

- 如果您更改响应文件中的安装目录，然后通过使用该响应文件来运行无人照管安装，那么安装程序将忽略该新安装目录并改用缺省目录。如果缺省目录中存在先前安装，那么会予以覆盖。

设置路径

设置 **PATH** 环境变量后，您可通过在 shell 提示符中输入名称来运行应用程序或程序。

关于此任务

注: 如果按本部分所述更改 **PATH** 环境变量，将覆盖路径中任何现有的 Java 可执行文件。

每次在工具名称前输入路径即可指定到该工具的路径。例如，如果 SDK 安装在 `/opt/IBM/javawrt3[_64]/` 中，您可以在 shell 提示符中输入以下命令来编译名为 *myfile.java* 的文件:

```
/opt/IBM/javawrt3[_64]/bin/javac myfile.java
```

要避免每次均需输入完整路径:

1. 在主目录（根据 shell，通常是 **.bashrc**）中编辑 shell 启动文件，并将绝对路径添加到 **PATH** 环境变量；例如:

   ```
   export PATH=/opt/IBM/javawrt3[_64]/bin:/opt/IBM/javawrt3[_64]/jre/bin:$PATH
   ```

2. 再次登录或运行更新后的 shell 脚本以激活新的 **PATH** 设置。
3. 使用 **javac** 工具来编译文件。例如，要编译文件 *myfile.java*，在 shell 提示符处输入:

```
javac -Xgcpolicy:metronome myfile.java
```

PATH 环境变量使 Linux 能够从当前目录中找到可执行文件，如 **javac**、**java** 和 **javadoc** 工具。要显示路径的当前值，请在命令提示符中输入以下命令:

```
echo $PATH
```

下一步做什么

请参阅『设置类路径』以确定是否需要设置 CLASSPATH 环境变量。

设置类路径

CLASSPATH 环境变量会告知 SDK 工具（例如 **java**、**javac** 和 **javadoc** 工具）哪里可以找到 Java 类库。

关于此任务

仅当以下条件之一适用时，才可显式设置 **CLASSPATH** 环境变量:

- 您需要使用一个不同的库或类文件（例如您自行开发的），而它又不在当前目录中。
- 您更改了 bin 和 lib 目录的位置并且它们不再具有相同的父目录。
- 您计划开发或运行在同一系统上使用不同运行时环境的应用程序。

要显示 **CLASSPATH** 的当前值，请在 shell 提示符处输入以下命令:

```
echo $CLASSPATH
```

如果开发并运行使用不同运行时环境的应用程序（包括已经单独安装的其他版本），那么必须为每个应用程序显式地设置 **CLASSPATH** 和 **PATH**。如果您同时运行多个应用程序并使用不同运行时环境，那么每个应用程序都必须在它自己的 shell 中运行。

如果一次只运行一个 Java 版本，那么可以使用 shell 脚本在不同的运行时环境间进行切换。

下一步做什么

请参阅『测试安装』以验证是否成功安装。

测试安装

如果安装成功，请使用 **-version** 选项进行检查。

关于此任务

Java 安装包括实时 JVM。

过程

通过完成以下步骤来测试您的安装:

1. 要查看实时 JVM 的版本信息，请在 shell 提示符处输入以下命令:

   ```
   java -Xgcpolicy:metronome -version
   ```

如果该命令成功，将返回以下消息:

```
java version "1.7.0"
WebSphere Real Time V3(build pxi3270-20110428_04)
IBM J9 VM (build 2.6, JRE 1.7.0 Linux x86-32 20110427_81014 (JIT enabled, AOT enabled)
J9VM - R26_head_20110426_2022_B81001
JIT  - r11_20110426_19388
GC   - R26_head_20110426_1548_B80973
J9CL - 20110427_81014)
JCL - 20110427_03 based on Oracle 7b145
```

注: 版本信息正确，但平台体系结构和日期可能与该示例不同。日期字符串格式为: yyyymmdd，后面可能跟有特定于组件的其他信息。

卸载 WebSphere Real Time for Linux

用于移除 WebSphere Real Time for Linux 的过程取决于使用的安装类型。

开始之前

对于 InstallAnywhere 可安装程序包，您的用户标识必须具有 root 用户权限。

关于此任务

没有针对 InstallAnywhere 归档程序包的卸载过程。要移除系统中的归档程序包，请删除安装程序包时选择的目标目录。对于 InstallAnywhere 可安装程序包，使用命令或再次运行安装程序来卸载产品，如以下步骤所述。

过程

- 可选: 使用 **uninstall** 命令来手动卸载。
  1. 切换到包含安装的目录。 例如:

     ```
     cd /opt/IBM/javawrt3
     ```

  2. 输入以下命令来启动卸载过程: `./_uninstall/uninstall`
- 可选: 如果卸载程序不好找，作为替代方法，您可以运行其他有人照看的安装。安装程序会检测产品是否已安装，然后给您选择是否卸载先前安装的机会。

第 5 章 运行 IBM WebSphere Real Time for Linux 应用程序

可在运行实时应用程序时帮助您的重要信息。

- 『线程调度和分派』
-
- 第 20 页的『使用 Metronome 垃圾回收器』

线程调度和分派

Linux 操作系统支持各种调度策略。缺省通用时间共享策略是 SCHED_OTHER（供大多数线程使用）。SCHED_RR 和 SCHED_FIFO 可供实时应用程序中的线程使用。仅 SCHED_OTHER 和 SCHED_RR 供 WebSphere Real Time for Linux 使用。

内核决定哪一个是处理器要运行的下一个可运行线程。内核维护可运行线程列表。它会查找具有最高优先级的线程并选择该线程作为下一个要运行的线程。

可使用以下命令列出线程优先级和策略:

```
ps -emo pid,ppid,policy,tid,comm,rtprio,cputime
```

其中策略:

- TS 是 SCHED_OTHER
- RR 是 SCHED_RR
- FF 是 SCHED_FIFO
- - 未报告任何策略

输出与以下示例相似:

```
  PID  PPID POL   TID COMMAND         RTPRIO     TIME
31531 30800 -       - java                 - 00:00:13
    -     - RR  31531 -                    6 00:00:00
    -     - RR  31532 -                    6 00:00:13
    -     - RR  31533 -                    6 00:00:00
    -     - RR  31538 -                    6 00:00:00
    -     - RR  31539 -                    6 00:00:00
    -     - RR  31540 -                    6 00:00:00
    -     - RR  31541 -                    6 00:00:00
    -     - RR  31542 -                    6 00:00:00
    -     - RR  31543 -                    6 00:00:00
    -     - RR  31544 -                    6 00:00:00
    -     - RR  31545 -                    6 00:00:00
    -     - RR  31546 -                    6 00:00:00
```

该输出显示 Java 进程以及策略为 SCHED_RR 且优先级为 6 的大量线程。

要查询当前调度策略，请使用 **sched_getscheduler** 或该示例中显示的 **ps** 命令。

有关进程的更多信息，请参阅第 34 页的『常用调试方法』。

常规 Java 线程优先级和策略

常规 Java 线程（即作为 `java.lang.Thread` 对象分配的线程）使用缺省调度策略 SCHED_OTHER。从 WebSphere Real Time for Linux V3 服务刷新 1 开始，您可以使用 SCHED_RR 调度策略运行常规 Java 线程。

缺省情况下，Java 线程使用缺省的 SCHED_OTHER 策略运行。该策略将 Java 线程映射到操作系统优先级 0。

使用 SCHED_RR 策略可以使您更好地控制应用程序，这可改善 Java 线程的实时性能。当使用 SCHED_RR 策略启动 Java 时，JVM 会检测主线程的优先级和策略。JVM 会相应地变更优先级和策略映射。所有 Java 线程以与主线程相同的操作系统优先级运行。虽然 SCHED_RR 可分配优先级 1 - 99，但是可用于 WebSphere Real Time for Linux V3 的 SCHED_RR 优先级为 1 - 10。如果将优先级设置为高于 10，那么主线程的优先级降至 10，并基于值 10 应用映射。

在命令行中更改进程的实时调度属性方法之一是使用命令 **chrt**。在以下示例中，主 Java 线程的优先级设为使用 SCHED_RR 调度策略，且操作系统优先级为 6。

```
chrt -r 6 java
```

可能需要对系统进行配置以允许更改优先级。请参阅 第 19 页的『配置系统以允许优先级更改』以获取更多信息。

表 2. Java 和操作系统优先级

Java 优先级	线程的 Java 优先级值	操作系统优先级
1	MIN_PRIORITY	6
2		6
3		6
4		6
5	NORM_PRIORITY（缺省值）	6
6		6
7		6
8		6
9		6
10	MAX_PRIORITY	6

所有与主 Java 线程关联的线程均以相同的操作系统优先级运行。

如果您运行命令 `chrt -r 11 java`，那么结果与运行 `chrt -r 10 java` 的结果相同。这是由于您无法将 10 以上的优先级应用于 JVM 线程使用的优先级映射，但是启动 JVM 并等待 JVM 终止的线程将始终保持优先级 11。

由于 SCHED_FIFO 在 WebSphere Real Time for Linux V3 中不受支持，因此如果您尝试使用命令 `chrt -f 6 java`，那么 JVM 会生成一条错误消息。

有关 **chrt** 命令的更多信息，请参阅 http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=/liaai/realtime/liaairtchrt.htm。

配置系统以允许优先级更改

缺省情况下，Linux 上的非 root 用户无法提高线程或进程的优先级。您可以更改系统配置以允许使用 Linux 的可插入认证模块 (PAM) 的 pam_limits 模块更改优先级。

如果您不能使用 **chrt** 实用程序更改线程或进程的优先级，那么通常会看到以下消息:

```
sched_setscheduler: Operation not permitted
```

在最新的 Linux 内核上，您可以更改系统配置以允许使用 pam_limits 模块更改优先级。该模块允许您在限制配置文件中配置对系统资源的限制。缺省文件是 /etc/security/limits.conf。

/etc/security/limits.conf 文件中的条目具有以下格式:

```
<domain> <type> <item> <value>
```

其中:

<domain> 可以是以下任意一项:

- 系统上可以变更资源限制的用户名。
- 其成员可以变更资源限制的组名，语法为 @group。
- 通配符“*”，用于指示任何用户或组都可以变更资源限制。

<type> 可以是以下任意一项:

- hard，其中硬限制由内核强制实施。
- soft，其中软限制适用，这些限制可在硬限制指定的范围内予以变更。
- 破折号“-”，用于指示硬限制和软限制。

<item> 是:

- 资源。将 rtprio 用于实时优先级。

<value> 是:

- 限制。使用范围 1 - 100 内的值以指示实时优先级设置的最大限制。

例如,

```
* - rtprio 100
```

允许所有用户使用 **chrt** 或其他机制更改实时进程的优先级。

缺省情况下，root 用户可以无限制地提高实时优先级。要对 root 用户应用限制，必须显式指定 root 用户。配置文件中的组和通配符限制不适用于 root 用户。

如果在该文件中指定单独用户限制，那么这些限制的优先级高于组限制。

对 limits.conf 的更改不会立即生效。您必须重新启动受影响的服务或重新引导系统才能使配置更改生效。

提高 Java 虚拟机 (JVM) 的实时优先级的功能在 Linux 内核 2.6.12 和更低版本中不可用。下表指示了在一些常见的 Linux 分发版本中是否提供了对该功能的支持。

表 3. 实时优先级更改支持

Linux 分发版本	Linux 内核版本	实时优先级更改支持（是/否）
Red Hat Enterprise Linux (RHEL) 4	2.6.9	否
RHEL 5 和更高版本	2.6.18 和更高版本	是
SUSE Linux Enterprise Server (SLES) 9	2.6.5-7	否
SLES 10 和更高版本	2.6.16 和更高版本	是
Red Hat Enterprise MRG - 所有版本	2.6.24 和更高版本	是
SUSE Linux Enterprise Real Time (SLERT) - 所有版本	2.6.16 和更高版本	是
Ubuntu 5.10	2.6.12	否
Ubuntu 6.06 和更高版本	2.6.15 和更高版本	是

要在实时 Linux 系统上启用优先级更改，您可以向 `realtime` 组添加用户，如 `limits.conf` 文件中所示。

启动辅助进程

Java 虚拟机 (JVM) API 中的 java.lang.Runtime.exec 方法为您的 Java 应用程序提供在单独进程中执行命令的能力。

通过该方法调用，会创建新的 java.lang.Process 对象。该对象可以用于控制新进程，或获取与其相关的信息。

为达到此目的，exec 方法会创建若干线程。在 IBM WebSphere Real Time for Linux 中，过程修改在实时环境中支持更具确定性的行为。

Runtime.exec 调用为每个派生子进程都创建一个“收获器”线程。收获器线程是唯一等待子进程终止的线程。当子进程终止时，收获器线程会记录子进程退出状态。收获器线程会衍生出新的进程，并赋予该进程与最初调用 Runtime.exec 的线程相同的优先级。

如果衍生的进程是另一个 WebSphere Real Time for Linux JVM，并且 Runtime.exec 方法由另一个通过 Linux 实时策略和优先级运行的方法进行了调用，那么新虚拟机的主线程会将其策略和优先级映射到同一 Linux 实时策略和优先级。该 Java 线程优先级介于 1 到 10 之间。

收获器线程还会创建两个侦听新进程的 `stdout` 和 `stderr` 流的新线程。`stdout` 和 `stderr` 数据会保存到这两个线程所使用的缓冲区中。这些缓冲区的保留时间会超过所衍生的进程的生命周期。这种持久性确保衍生进程所持有的资源在该进程终止时立即被清除。

使用 Metronome 垃圾回收器

Metronome 垃圾回收器用于取代 WebSphere Real Time for Linux 中的标准垃圾回收器。

控制暂停时间

Metronome 垃圾回收器 (GC) 暂停时间可以针对每个 Java 进程进行微调。

缺省情况下，Metronome GC 会在每次单独暂停中暂停 3 毫秒，这称为定量。完整的垃圾回收周期需要许多次这种暂停，这些暂停进行了分布以给予应用程序足够时间来运行。您可以使用 **-Xgc:targetPauseTime** 选项来更改此最长单次暂停时间值。例如，使用 **-Xgc:targetPauseTime=20** 来运行会使 GC 以不长于 20 毫秒的单独暂停来运行。

IBM Monitoring and Diagnostics Tools for Java - Garbage Collection and Memory Visualizer (GCMV) 可用于监控应用程序的 GC 暂停时间，以及帮助对 Java 应用程序中的性能问题进行诊断和调整。该工具解析和绘制各种类型日志中的数据，包括:

- 详细垃圾回收日志。
- 跟踪垃圾回收日志，通过使用 **-Xtgc** 参数生成。
- 本机内存日志，通过使用 **ps**、**svmon** 或 **perfmon** 系统命令生成。

该部分中的图形由 GCMV 生成，并显示对垃圾回收周期更改目标暂停时间的影响。每个图形都绘制 Metronome 垃圾回收周期（y 轴）和应用程序运行时间（X 轴）之间的实际暂停时间。

注: GCMV 支持较旧版本的详细垃圾回收格式。如果您要使用 GCMV 分析详细 GC 输出，请通过 **-Xgc:verboseFormat=deprecated** 选项生成输出。有关更多信息，请参阅 GC command-line options。

如果设置了缺省目标暂停时间，那么详细 GC 暂停时间图形将显示暂停时间保持在 3 毫秒标记周围或之下:

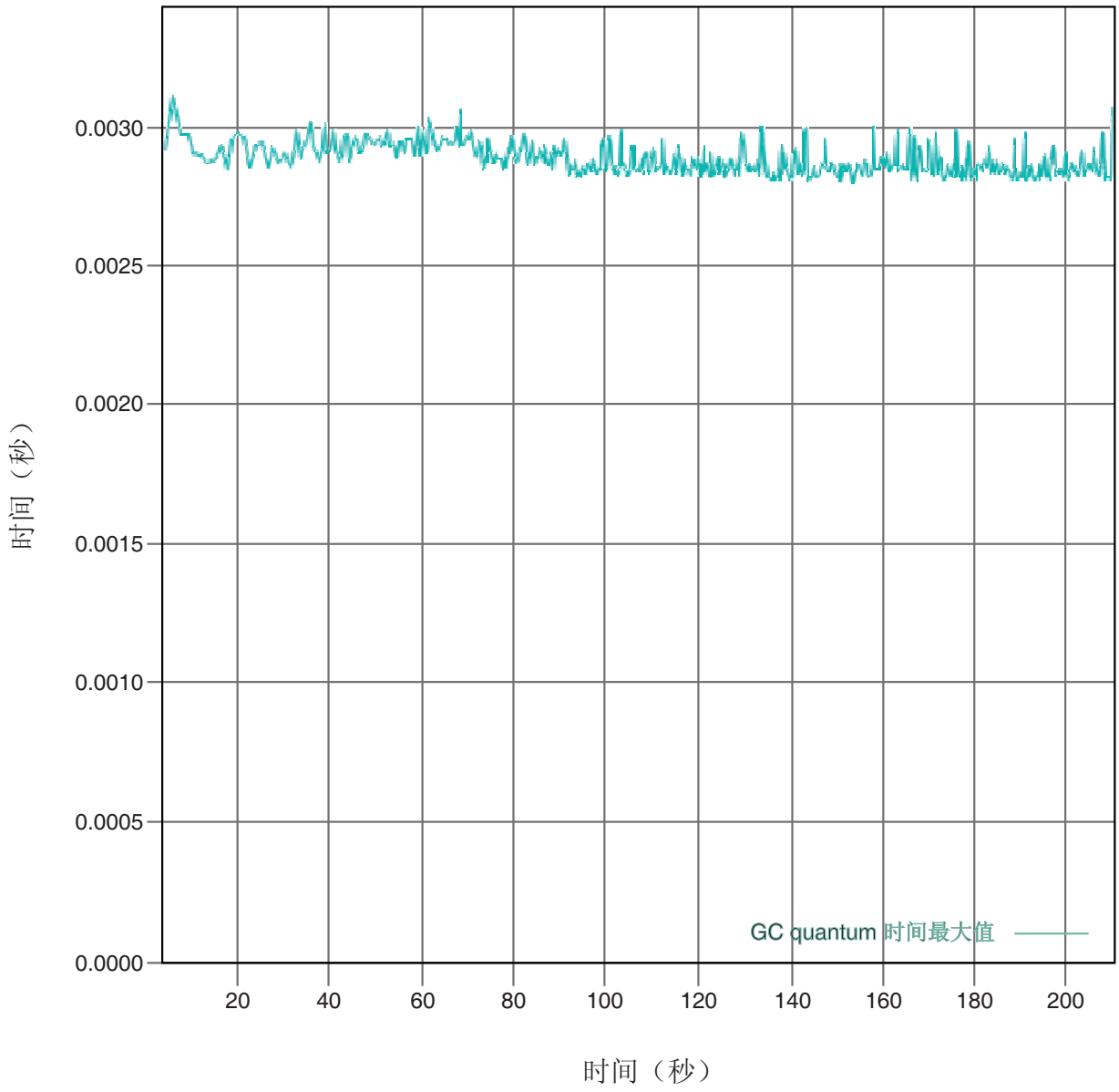

图 1. 目标暂停时间设置为缺省值（3 毫秒）时的实际垃圾回收暂停时间

如果将目标暂停时间设置为 6 毫秒，那么详细 GC 暂停时间图形将显示暂停时间保持在 6 毫秒标记周围或之下:

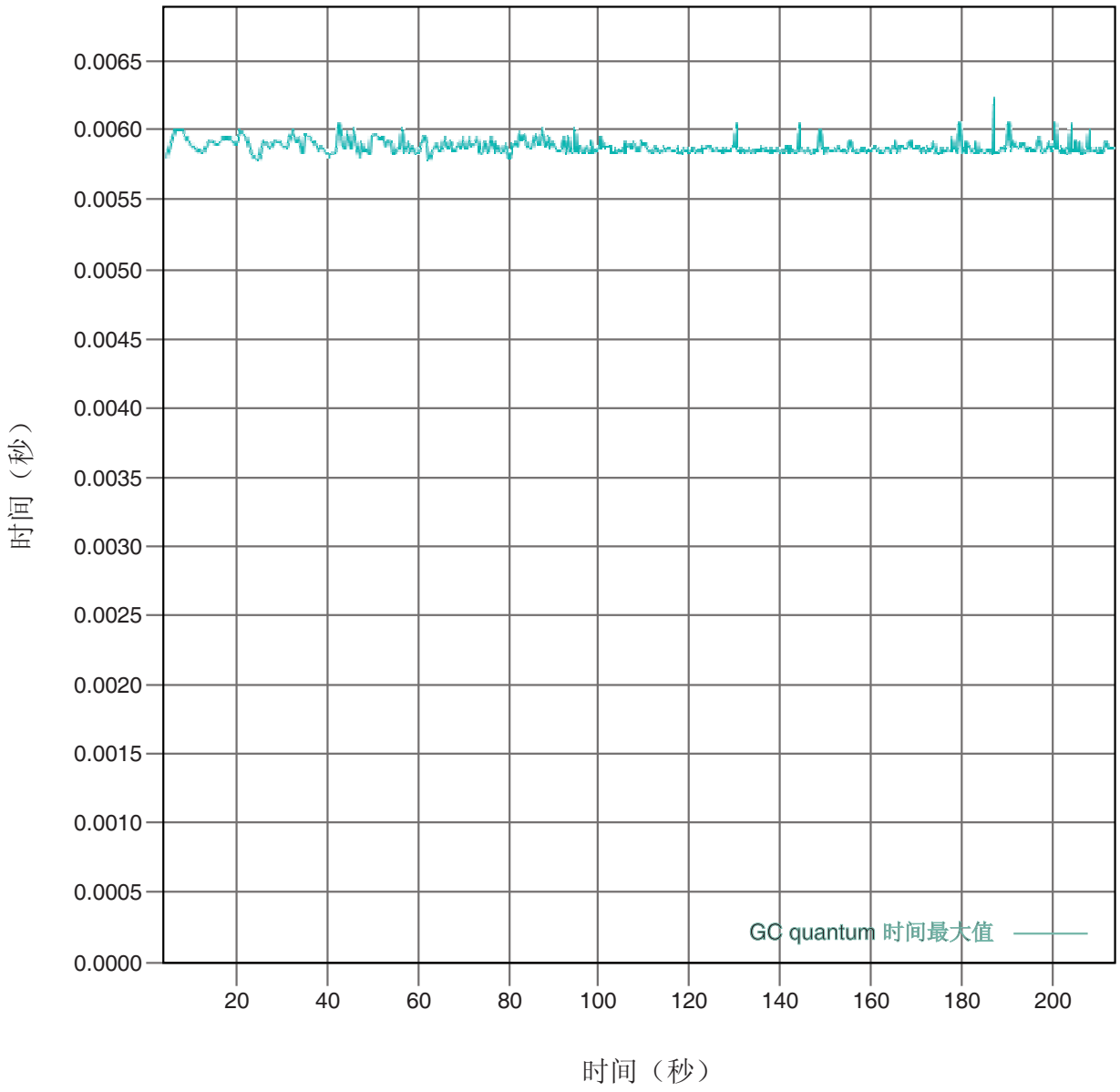

图 2. 目标暂停时间设置为 6 毫秒时的实际暂停时间

如果将目标暂停时间设置为 10 毫秒，那么详细 GC 暂停时间图形将显示暂停时间保持在 10 毫秒标记周围或之下:

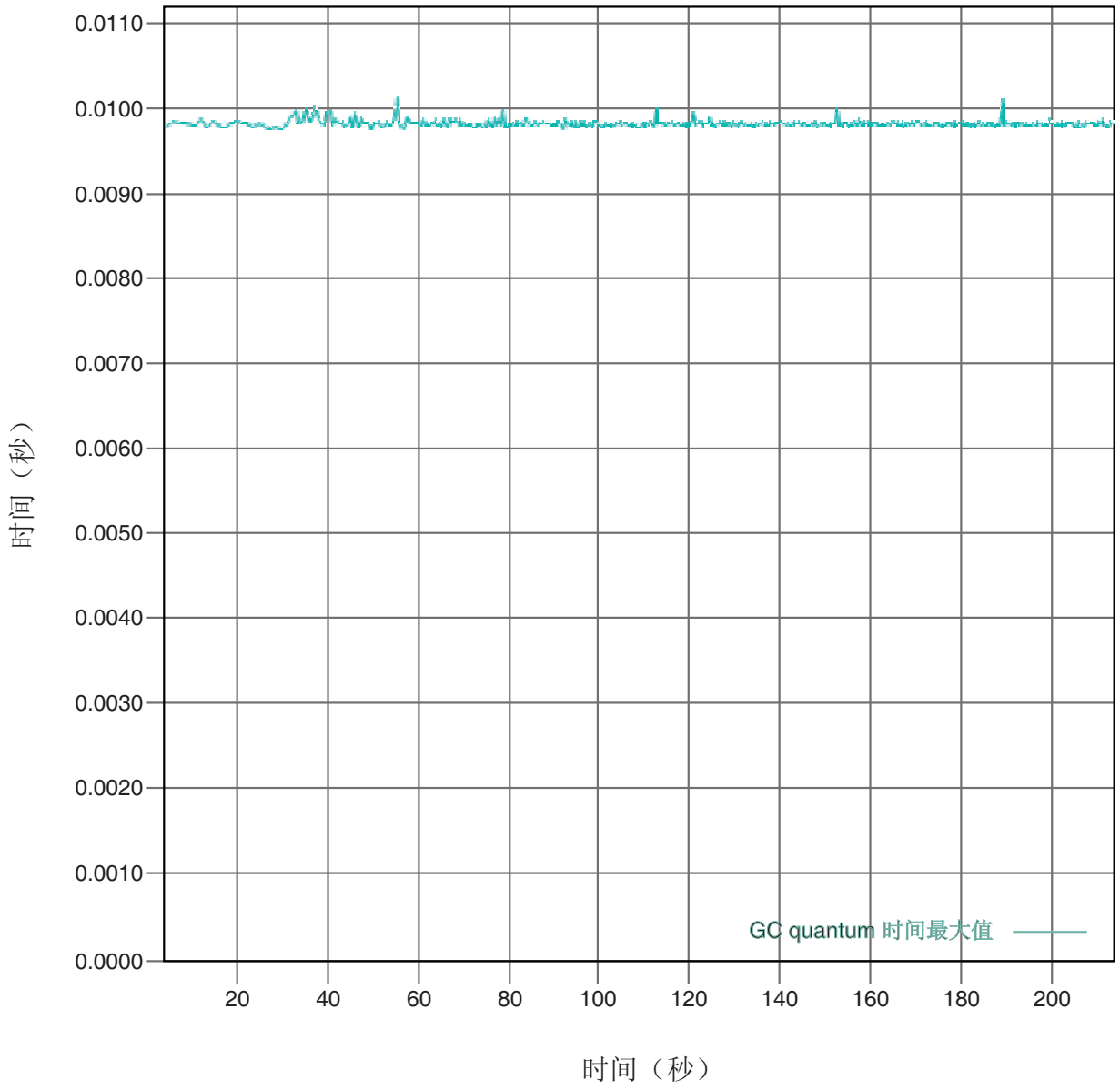

图 3. 目标暂停时间设置为 10 毫秒时的实际暂停时间

如果将目标暂停时间设置为 15 毫秒，那么详细 GC 暂停时间图形将显示暂停时间保持在 15 毫秒标记周围或之下:

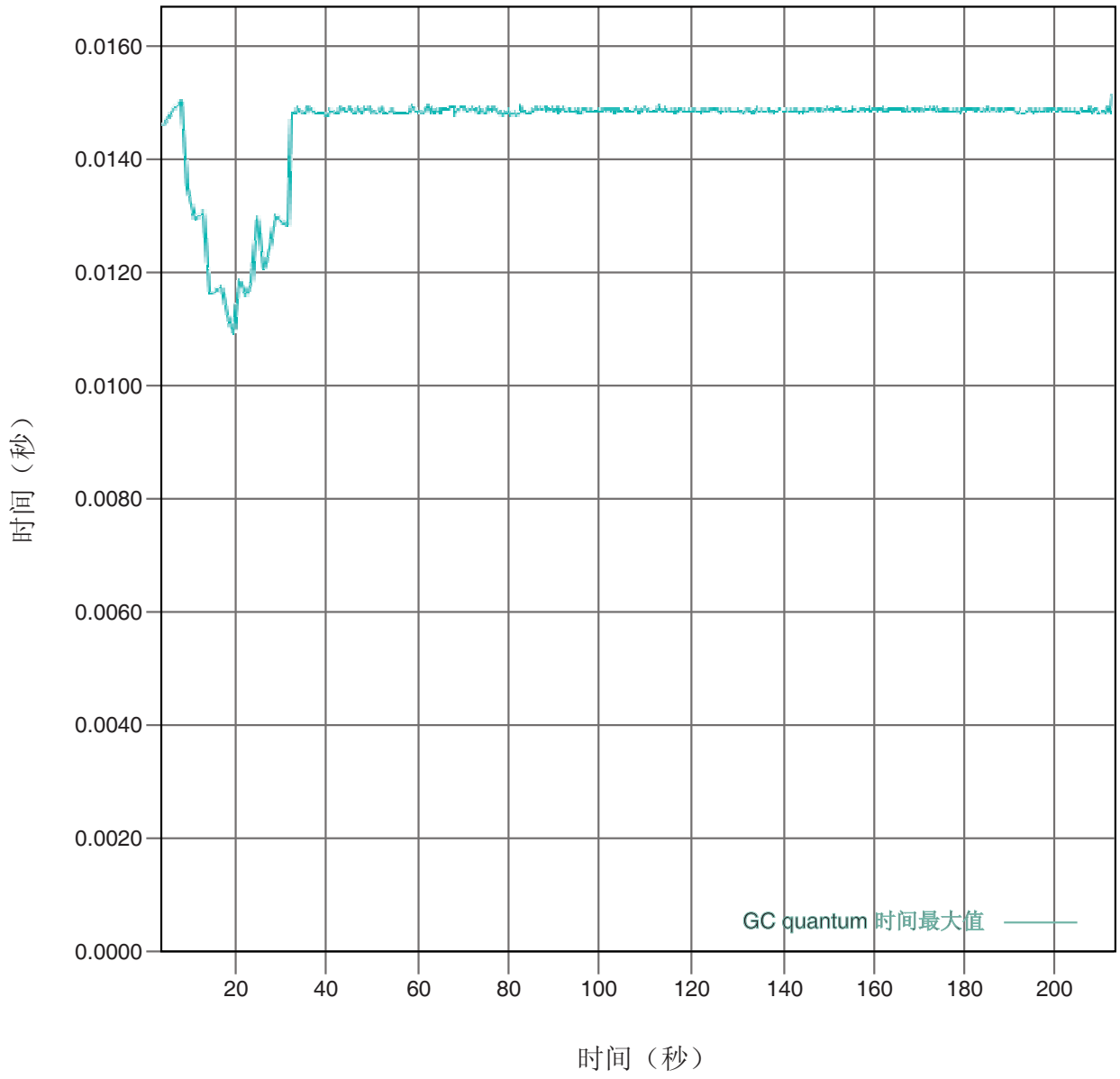

图 4. 目标暂停时间设置为 15 毫秒时的实际暂停时间

控制处理器利用率

您可以限制可供 Metronome 垃圾回收器使用的处理能力大小。

您可以使用 **-Xgc:targetUtilization=N** 选项限制 Metronome 垃圾回收器所使用的 CPU 量，以控制该垃圾回收器的垃圾回收。

例如:

```
java -Xgcpolicy:metronome -Xgc:targetUtilization=80 yourApplication
```

该示例指定应用程序在每 60 毫秒中运行 80% 的时间。其余 20% 的时间用于垃圾回收。如果给予 Metronome 垃圾回收器足够的资源，那么它可保证达到利用率级别。当堆中的可用空间量降低到动态确定的阈值以下时，会开始进行垃圾回收。

Metronome 垃圾回收器限制

本主题描述了影响 Metronome GC 策略的任何已知问题或限制。

x86 平台上的 AESNI 支持

x86 体系结构中 AESNI 指令的软件开发当前不支持与 Metronome GC 策略一起使用。

垃圾回收期间的长暂停时间

在极少数情况下，您在垃圾回收期间可能会遇到比预期更长的暂停。在垃圾回收期间，会使用根扫描进程。垃圾回收器会从已知的活动引用开始走遍整个堆。这些引用包括:

- 活动线程调用堆栈中的活动引用变量。
- 静态引用。

为查找应用程序线程的堆栈上的所有活动对象引用，垃圾回收器会扫描该线程的调用堆栈中的所有堆栈帧。将在不可中断的步骤中扫描每个活动线程堆栈。因此，扫描必须在单次 GC 暂停内进行。

其效果是如果您的一些线程具有非常深的堆栈，那么系统性能可能会比预期差，这是由回收周期开始时的延长了的垃圾回收暂停所致。

第 6 章 开发应用程序

关于编写实时应用程序（包括代码样本）的重要信息。

- 『样本实时散列映射』

样本实时散列映射

WebSphere Real Time for Linux 包括 HashMap 和 HashSet 实施，它们为 put 方法提供的性能比 IBM SDK for Java 7 中的标准 HashMap 更为稳定。

IBM 提供的标准 java.util.HashMap 可为高吞吐量应用程序提供优良的服务。它还对知道自身散列映射需要增加到的最大大小的应用程序提供帮助。对于需要增加到可变大小的散列映射的应用程序而言，根据不同的用法，标准散列映射存在潜在性能问题。对于使用 put 方法将新条目添加到散列映射而言，标准散列映射能够提供良好的响应时间。但是当散列映射填满后，必须分配一个更大的后备存储器。这意味着必须迁移当前后备存储器中的条目。如果散列映射较大，那么执行 put 的时间也较长。例如，操作可能需要几毫秒。

WebSphere Real Time for Linux 包含一个样本实时散列映射。它提供的功能接口与标准 java.util.HashMap 完全相同，但是为 put 方法提供能够提供更为稳定的性能。样本散列映射会另创建一个后备存储器，而不是在散列映射填满时创建后备存储器并迁移所有条目。新建的后备存储器会链接到该散列映射中的其他后备存储器。链接最初在分配空后备存储器并链接到其他后备存储器时会导致性能略微下降。后备散列映射更新后，迁移所有条目的速度比以前更快。实时散列映射的劣势在于 get、put 和 remove 操作的速度稍微变慢。操作速度变慢的原因在于每一项查找均必须在整个后备散列映射集（而不是某一个散列映射）中执行。

要试用实时散列映射，请将 `RTHashMap.jar` 文件添加到引导类路径的开头。如果将 WebSphere Real Time for Linux 安装到目录 `$WRT_ROOT` 中，请添加以下选项以将实时散列映射用于您的应用程序，而不是标准散列映射:

```
-Xbootclasspath/p:$WRT_ROOT/demo/realtime/RTHashMap.jar
```

用于实时散列映射实施的源和类文件包含在 `demo/realtime/RTHashMap.jar` 文件中。此外，还提供了实时 java.util.LinkedHashMap 和 java.util.HashSet 实施。

第 7 章 性能

对 WebSphere Real Time for Linux 针对始终短暂的 GC 暂停，而非最高吞吐量性能或最小内存占用量进行优化。

经认可的硬件配置上的性能

经认可的系统具有足够的时钟粒度和处理器速度来支持 WebSphere Real Time for Linux 性能目标。例如，未超负荷的系统上运行的编写良好并具有足够堆大小的应用程序一般会经历约 3 毫秒且不超过 3.2 毫秒的 GC 暂停时间。在 GC 周期内，具有缺省环境设置的应用程序不会暂停超过任何滑动的 60 毫秒时间窗内所耗用时间的 30%。在任意 60 毫秒周期内的 GC 暂停中所耗用的总时间通常总共为约 18 毫秒。

降低计时可变性

标准 JVM 中可变性的主要根源是垃圾回收暂停。在 WebSphere Real Time for Linux 中，通过使用 Metronome 垃圾回收器来避免标准垃圾回收器方式所带来的可能长时间暂停。请参阅第 20 页的『使用 Metronome 垃圾回收器』。

JVM 之间的类数据共享

类数据共享提供减少内存占用量和改善 JVM 启动时间的透明方法。要进一步了解类数据共享，请参阅『JVM 之间的类数据共享』。

压缩引用

Metronome GC 在 64 位平台上同时支持压缩和未压缩引用。在使用压缩引用时，JVM 会将所有引用作为 32 位值存储到对象、类、线程和监控器。使用压缩引用可提高许多应用程序的性能，因为对象变得更小，从而降低了垃圾回收频率并提高了内存高速缓存利用率。

注: 可用于压缩引用的堆大小限制为约 28 GB。
有关压缩引用的更多信息，请访问 http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/understanding/mm_compressed_references.html。

JVM 之间的类数据共享

无论是否使用 **-Xrealtime** 选项运行，对共享类的支持都相同。

通过将类数据存储在磁盘上的内存映射的缓存文件中，您可以在 Java 虚拟机 (JVM) 之间共享这些数据。当多个 JVM 共享一个高速缓存时，共享会降低总体虚拟存储器耗用。当创建高速缓存后，类共享还会降低 JVM 的启动时间。共享类高速缓存独立于任何运行的 JVM，并且持续存在直到它被删除。

共享的高速缓存可以包含:

- 引导程序类
- 应用程序类
- 描述了的元数据

- 提前（AOT）编译的代码

注: 非实时 JVM 不能移除实时共享类高速缓存。

第 8 章 安全性

本部分包含关于安全性的重要信息。

共享类高速缓存的安全注意事项

共享类高速缓存旨在更方便地执行高速缓存的管理和使用，但是缺省的安全策略可能并不适用。

使用共享类高速缓存时，必须了解新文件的缺省许可权，这样可通过限制访问权限来提高安全性。

文件	缺省许可权
新共享高速缓存	对组和其他条目的读许可权
javasharedresources 目录	读、写和执行许可权

您需要同时具有缓存文件和缓存目录的写许可权才可破坏或增加高速缓存。

更改缓存文件的文件许可权

要限制共享类高速缓存的访问权，可使用 **chmod** 命令。

所需更改	命令
限制用户和组的访问权限	chmod 770 /tmp/javasharedresources
限制用户的访问权限	chmod 700 /tmp/javasharedresources
仅限制用户对某特定高速缓存的读和写访问权	chmod 600 /tmp/javasharedresources/*<file for shared cache>*
仅限制用户和组对某特定高速缓存的读和写访问权	chmod 660 /tmp/javasharedresources/*<file for shared cache>*

连接到无访问许可权的高速缓存

如果尝试连接到无相应访问许可权的高速缓存，会看到一条错误消息:

```
JVMSHRC226E Error opening shared class cache file
JVMSHRC220E Port layer error code = -302
JVMSHRC221E Platform error message: Permission denied
JVMJ9VM015W Initialization error for library j9shr25(11): JVMJ9VM009E J9VMDllMain
failed
Could not create the Java virtual machine.
```

第 9 章 故障诊断与支持

WebSphere Real Time for Linux 的故障诊断与支持

常规问题确定方法

确定问题可以帮助您了解故障类型以及适当的纠正措施。

了解问题类型后，可能要执行以下一个或多个任务:

- 解决问题。
- 找到良好的变通方法。
- 收集用于向 IBM 生成错误报告的必要数据。

Linux 问题确定

本部分描述 Linux 上的问题确定。

IBM SDK for Java V7 用户指南包含了关于 Linux 上问题诊断的有用指南，涵盖以下内容:

- 设置并检查您的 Linux 环境
- 常用调试方法
- 崩溃诊断
- 调试挂起
- 调试内存泄漏
- 调试性能问题

您可以在以下位置找到此信息: IBM SDK for Java 7 - Linux problem determination。

以下信息是对 IBM WebSphere Real Time for Linux 的补充。

设置并检查您的 Linux 环境

在 IBM WebSphere Real Time for Linux 上，检查 JVM 是否已正确配置以生成系统转储。

Linux 系统转储（核心文件）

发生崩溃时，要获取的最重要诊断数据是 Linux 系统转储（核心文件）。要确保生成该文件，您必须检查操作系统设置以及可用磁盘空间，如 IBM SDK for Java V7 用户指南中所述。

Java 虚拟机设置

JVM 必须配置为在发生崩溃时生成核心文件。在命令行上运行 `java -Xdump:what`。该选项的输出是:

```
-Xdump:system:
    events=gpf+abort+traceassert+corruptcache,
    label=/mysdk/sdk/jre/bin/core.%Y%m%d.%H%M%S.%pid.dmp,
    range=1..0,
    priority=999,
    request=serial
```

所显示的值是缺省设置。在发生崩溃时，必须至少设置 events=gpf 以生成核心文件。您可以使用以下命令行选项来更改和设置选项
-Xdump:system[:name1=value1,name2=value2 ...]

常用调试方法

因为 Java 线程名称在操作系统中可视，所以您可以使用 ps 命令来帮助进行调试。在使用跟踪工具时，必须对 IBM WebSphere Real Time for Linux 使用正确的命令。

检查进程信息

在 IBM WebSphere Real Time for Linux 上运行 ps 命令时可预期看到的输出为:

```
ps -eLo pid,tid,rtprio,comm,cmd
13654 13654      - java            jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13655      - main            jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13656      - Signal Reporter jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13661      - JIT Compilation jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13662      - JIT Sampler     jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13666      - Signal Dispatch jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13667      - Finalizer maste jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13668      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13669      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13670      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13671      - Gc Slave Thread jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13672      - Metronome GC Al jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13673      - Thread-2        jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13698      - process reaper  jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13700      - stdout reader j jre/bin/java -Xgcpolicy:metronome -jar example.jar
13654 13701      - stderr reader j jre/bin/java -Xgcpolicy:metronome -jar example.jar
```

e 选择所有进程。

L 显示线程。

o 提供要显示的列的预定义格式。指定的列为进程标识、线程标识、调度策略、实时线程优先级以及与进程关联的命令。这些信息对于了解在给定时间应用程序和虚拟机中的什么线程正在运行很有用。

跟踪工具

Linux 上的三种跟踪工具是 **strace**、**ltrace** 和 **mtrace**。命令 man strace 显示完整一组可用选项。

strace

strace 工具跟踪系统调用。您可以在已经可用的进程上使用该工具，也可以通过新进程启动该工具。 strace 记录程序所发出的系统调用和进程所接收的信号。对于每个系统调用，均将使用名称、参数和返回值。strace 允许您在无需源的情况下跟踪程序（不需要进行重新编译）。如果您使用带有 **-f** 选项的 strace，那么将跟踪因派生系统调用而已创建的子进程。您可以使用 strace 来调查插件问题或尝试了解程序无法正确启动的原因。

要将 strace 用于 Java 应用程序，请输入 strace java -Xgcpolicy:metronome <class-name>。

您可以通过使用 -o选项来将跟踪输出从 strace 工具定向到文件。

ltrace

ltrace 工具依赖于分发。它与 strace 非常相似。该工具在执行进程进行调用时拦截并记录动态库调用。strace 对于执行进程所接收的信号执行上述同样的操作。

要将 ltrace 用于 Java 应用程序，请输入 `ltrace java -Xgcpolicy:metronome <class-name>`。

mtrace

mtrace 包含在 GNU 工具集中。它用于安装 malloc、realloc 和 free 的处理程序，并且使这些函数的所有用户能够被跟踪并记录到文件。该跟踪会降低程序效率，不应在一般使用情况下启用。要使用 mtrace，请将 **IBM_MALLOCTRACE** 设置为 1，并将 **MALLOC_TRACE** 设置为指向跟踪信息将存储到的有效文件。您必须拥有对该文件的写访问权。

要将 mtrace 用于 Java 应用程序，请输入:

```
export IBM_MALLOCTRACE=1
export MALLOC_TRACE=/tmp/file
java -Xgcpolicy:metronome <class-name>
mtrace /tmp/file
```

崩溃诊断

在收集崩溃之前关于运行中的进程以及 Java 环境的信息时，请遵循以下准则。

收集进程信息

当搜索在发生崩溃之前发生的事件时，请使用 **gdb** 和 **bt** 命令来显示故障线程的堆栈跟踪，而不是分析核心文件。

查明 Java 环境的相关信息

使用 Java 转储来确定各线程所执行的操作以及哪些 Java 方法在运行。将函数地址与库地址进行对照以确定在各个点运行的代码源。

使用 **-verbose:gc** 选项查看 Java 堆的状态。提出以下问题:

- 在可能导致崩溃的内存区域之一中是否存在内存不足情况？
- 崩溃是否在垃圾回收期间发生，从而指示可能是垃圾回收故障？
- 崩溃是否在垃圾回收之后发生，从而指示可能是内存损坏？

调试性能问题

在调试性能问题时，除了 IBM SDK for Java V7 用户指南中的主题之外，还请考虑 IBM WebSphere Real Time for Linux 的以下特定事项。

调整内存区域大小

Java 堆大小是 JVM 的最重要调整参数之一。选择正确大小可优化性能。使用正确大小可使垃圾回收器更容易提供所需的利用率。

有关调整内存区域大小的更多信息，请参阅第 60 页的『Metronome 垃圾回收器故障诊断』。

JIT 编译和性能

在使用 JIT 时，您应考虑对实时行为的影响。

Linux 上的已知限制

Linux 一直在快速发展，而 JVM 与操作系统之间的交互一直存在各种问题，尤其是在线程领域。

请注意以下可能影响 Linux 系统的限制。

作为进程的线程

如果 Java 线程的数量超过所允许的最大进程数，那么您的程序可能:

- 收到错误消息
- 收到 **SIGSEGV** 错误
- 停止

有关更多信息，请参阅位于以下位置的 *The Volano Report*: http://www.volano.com/report/index.html。

浮动堆栈限制

如果在无浮动堆栈的情况下运行，那么无论 **-Xss** 的设置如何，均会为每个线程提供最低 256 KB 的本机堆栈大小。

在浮动堆栈 Linux 系统上，会使用 **-Xss** 值。如果您从非浮动堆栈 Linux 系统进行迁移，请确保任何 **-Xss** 值都足够大并且不依赖于最小值 256 KB。

glibc 限制

如果您收到一条消息指示由于未找到符号（例如 __bzero）而无法装入 `libjava.so` 库，那么您可能安装了较低版本的 GNU C 运行时库 glibc。用于 Linux 线程实施的 SDK 需要 glibc V2.3.2 或更高版本。

字体限制

当您在 Red Hat 系统上进行安装时，为了让字体服务器能够找到 Java TrueType 字体，请运行（例如，在 Linux IA32 上）:

```
/usr/sbin/chkfontpath --add /opt/IBM/javawrt3[_64]/jre/lib/fonts
```

您必须在安装时执行该操作，并且必须以“root”用户身份登录来运行该命令。要了解更详细的字体问题，请参阅《Linux SDK 和运行时环境用户指南》。

Linux Completely Fair Scheduler 会影响 Java 的性能

经常使用同步的 Java 应用程序在包含 Completely Fair Scheduler 的 Linux 分发版上的运行效果可能很差。Completely Fair Scheduler (CFS) 是在发行版 2.6.23 时采用到主线 Linux 内核中的调度程序。CFS 算法不同于先前的 Linux 发行版的调度算法。它可能会改变某些应用程序的性能属性。特别而言，CFS 会以不同的方式实现 sched_yield()，使得生产线程更可能相应地获得 CPU 时间。

如果出现此问题，可以通过 Java 应用程序观察 CPU 的高使用率，并通过同步块减慢进度。此应用程序可能由于进度较慢而看似停止。

有两种可能的变通方法:

- 使用附加参数 **-Xthr:minimizeUserCPU** 来启动 JVM。
- 配置 Linux 内核以使用与较低版本兼容性更高的 sched_yield() 实施。通过将可调整内核属性 sched_compat_yield 设置为 **1** 来达到此目的。例如:

  ```
  echo "1" > /proc/sys/kernel/sched_compat_yield
  ```

如果性能不差，请不要使用这些变通方法。

此问题可能会影响包含 Completely Fair Scheduler 的 Linux 内核上运行的 IBM Developer Kit and Runtime Environment for Linux 5.0（所有版本）和 6.0（不高于 SR4 的所有版本）。对于 SR 4 之后的 IBM Developer Kit and Runtime Environment for Linux V6.0，将自动检测内核中 CFS 的使用并启用 **-Xthr:minimizeUserCPU** 选项。包含 Completely Fair Scheduler 的部分 Linux 分发版有 Ubuntu 8.04 和 SUSE Linux Enterprise Server 11。

更多有关 CFS 的信息，请访问使用 Completely Fair Scheduler 进行多处理。

Linux Red Hat MRG 内核中的性能问题

Red Hat MRG 内核的配置问题可能会导致在启用了详细垃圾回收的情况下启动 WebSphere Real Time 时应用程序线程意外暂停。这些暂停不会在详细 GC 输出中报告，但根据网络配置情况，可能会持续若干毫秒。从远程定义的 LDAP 用户启动的 JVM 最受影响，因为名称服务高速缓存守护程序 (nscd) 未启动，从而导致网络延迟。通过启动 nscd 可解决该问题。按照以下步骤来检查 nscd 服务的状态并更正此问题:

1. 通过输入以下命令来检查 nscd 守护程序是否正在运行:

   ```
   /sbin/service nscd status
   ```

   如果该守护程序未在运行，您将看到以下消息:

   ```
   nscd is stopped
   ```

2. 以 root 用户身份使用以下命令启动 nscd 服务:

   ```
   /sbin/service nscd start
   ```

3. 以 root 用户身份使用以下命令更改 nscd 服务的启动信息:

   ```
   /sbin/chkconfig nscd on
   ```

nscd 进程现正在运行，并会在重新引导之后自动启动。

NLS 问题确定

JVM 包含针对不同语言环境的内置支持。

IBM SDK for Java V7 用户指南包含关于诊断 NLS 问题的有用指南，涵盖以下内容:

- 字体概述
- 字体实用程序
- 常见 NLS 问题和可能的原因

您可以在以下位置找到此信息: IBM SDK for Java 7 - NLS problem determination。

ORB 问题确定

调试 ORB 问题的首要任务是确定问题位于分布式应用程序的客户机端还是服务器端。典型的 RMI-IIOP 会话可看作请求访问某对象的客户机和提供该对象的服务器之间的简单同步通信。

IBM SDK for Java V7 用户指南包含关于诊断 ORB 问题的有用指南，涵盖以下内容:

- 确定 ORB 问题
- 解释堆栈跟踪
- 解释 ORB 跟踪
- 常见问题
- IBM ORB 服务: 收集数据

您可以在以下位置找到此信息: IBM SDK for Java 7 - ORB problem determination。

以下信息是对 IBM WebSphere Real Time for Linux 的补充。

IBM ORB 服务: 收集数据

收集用于服务的 Java 版本输出时，请运行以下命令:

```
java -Xgcpolicy:metronome -version
```

初步测试

发生问题时，ORB 可能会生成包含以下内容的 `org.omg.CORBA.*` 异常:

- 指示原因的文本
- 次代码
- 完成状态

在推断 ORB 是问题的原因之前，请确保以下事项成立:

- 场景可在类似配置中重现。
- JIT 已禁用。
- 未在使用任何 AOT 已编译代码

其他操作包括:

- 关闭其他处理器。
- 如果可能，关闭同时多线程 (SMT)。
- 消除客户机或服务器的内存依赖性。物理内存短缺可能是低性能、明显挂起或崩溃的原因。要除去这些问题，请确保拥有合理的内存空间。
- 检查物理网络问题，例如防火墙、通信链路、路由器和 DNS 名称服务器。这些是 CORBA COMM_FAILURE 异常的主要原因。测试时，请 ping 您自己的工作站名称。
- 如果应用程序正在使用 DB2® 等数据库，请切换到最可靠的驱动程序。例如，要隔离 DB2 AppDriver，请切换到 Net Driver，该驱动程序速度较慢且使用套接字，但更加可靠。

OutOfMemory 错误故障诊断

处理 OutOfMemoryError 异常。

有关 Metronome 垃圾回收器的常规故障诊断信息，请参阅第 60 页的『Metronome 垃圾回收器故障诊断』。

诊断 OutOfMemoryError

在 Metronome 垃圾回收器中诊断 OutOfMemoryError 异常比在标准 JVM 中进行诊断复杂得多，这是由此垃圾收集器的周期性决定的。

通常，实时应用程序需要的堆空间大约比标准 Java 应用程序多 20%。

缺省情况下，发生未捕获的 OutOfMemoryError 时，JVM 将生成以下诊断输出:

- 快照转储；请参阅第 43 页的『使用转储代理程序』。
- Heapdump；请参阅第 50 页的『使用 Heapdump』。
- Javadump；请参阅第 46 页的『使用 Javadump』。
- 系统转储；请参阅第 53 页的『使用系统转储和转储查看器』。

转储文件名在控制台输出中给出:

```
JVMDUMP006I Processing dump event "systhrow", detail "java/lang/OutOfMemoryError" - please wait.
JVMDUMP007I JVM Requesting Snap dump using 'Snap.20081017.104217.13161.0001.trc'
JVMDUMP010I Snap dump written to Snap.20081017.104217.13161.0001.trc
JVMDUMP007I JVM Requesting Heap dump using 'heapdump.20081017.104217.13161.0002.phd'
JVMDUMP010I Heap dump written to heapdump.20081017.104217.13161.0002.phd
JVMDUMP007I JVM Requesting Java dump using 'javacore.20081017.104217.13161.0003.txt'
JVMDUMP010I Java dump written to javacore.20081017.104217.13161.0003.txt
JVMDUMP013I Processed dump event "systhrow", detail "java/lang/OutOfMemoryError".
```

显示在控制台输出中并且在 Javadump 中提供的 Java 回溯指示了 Java 应用程序中发生 OutOfMemoryError 的位置。JVM 内存管理组件发出一个跟踪点，该跟踪点提供了失败分配的大小、类块地址和内存空间名称。您可以在快照转储中找到此跟踪点:

```
<< lines omitted... >>
09:42:17.563258000 *0xf2888e00       j9mm.101  Event       J9AllocateIndexableObject() returning NULL! 80
bytes requested for object of class 0xf1632d80 from memory space 'Metronome' id=0xf288b584
```

跟踪点标识和数据字段可能与显示的内容不同，具体取决于所分配的对象的类型。在此示例中，跟踪点指出，当应用程序尝试在 Metronome heap 的内存段 id=0x809c5f0 中分配大小为 33.6 MB 且类型为 class 0x81312d8 的对象时，发生分配故障。

您可以通过查看 Javadump 中的内存管理信息来确定受影响的 内存区域:

```
NULL           ------------------------------------------------------------
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
NULL
1STMEMTYPE     Object Memory
NULL           region     start      end        size       name
1STHEAP        0xF288B584 0xF2A1C000 0xF6A1C000 0x04000000 Default
NULL
1STMEMUSAGE    Total memory available: 67108864 (0x04000000)
1STMEMUSAGE    Total memory in use:    66676824 (0x03F96858)
1STMEMUSAGE    Total memory free:      00432040 (0x000697A8)

<< lines removed for clarity >>
```

可以通过查看 Javadump 的类部分确定所分配对象的类型:

```
NULL           ------------------------------------------------------------
0SECTION       CLASSES subcomponent dump routine
NULL           ================================
<< lines omitted... >>
1CLTEXTCLLOD    ClassLoader loaded classes
2CLTEXTCLLOAD    Loader *System*(0xF182BB80)
<< lines omitted... >>
3CLTEXTCLASS      [C(0xF1632D80)
```

Javadump 中的信息确认所尝试的分配是针对字符数组在正常堆 (ID=0xF288B584) 中进行，并且该堆的分配总大小（由相应的 1STHEAP 行指示）是十进制字节数 67108864 或者十六进制字节数 0x04000000，即 64 MB。

在此示例中，失败的分配相对于堆的总大小较大。如果应用程序应该创建 33 MB 的对象，那么下一步是使用 **-Xmx** 选项增大堆大小。

更为常见的情况时，失败的分配相对于堆的总大小较小。这是因为，先前的分配已将堆用尽。在这些情况下，下一步是使用 Heapdump 来调查分配给现有对象的内存量。

Heapdump 是一个压缩二进制文件，其中包含所有对象及其对象类、大小和引用的列表。请使用 IBM Monitoring and Diagnostic Tools for Java - Memory Analyzer 工具来分析 Heapdump，此工具可以从 IBM Support Assistant (ISA) 下载。

通过使用 MDD4J，可以装入 Heapdump 并找到可能耗用了大量堆空间的对象的树结构。此工具可以提供堆中的对象的各种视图。例如，MDD4J 可以显示一个视图以详细描述可疑的泄漏，并给出堆中前 5 个最大的对象和包。选择树形视图将提供有关发生泄漏的容器对象的性质的更多信息。

IBM JVM 管理内存的方式

IBM JVM 需要内存以用于多个不同的组件，包括类、已编译的代码、Java 对象、Java 堆栈和 JNI 堆栈的内存区域。其中的一些内存区域必须在连续内存中。其他内存区域可以分段为较小的区域并链接到一起。

动态装入的类和已编译的代码存储在动态装入的类的分段内存区域中。这些类进一步划分到可写内存区域（RAM 类）和只读内存区域（ROM 类）中。在运行时，应用程序启动期间，将对类高速缓存中的 ROM 类和 AOT 代码进行内存映射（但不会将其装入），从而将其映射到连续内存区域。应用程序引用这些类时，类高速缓存中的类以及已编译的代码将映射到存储器。类的 ROM 组成部分由多个引用了这个类的进程共享。类的 RAM 组成部分在动态装入的类首次被 JVM 引用时，在这个类的分段内存区域中创建。类高速缓存中各个类方法的 AOT 编译代码将复制到可执行动态代码内存区域中，这是因为，各个进程不会共享此代码。并非从类高速缓存中装入的类与高速缓存的类相似，只不过 ROM 类信息在动态装入的类的分段内存区域中创建。动态生成的代码与高速缓存的类的 AOT 代码存储在相同的动态代码内存区域中。

每个 Java 线程的堆栈都可以跨分段内存区域。每个线程的 JNI 堆栈都占用连续的内存区域。

要确定如何配置 JVM，请在使用 **-verbose:sizes** 选项的情况下运行。此选项将打印出允许您管理大小的内存区域的相关信息。对于非连续内存区域，将打印一个增量，以描述每次需要增大该区域时获取的内存量。

以下是使用 **-Xrealtime -verbose:sizes** 选项时的输出示例:

```
-Xmca32K                    RAM 类段增量
-Xmco128K                   ROM 类段增量
-Xms64M                     初始内存大小

-Xmx64M                     最大内存
-Xmso256K                   操作系统线程堆栈大小
-Xiss2K                     Java 线程堆栈初始大小
-Xssi16K                    Java 线程堆栈增量
-Xss256K                    Java 线程堆栈最大大小
```

此示例指示 RAM 类段最初为 0，但根据需要以 32 KB 块为增量增大。ROM 类段最初为 0，并且根据需要以 128 KB 块为增量增大。您可以使用 **-Xmca** 和 **-Xmco** 选项来控制这些大小。RAM 类和 ROM 类段根据需要增大，因此您通常不需要更改这些选项。

使用 **-Xshareclasses** 选项可以确定使用类高速缓存时内存映射区域的大小。以下是 `java -Xgcpolicy:metronome -Xshareclasses:printStats` 命令的输出样本。

```
Current statistics for cache "sharedcc_chamlain":


base address = 0xF1BBD000
end address = 0xF2BAF000
allocation pointer = 0xF1CA95A0

cache size = 16776852
free bytes = 15499564
ROMClass bytes = 1198572
AOT bytes = 0
Data bytes = 57300
Metadata bytes = 21416
Metadata % used = 1%

# ROMClasses = 368
# AOT Methods = 0
# Classpaths = 1
# URLs = 0
# Tokens = 0
# Stale classes = 0
% Stale classes = 0%

Cache is 7% full
```

在运行时，引用类时，将大约 3 MB 的 AOT 字节和元数据字节复制到动态代码分段区域。引用类时，数据字节将复制到 RAM 类分段区域。

使用诊断工具

有多种可用于帮助诊断 IBM WebSphere Real Time for Linux JVM 问题的诊断工具。

IBM SDK for Java 7 提供了多种诊断工具，这些工具可用于诊断 IBM WebSphere Real Time for Linux JVM 的问题。本部分介绍可用的工具，并提供关于如何使用这些工具的更多信息的链接。

在使用 SDK 诊断工具时有一个要点需谨记。在调用实时 JVM 时，请使用以下选项:

```
java  -Xgcpolicy:metronome
```

在为实时 JVM 运行诊断工具时必须使用该选项。例如，要显示 IBM WebSphere Real Time for Linux JVM 的已注册转储代理进程，请输入:

```
java -Xgcpolicy:metronome -Xdump:what
```

此处作为补充信息提供了关于对 IBM WebSphere Real Time for Linux 使用这些工具的任何其他差异，还一起提供了用来帮助您进行诊断的样本输出。

要查看 IBM SDK for Java 7 所生成的诊断信息的摘要，请访问 Summary of diagnostic information。

使用 IBM Monitoring and Diagnostic Tools for Java

IBM 提供了工具和文档来帮助您了解、监视和诊断使用 IBM JRE 的应用程序中的问题。

以下是可用的工具:

- 运行状况中心
- 垃圾回收和内存可视化器
- 交互式诊断数据资源管理器
- 内存分析器

垃圾回收和内存可视化器

垃圾回收和内存可视化器 (GCMV) 帮助您了解 Java 应用程序的内存使用、垃圾回收行为以及性能。

GCMV 解析和绘制来自不同类型日志的数据，包括以下类型:

- 详细垃圾回收日志。
- 跟踪垃圾回收日志，通过使用 -Xtgc 参数生成。
- 本机内存日志，通过使用 ps、**svmon** 或 **perfmon** 系统命令生成。

该工具有助于诊断问题（例如内存泄漏）、分析各种可视格式的数据以及提供调优建议。

GCMV 是以 IBM Support Assistant (ISA) 附加组件的形式提供的。有关该附加组件的安装和入门信息，请参阅: http://www.ibm.com/developerworks/java/jdk/tools/gcmv/。

IBM 信息中心提供了有关 GCMV 的进一步信息。

运行状况中心

“运行状况中心”是用于监视正在运行的 Java 虚拟机 (JVM) 状态的诊断工具。

此工具被分成两个部分来提供:

- 可从正在运行的应用程序中收集数据的 Health Center 代理程序。
- 会连接到代理程序的基于 Eclipse 的客户机。客户机可解释数据，还能提供可提高受监控应用程序性能的建议。

运行状况中心是以 IBM Support Assistant (ISA) 附加组件的形式提供的。有关该附加组件的安装和入门信息，请参阅: http://www.ibm.com/developerworks/java/jdk/tools/healthcenter/。

IBM 信息中心提供了有关运行状况中心的进一步信息。

交互式诊断数据资源管理器

交互式诊断数据资源管理器 (IDDE) 是转储查看器（**jdmpview** 命令）的基于 GUI 的替代方法。 IDDE 提供的功能与转储查看器相同，但还具有额外的支持功能，例如保存命令输出的功能。

使用 IDDE 可以更轻松地浏览和检查 JVM 生成的转储文件。在 IDDE 中，您可以在调查日志中输入命令来浏览转储文件。调查日志提供的支持包括以下项:

- 命令帮助
- 自动完成文本和某些参数，例如类名
- 保存命令和输出的功能，您随后可以将其发送给其他人
- 突出显示的文本和问题标记
- 添加个人注释的功能
- IDDE 中支持使用 Memory Analyzer

IDDE 是以 IBM Support Assistant (ISA) 附加组件的形式提供的。有关该附加组件的安装和入门信息，请参阅 developerWorks® 上的 IDDE概述。

IBM信息中心提供了与 IDDE 有关的更多信息。

内存分析器

内存分析器帮助您分析使用操作系统级别转储和“可移植堆转储 (PHD)”的 Java 堆。

该工具可以分析包含上百万对象的转储，从而提供以下信息:

- 对象的保留大小。
- 阻止垃圾回收器回收对象的进程。
- 自动截取可疑的泄漏的报告。

该工具基于 Eclipse Memory Analyzer (MAT) 项目，并且使用 IBM Diagnostic Tool Framework for Java (DTFJ) 功能部件以启用对 IBM JVM 中转储的处理。

内存分析器是以 IBM Support Assistant (ISA) 附加组件的形式提供的。有关该附加组件的安装和入门信息，请参阅: http://www.ibm.com/developerworks/java/jdk/tools/memoryanalyzer/。

IBM 信息中心提供了有关内存分析器的进一步信息。

使用转储代理程序

在 JVM 初始化期间设置转储代理程序。这使您能够使用在 JVM 中发生的事件，例如，垃圾回收、线程启动或 JVM 终止来启动转储或启动外部工具。

《IBM SDK for Java V7 用户指南》包含有关转储代理程序的有用指导，包括:

- 使用 **-Xdump** 选项
- 转储代理程序
- 转储事件
- 对转储代理程序的高级控制
- 转储代理程序令牌
- 缺省转储代理程序

- 除去转储代理程序
- 转储代理程序环境变量
- 信号映射
- 转储代理程序缺省位置

您可以在此处查找信息: IBM SDK for Java 7 - 使用转储代理程序。

此处为 IBM WebSphere Real Time for Linux 提供了补充信息:

转储事件

转储代理程序由 JVM 操作期间发生的事件触发。对于 IBM WebSphere Real Time for Linux, 慢事件的缺省值为 5 毫秒。

可过滤某些事件来提高输出的相关性。请参阅 第 45 页的『filter 选项』以获取更多信息。

注: 卸装和扩展事件当前不会在 WebSphere Real Time 中发生。类位于永久内存中, 不能对其进行卸装。

注: gpf 和 abort 事件不能触发堆转储、准备堆 (request=prepwalk) 或压缩堆 (request=compact)。

下表显示了可用作转储代理程序触发器的事件:

事件	触发条件...	过滤操作
gpf	发生一般保护故障 (GPF) 。	
user	JVM 从操作系统收到 SIGQUIT 信号。	
abort	JVM 从操作系统收到 SIGABRT 信号。	
vmstart	虚拟机已启动。	
vmstop	虚拟机已停止。	过滤退出码; 例如, **filter=#129..#192#-42#255**
load	已装入类。	过滤类名; 例如, **filter=java/lang/String**
unload	已卸装类。	
throw	抛出异常。	过滤异常类名; 例如, **filter=java/lang/OutOfMem***
catch	捕获到异常。	过滤异常类名; 例如, **filter=*Memory***
uncaught	应用程序未捕获到 Java 异常。	过滤异常类名; 例如, **filter=*MemoryError**
systhrow	Java 异常将由 JVM 抛出。这与“throw”事件不同, 因为它仅针对在 JVM 内部检测到的错误条件触发。	过滤异常类名; 例如, **filter=java/lang/OutOfMem***
thrstart	启动了新线程。	
blocked	线程被阻止。	
thrstop	线程停止。	
fullgc	启动了垃圾收集循环。	
slow	线程响应内部 JVM 请求的时间超过 5ms。	更改被视为事件很慢的时间量; 例如, 如果线程响应内部 JVM 请求的时间超过 300ms, 那么将触发 **filter=#300ms**。

事件	触发条件...	过滤操作
allocation	为 Java 对象分配了与给定过滤器规范相匹配的大小	过滤对象大小；必须已提供过滤器。例如，**filter=#5m** 将在大于 5 Mb 的对象上触发。也支持使用范围；例如，**filter=#256k..512k** 将在大小介于 256 Kb 到 512 Kb 之间的对象上触发。
traceassert	JVM 中发生内部错误	不适用。
corruptcache	JVM 发现共享类高速缓存已损坏。	不适用。

filter 选项

某些 JVM 事件在应用程序的生命周期中可出现上千次。转储代理程序可使用过滤器和范围来避免生成过多的转储。

通配符

可通过仅在过滤器的开头或结尾使用星号来在异常事件过滤器中使用通配符。以下命令无效，因为第二个星号不在结尾处:

```
-Xdump:java:events=vmstop,filter=*InvalidArgumentException#*.myVirtualMethod
```

为了使此过滤器有效，必须将其更改为:

```
-Xdump:java:events=vmstop,filter=*InvalidArgumentException#MyApplication.*
```

类装入和异常事件

可以通过 Java 类名来过滤类装入 (load) 和异常（throw、catch、uncaught 和 systhrow）事件:

```
-Xdump:java:events=throw,filter=java/lang/OutOfMem*
-Xdump:java:events=throw,filter=*MemoryError
-Xdump:java:events=throw,filter=*Memory*
```

您可以通过 Java 方法名称来过滤 throw、uncaught 和 systhrow 异常事件:

```
-Xdump:java:events=throw,filter=ExceptionClassName[#ThrowingClassName.
throwingMethodName[#stackFrameOffset]]
```

可选部分将显示在方括号中。

您可以通过 Java 方法名称来过滤捕获异常事件:

```
-Xdump:java:events=catch,filter=ExceptionClassName[#CatchingClassName.
catchingMethodName]
```

可选部分将显示在方括号中。

vmstop 事件

可以通过使用一个或多个退出码来过滤 JVM 关闭事件:

```
-Xdump:java:events=vmstop,filter=#129..192#-42#255
```

慢事件

您可以过滤慢事件以更改缺省值为 5ms 的时间阈值:

```
-Xdump:java:events=slow,filter=#300ms
```

不能将过滤器设置为慢于缺省时间的时间。

分配事件

您必须过滤分配事件以指定导致触发的对象大小。您可以将过滤器大小从零设置为 32 位平台上 32 位指针的最大值，或 64 位平台上 64 位指针的最大值。将较低的过滤器值设置为零将在所有分配上触发转储。

例如，要在大小大于 5 Mb 的分配上触发转储，请使用:

```
-Xdump:stack:events=allocation,filter=#5m
```

要在大小介于 256Kb 到 512Kb 之间的分配上触发转储，请使用:

```
-Xdump:stack:events=allocation,filter=#256k..512k
```

其他事件

如果将过滤器应用于不支持过滤的事件，将忽略过滤器。

使用 Javadump

Javadump 生成的文件包含与 JVM 以及在执行期间的某个点捕获的 Java 应用程序相关的信息。例如，此信息可能与操作系统、应用程序环境、线程、堆栈、锁定和内存有关。

《IBM SDK for Java V7 用户指南》包含有关 Javadump 的有用指导，包括:

- 启用 Javadump
- 触发 Javadump
- 解释 Javadump
- 环境变量和 Javadump

您可以在此处查找信息: IBM SDK for Java 7 - 使用 Javadump。

以下主题中提供了 IBM WebSphere Real Time for Linux 的补充信息和样本输出。

存储管理 (MEMINFO)

MEMINFO 部分提供了有关 Memory Manager 的信息，包括堆、永久和作用域内存区域。

Javadump 的 MEMINFO 部分显示了有关 Memory Manager 的信息。请参阅使用 Metronome 垃圾回收器，获取有关 Memory Manager 组件工作方式的详细信息。

这部分的 Javadump 提供了不同存储管理值，包括:

- 可用内存量
- 已使用内存量
- 堆的当前大小
- 永久内存区域的当前大小
- 作用域内存区域的当前大小

本部分还包含垃圾回收历史数据。这些数据显示为跟踪点序列，每一个都带有时间戳记，最近的跟踪点排在最前面。

由标准 JVM 生成的 Javadump 包含“GC History”部分。该信息不会包含在使用实时 JVM 时生成的 Javadump 中。使用 **-verbose:gc** 选项或 JVM 快照跟踪获取有关 GC 行为的信息。请参阅第 60 页的『使用 verbose:gc 信息』和《IBM SDK for Java V7 用户指南》的堆代理程序部分以获取更多详细信息。

在 Javadump 中，分段是 Java 运行时为使用大量内存的任务分配的内存块。示例任务包括:

- 维护 JIT 高速缓存
- 存储 Java 类

Java 运行时环境还会分配其他本机内存，这些内存未在 MEMINFO 部分中列出。Java 运行时分段所使用内存总量并不必然代表 Java 运行时的完全内存占用量。Java 运行时分段由分段数据结构和相关的本机内存块构成。

以下示例显示了一些典型输出。所有值都以十六进制值形式提供。MEMINFO 部分的列标题的含义如下:

- 对象内存部分 (HEAPTYPE):

  id 空间或区域的标识。

  start 堆的此区域的开始地址。

  end 堆的此区域的结束地址。

  size 堆的此区域的大小。

  space/region
  对于仅包含 id 和名称的行，此列显示了内存空间的名称。其他情况下，列显示了内存空间的名称，后面是包含在该内存空间中的特定区域的名称。

- 内部内存部分 (SEGTYPE)，包括类内存、JIT 代码高速缓存和 JIT 数据高速缓存:

  segment
  分段控制数据结构的地址。

  start 本机内存分段的开始地址。

  alloc 本机内存分段中的当前分配地址。

  end 本机内存分段的结束地址。

  type 内部位字段，描述本机内存分段的特征。

  size 本机内存分段的大小。

```
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
NULL
1STHEAPTYPE    Object Memory
NULL           id         start      end        size       space/region
1STHEAPSPACE   0x00497030     --         --         --     Generational
1STHEAPREGION  0x004A24F0 0x02850000 0x05850000 0x03000000 Generational/Tenured Region
1STHEAPREGION  0x004A2468 0x05850000 0x06050000 0x00800000 Generational/Nursery Region
1STHEAPREGION  0x004A23E0 0x06050000 0x06850000 0x00800000 Generational/Nursery Region
NULL
1STHEAPTOTAL   Total memory:           67108864 (0x04000000)
1STHEAPINUSE   Total memory in use:    33973024 (0x02066320)
1STHEAPFREE    Total memory free:      33135840 (0x01F99CE0)
NULL
1STSEGTYPE     Internal Memory
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x073DFC9C 0x0761B090 0x0761B090 0x0762B090 0x01000040 0x00010000
       (lines removed for clarity)
1STSEGMENT     0x00497238 0x004FA220 0x004FA220 0x0050A220 0x00800040 0x00010000
NULL
```

```
1STSEGTOTAL    Total memory:           873412 (0x000D53C4)
1STSEGINUSE    Total memory in use:         0 (0x00000000)
1STSEGFREE     Total memory free:      873412 (0x000D53C4)
NULL
1STSEGTYPE     Class Memory
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x0731C858 0x0745C098 0x07464098 0x07464098 0x00010040 0x00008000
        (lines removed for clarity)
1STSEGMENT     0x00498470 0x070079C8 0x07026DC0 0x070279C8 0x00020040 0x00020000
NULL
1STSEGTOTAL    Total memory:          2067100 (0x001F8A9C)
1STSEGINUSE    Total memory in use:   1839596 (0x001C11EC)
1STSEGFREE     Total memory free:     227504 (0x000378B0)
NULL
1STSEGTYPE     JIT Code Cache
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x004F9168 0x06960000 0x069E0000 0x069E0000 0x00000068 0x00080000
NULL
1STSEGTOTAL    Total memory:           524288 (0x00080000)
1STSEGINUSE    Total memory in use:    524288 (0x00080000)
1STSEGFREE     Total memory free:           0 (0x00000000)
NULL
1STSEGTYPE     JIT Data Cache
NULL           segment    start      alloc      end        type       size
1STSEGMENT     0x004F92E0 0x06A60038 0x06A6839C 0x06AE0038 0x00000048 0x00080000
NULL
1STSEGTOTAL    Total memory:           524288 (0x00080000)
1STSEGINUSE    Total memory in use:     33636 (0x00008364)
1STSEGFREE     Total memory free:      490652 (0x00077C9C)
NULL
1STGCHTYPE     GC History
3STHSTTYPE     15:18:14:901108829 GMT j9mm.134 -   Allocation failure end: newspace=7356368/8388608
oldspace=32038168/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:901104380 GMT j9mm.470 -   Allocation failure cycle end: newspace=7356416/8388608
oldspace=32038168/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:901097193 GMT j9mm.65 -   LocalGC end: rememberedsetoverflow=0
causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=0 failedflipbytes=0 failedtenurecount=0
failedtenurebytes=0 flipcount=11454 flipbytes=991056 newspace=7356416/8388608 oldspace=32038168/50331648
loa=3523072/3523072 tenureage=1
3STHSTTYPE     15:18:14:901081108 GMT j9mm.140 -   Tilt ratio: 50
3STHSTTYPE     15:18:14:893358658 GMT j9mm.64 -   LocalGC start: globalcount=3 scavengecount=24 weakrefs=0
soft=0 phantom=0 finalizers=0
3STHSTTYPE     15:18:14:893354551 GMT j9mm.63 -   Set scavenger backout flag=false
3STHSTTYPE     15:18:14:893348733 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.002
meanexclusiveaccessms=0.002 threads=0 lastthreadtid=0x00495F00 beatenbyotherthread=0
3STHSTTYPE     15:18:14:893348391 GMT j9mm.469 -   Allocation failure cycle start: newspace=0/8388608
oldspace=38199368/50331648 loa=3523072/3523072 requestedbytes=48
3STHSTTYPE     15:18:14:893347364 GMT j9mm.133 -   Allocation failure start: newspace=0/8388608
oldspace=38199368/50331648 loa=3523072/3523072 requestedbytes=48
3STHSTTYPE     15:18:14:866523613 GMT j9mm.134 -   Allocation failure end: newspace=2359064/8388608
oldspace=38199368/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:866519507 GMT j9mm.470 -   Allocation failure cycle end: newspace=2359296/8388608
oldspace=38199368/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:866513004 GMT j9mm.65 -   LocalGC end: rememberedsetoverflow=0
causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=5056 failedflipbytes=445632
failedtenurecount=0 failedtenurebytes=0 flipcount=9212 flipbytes=6017148 newspace=2359296/8388608
oldspace=38199368/50331648 loa=3523072/3523072 tenureage=1
3STHSTTYPE     15:18:14:866493839 GMT j9mm.140 -   Tilt ratio: 64
3STHSTTYPE     15:18:14:859814852 GMT j9mm.64 -   LocalGC start: globalcount=3 scavengecount=23 weakrefs=0
soft=0 phantom=0 finalizers=0
3STHSTTYPE     15:18:14:859808692 GMT j9mm.63 -   Set scavenger backout flag=false
3STHSTTYPE     15:18:14:859801848 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.004
meanexclusiveaccessms=0.004 threads=0 lastthreadtid=0x00495F00 beatenbyotherthread=0
3STHSTTYPE     15:18:14:859801163 GMT j9mm.469 -   Allocation failure cycle start: newspace=0/10747904
oldspace=38985800/50331648 loa=3523072/3523072 requestedbytes=232
3STHSTTYPE     15:18:14:859800479 GMT j9mm.133 -   Allocation failure start: newspace=0/10747904
oldspace=38985800/50331648 loa=3523072/3523072 requestedbytes=232
3STHSTTYPE     15:18:14:652219028 GMT j9mm.134 -   Allocation failure end: newspace=2868224/10747904
oldspace=38985800/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:650796714 GMT j9mm.470 -   Allocation failure cycle end: newspace=2868224/10747904
oldspace=38985800/50331648 loa=3523072/3523072
3STHSTTYPE     15:18:14:650792607 GMT j9mm.475 -   GlobalGC end: workstackoverflow=0 overflowcount=0
memory=41854024/61079552
3STHSTTYPE     15:18:14:650784052 GMT j9mm.90 -   GlobalGC collect complete
3STHSTTYPE     15:18:14:650780971 GMT j9mm.57 -   Sweep end
3STHSTTYPE     15:18:14:650611567 GMT j9mm.56 -   Sweep start
3STHSTTYPE     15:18:14:650610540 GMT j9mm.55 -   Mark end
3STHSTTYPE     15:18:14:645222792 GMT j9mm.54 -   Mark start
3STHSTTYPE     15:18:14:645216632 GMT j9mm.474 -   GlobalGC start: globalcount=2
```

```
          (lines removed for clarity)
NULL
NULL ------------------------------------------------------------------------
```

线程和堆栈跟踪 (THREADS)

对于应用程序员而言，Java 转储最有用的部分之一是 THREADS 节。该节显示 Java 线程、本机线程和堆栈跟踪的列表。

Java 线程由操作系统的本机线程执行。每个线程用诸如以下的行表示:

```
"main" J9VMThread:0x41D11D00, j9thread_t:0x003C65D8, java/lang/Thread:0x40BD6070, state:CW, prio=5
 (native thread ID:0xA98, native priority:0x5, native policy:UNKNOWN)
 Java callstack:
 at java/lang/Thread.sleep(Native Method)
 at java/lang/Thread.sleep(Thread.java:862)
 at mySleep.main(mySleep.java:31)
```

使用 **ps** 命令时，Java 线程名称在操作系统中是可见的。更多关于使用 **ps** 命令的信息，请参阅第 34 页的『常用调试方法』。

第一行中的属性包括 JVM 线程结构和 Java 线程对象的线程名称、地址，以及线程状态和 Java 线程优先级。第二行中的属性包括本机操作系统线程标识、本机操作系统线程优先级和本机操作系统调度策略。

线程名称可在以下三种方式中可见:

- 列出在 javacore 文件中。并非所有线程都会列出在 javacore 文件中。
- 使用 **ps** 命令列出操作系统中的线程时。
- 使用 java.lang.Thread.getName() 方法时。

下表提供了有关 IBM WebSphere Real Time for Linux 线程名称的信息。

表 4. IBM WebSphere Real Time for Linux 中的线程名称

线程详细信息	线程名称
由垃圾回收模块使用的内部 JVM 线程，用于分派由辅助线程执行的对象最终化。	Finalizer master
由垃圾回收器使用的警报线程。	GC Alarm
用于垃圾回收的从属线程。	GC Slave
由即时编译器模块使用的内部 JVM 线程，用于对应用程序中方法的使用情况进行采样。	IProfiler
由 VM 使用的线程，用于管理应用程序收到的信号，而无论信号由内部还是外部生成。	Signal Reporter
用于编译 Java 代码的内部 JVM 线程。	JIT Compilation Thread
用于允许将 JVMTI 代理程序附加到运行中的 JVM 的内部 JVM 线程。	Attach API wait loop

Java 线程优先级会根据平台映射至操作系统优先级值。较大的 Java 线程优先级值表明该线程具有较高的优先级。换言之，该线程会比低优先级线程更频繁地运行。

状态值可以是:

- R - 可运行 - 条件满足时，该线程将运行。
- CW - 等待条件 - 该线程正在等待。例如，由于:

  - 调用了 sleep()
  - 针对 I/O 已阻止了该线程
  - 调用了 wait() 方法，以等待通知监视器
  - 该线程通过 join() 调用正在与另一个线程同步
- S – 已暂挂 – 该线程已被另一个线程暂挂。
- Z – 僵死 – 该线程已被结束。
- P – 停放 – 该线程已因新的并发 API (java.util.concurrent) 而被停放。
- B – 已阻塞 – 该线程正在等待获取其他对象当前拥有的锁。

如果某个线程已停发或已阻塞，则输出中包含针对该线程的一行，以 3XMTHREADBLOCK 开始，列出该线程要等待的资源，以及当前拥有该资源的线程（如果可能）。有关更多信息，请参阅《IBM SDK for Java V7 用户指南》中关于已阻塞线程的主题。

当您启动 Javadump 以获取诊断信息时，JVM 会在生成 javacore 之前停顿 Java 线程。TITLE 部分的 1TIPREPSTATE 行中会显示 exclusive_vm_access 的准备状态。

```
1TIPREPSTATE Prep State: 0x4 (exclusive_vm_access)
```

当触发 javacore 时运行 Java 代码的线程目前处于 CW（条件等待）状态。

```
3XMTHREADINFO      "main" J9VMThread:0x41481900, j9thread_t:0x002A54A4, java/lang/Thread:0x004316B8,
state:CW, prio=5
3XMTHREADINFO1            (native thread ID:0x904, native priority:0x5, native policy:UNKNOWN)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at java/lang/String.getChars(String.java:667)
4XESTACKTRACE                at java/lang/StringBuilder.append(StringBuilder.java:207)
```

javacore LOCKS 部分显示了这些线程正在内部 JVM 锁定中等待。

```
2LKREGMON          Thread public flags mutex lock (0x002A5234): <unowned>
3LKNOTIFYQ            Waiting to be notified:
3LKWAITNOTIFY            "main" (0x41481900)
```

使用 Heapdump

术语 Heapdump 描述了 IBM Virtual Machine for Java 机制，该机制生成 Java 堆上所有活动对象的转储，即，正在运行的 Java 应用程序使用的那些活动对象。

《IBM SDK for Java V7 用户指南》包含有关 Heapdump 的有用指导，包括:

- 获取 Heapdump
- 用于处理 Heapdump 的工具
- 使用 **-Xverbose:gc** 来获取堆信息
- 环境变量和 Heapdump
- 文本（经典）Heapdump 文件格式
- 可移植堆转储 (PHD) 文件格式

您可以在此处查找信息: IBM SDK for Java 7 - 使用 Heapdump。

IBM WebSphere Real Time for Linux 的补充信息:

文本（经典）Heapdump 文件格式

文本或经典 Heapdump 是堆中所有对象实例的列表，包括对象类型、大小以及对象之间的引用。

头记录

头记录是包含版本信息字符串的单条记录。

```
// Version: <版本字符串包含 SDK 级别、平台和 JVM 构建级别>
```

示例:

```
// Version: J2RE 7.0 IBM J9 2.6 Linux x86-32 build 20101016_024574_lHdRSr
```

对象记录

对象记录是多条记录，每条记录用于堆中的每个对象实例，提供对象地址、大小、类型以及对象的引用。

```
<object address, in hexadecimal> [<length in bytes of object instance, in decimal>]
OBJ <object type> <class block reference, in hexadecimal>
<heap reference, in hexadecimal <heap reference, in hexadecimal> ...
```

对象地址和堆引用位于堆中，但是类块地址在堆外。将列出在对象实例中找到的所有引用，包括那些为空值的引用。对象类型是包括软件包或原始阵列的类名或类阵列类型，通过其标准的 JVM 类型签名显示，请参阅 第 53 页的『Java VM 类型签名』。对象记录也可包含其他的类块引用，通常在反映类实例的情况下。

示例:

类型为 java/lang/String 的对象实例，长度为 28 字节:

```
0x00436E90 [28] OBJ java/lang/String
```

java/lang/String 的类块地址，后跟对 char 阵列实例的引用:

```
0x415319D8 0x00436EB0
```

类型为 char 阵列的对象实例，长度为 44 字节:

```
0x00436EB0 [44] OBJ [C
```

char 阵列的类块地址:

```
0x41530F20
```

类型阵列为 java/util/Hashtable Entry 内部类的对象:

```
0x004380C0 [108] OBJ [Ljava/util/Hashtable$Entry;
```

类型为 java/util/Hashtable Entry 内部类的对象:

```
0x4158CD80 0x00000000 0x00000000 0x00000000 0x00000000 0x00421660 0x004381C0
0x00438130 0x00438160 0x00421618 0x00421690 0x00000000 0x00000000 0x00000000
0x00438178 0x004381A8 0x004381F0 0x00000000 0x004381D8 0x00000000 0x00438190
0x00000000 0x004216A8 0x00000000 0x00438130 [24] OBJ java/util/Hashtable$Entry
```

类块地址和堆引用，包括空引用:

```
0x4158CB88 0x004219B8 0x004341F0 0x00000000
```

类记录

类记录是多条记录，每条记录用于每个已装入的类，提供类块地址、大小、类型以及类的引用。

```
<class block address, in hexadecimal> [<length in bytes of class block, in decimal>]
CLS <class type>
<class block reference, in hexadecimal> <class block reference, in hexadecimal> ...
<heap reference, in hexadecimal> <heap reference, in hexadecimal>...
```

类块地址和类块引用在堆的外部，但是类记录还可以将引用包含到堆中，这通常适用于静态类数据成员。将列出在类块中找到的所有引用，包括那些为空值的引用。类类型是包括软件包或原始阵列的类名或类阵列类型，通过其标准的 JVM 类型签名显示，请参阅 第 53 页的『Java VM 类型签名』。

示例:

用于类 java/lang/Runnable 的长度为 32 字节的类块:

```
0x41532E68 [32] CLS java/lang/Runnable
```

对其他类块的引用和堆引用（包括空引用）:

```
0x4152F018 0x41532E68 0x00000000 0x00000000 0x00499790
```

用于类 java/lang/Math 的长度为 168 字节的类块:

```
0x00000000 0x004206A8 0x00420720 0x00420740 0x00420760 0x00420780 0x004207B0
0x00421208 0x00421270 0x00421290 0x004212B0 0x004213C8 0x00421458 0x00421478
0x00000000 0x41589DE0 0x00000000 0x4158B340 0x00000000 0x00000000 0x00000000
0x4158ACE8 0x00000000 0x4152F018 0x00000000 0x00000000 0x00000000
```

尾部记录 1

尾部记录 1 是包含记录计数的单条记录。

```
// Breakdown - Classes: <类记录计数（十进制）>,
Objects: <对象记录计数（十进制）>,
ObjectArrays: <对象数组记录计数（十进制）>,
PrimitiveArrays: <原语数组记录计数（十进制）>
```

示例:

```
// Breakdown - Classes: 321, Objects: 3718, ObjectArrays: 169,
PrimitiveArrays: 2141
```

尾部记录 2

尾部记录 2 是包含总数的单条记录。

```
// EOF:  Total 'Objects',Refs(null) :
<对象总计（十进制）>,
<引用总计（十进制）>
(空引用总计（十进制）>)
```

示例:

```
// EOF:  Total 'Objects',Refs(null) : 6349,23240(7282)
```

Java VM 类型签名

Java VM 类型签名是 Java 类型的缩写，如下表中所示:

Java VM 类型签名	Java 类型
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L <*标准类*> ;	<*标准类*>
[<*类型*>	<*类型*>[ ]（<*类型*> 的阵列）
（<*参数类型*>）<*返回类型*>	方法

使用系统转储和转储查看器

JVM 可以在可配置条件下生成本机系统转储，也被称为核心转储。系统转储通常很大。用于分析系统转储的大多数工具是特定于平台的。使用 **gdb** 工具可以分析 Linux 上的系统转储。

《IBM SDK for Java V7 用户指南》包含有关使用系统转储和转储查看器的有用指导，包括:

- 系统转储概述
- 系统转储缺省值
- 使用转储查看器
  - 使用 **jextract**
  - 使用转储查看器跟踪问题
  - **jdmpview** 中的可用命令
  - 示例会话
  - **jdmpview** 命令快速参考

您可以在此处查找信息: IBM SDK for Java 7 - 使用系统转储和转储查看器。

IBM WebSphere Real Time for Linux 的补充信息:

jdmpview 中的可用命令

jdmpview 是一种交互式的命令行工具，用于浏览来自 JVM 系统转储的信息以及执行各种分析功能。

info jitm

列出 AOT 和 JIT 编译的方法及其地址:

- 方法名称和特征符
- 方法起始地址

- 方法结束地址

有关所有其他命令选项，请参阅《IBM SDK for Java V7 用户指南》。

跟踪 Java 应用程序和 JVM

JVM 跟踪是 IBM WebSphere Real Time for Linux 中提供的一种跟踪工具，其对性能的影响最小。在大多数情况下，将以压缩二进制格式保存跟踪数据，这种格式的数据可使用提供的 Java 格式化程序进行格式化。

缺省情况下将启用跟踪，并且一小组跟踪点将转至内存缓冲区。您可以在运行时使用级别、组件、组名或个别跟踪点标识来启用跟踪点。

《IBM SDK for Java V7 用户指南》包含有关跟踪应用程序的详细信息，包括:

- 可跟踪哪些内容
- 跟踪点类型
- 缺省跟踪
- 记录跟踪数据
- 控制跟踪
- 跟踪 Java 应用程序
- 跟踪 Java 方法

在跟踪 IBM WebSphere Real Time for Linux 时，您必须在包含跟踪选项时正确的调用实时 JVM。例如，在指定跟踪选项时，请输入:

```
java -Xgcpolicy:metronome -Xtrace:<options>
```

您可以在此处查找 IBM SDK for Java V7 信息: 跟踪 Java 应用程序和 JVM。

JIT 和 AOT 问题确定

您可以使用命令行选项帮助诊断 JIT 和 AOT 编译器问题，以及调优性能。

尽管 IBM WebSphere Real Time for Linux 与 IBM SDK for Java V7 共享一些公共组件，但 JIT 和 AOT 的行为是不同的。本部分涵盖了针对 IBM WebSphere Real Time for Linux 上的 JIT 和 AOT 问题的故障诊断。

诊断 JIT 或 AOT 问题

有效字节码偶尔可能会编译为无效的本机代码，从而导致 Java 程序失败。通过确定 JIT 或 AOT 编译器是否出错，以及出错的位置（如果出错），您可以为 Java 服务团队提供有用的帮助。

关于此任务

要确定填充共享类缓存时编译的方法，请在 admincache 命令行上使用 **-Xaot:verbose** 选项。例如:

```
admincache -Xrealtime -Xaot:verbose -populate -aot my.jar -cp <My Class Path>
```

本部分描述了如何确定自己的问题是否与编译器相关。本部分还推荐了一些用于解决编译器相关问题的可能的变通方法和调试方法。

禁用 JIT 或 AOT 编译器:

如果您怀疑 JIT 或 AOT 编译器中出现问题，那么可以禁用编译，以查看问题是否仍然存在。如果问题仍然存在，那么可知问题不是由编译器引起的。

关于此任务

缺省情况下，启用 JIT 编译器。同时也会启用 AOT 编译器，但只有在启用了共享类后该编译器才处于活动状态。由于效率原因，不会编译 Java 应用程序中的所有方法。JVM 保留应用程序中每种方法的调用计数；每当调用并解释方法时，就会增加该方法的调用计数。当计数达到编译阈值时，就会在本机编译和执行该方法。

调用计数机制将方法的编译分布在应用程序的整个生命周期内，赋予最常使用的方法较高的优先级。一些不常使用的方法可能根本不会被编译。因此，当 Java 程序失败时，问题可能出在 JIT 或 AOT 编译器中，也可能出在 JVM 中的其他位置。

诊断故障的第一步就是确定问题的位置。要执行该操作，必须先在纯解释方式下运行 Java 程序（即禁用 JIT 和 AOT 编译器）。

过程

1. 除去命令行中的任何 **-Xjit** 和 **-Xaot** 选项（及伴随的参数）。
2. 使用 **-Xint** 命令行选项禁用 JIT 和 AOT 编译器。 由于性能原因，请勿在生产环境中使用 **-Xint** 选项。

下一步做什么

在禁用编译的情况下运行 Java 程序会导致以下一种情况:

- 故障仍然存在。问题不在 JIT 或 AOT 编译器中。在某些情况下，程序可能以不同的方式失败；不过，问题与编译器无关。
- 故障消失。问题很有可能在 JIT 或 AOT 编译器中。

  如果您未在使用共享类，那么出错的是 JIT 编译器。如果您正在使用共享类，那么必须在只启用 JIT 编译的情况下运行应用程序，从而确定出错的编译器。使用 **-Xnoaot** 选项（而不是 **-Xint** 选项）运行应用程序。这将导致以下一种情况:
  - 故障仍然存在。JIT 编译器中出现问题。您还可以使用 **-Xnojit** 选项（而不是 **-Xnoaot** 选项），以确保只有 JIT 编译器出错。
  - 故障消失。AOT 编译器中出现问题。

选择性地禁用 JIT 编译器:

如果 Java 程序故障指向 JIT 编译器问题，您可以尝试进一步缩小问题范围。

关于此任务

缺省情况下，JIT 编译器在各种优化级别对方法进行优化。不同的优化选项将根据不同方法的调用计数应用于这些方法。经常调用的方法会在较高级别上进行优化。通过更改 JIT 编译器参数，您可以控制对方法进行优化时采用的优化级别。您可以确定优化器是否发生故障，如果发生故障，可以确定发生问题的优化。

将 JIT 参数指定为用逗号分隔的列表，并附加到 **-Xjit** 选项。语法为: **-Xjit**:<*param1*>,<*param2*>=<*value*>。例如:

```
java -Xjit:verbose,optLevel=noOpt HelloWorld
```

将运行 HelloWorld 程序，对 JIT 启用详细输出，并使 JIT 生成本机代码而不执行任何优化。

按照以下步骤来确定编译器的哪个部分导致故障:

过程

1. 设置 JIT 参数 **count=0**，以便将编译阈值更改为零。此参数将导致先编译每个 Java 方法，然后再运行这些方法。 仅当诊断问题时才应使用 **count=0**，这是因为此参数会显著增加编译的方法数，包括编译那些并不经常使用的方法。额外的编译将使用更多计算资源，并导致应用程序速度减慢。 如果指定了 **count=0**，那么到达问题区域时，应用程序将立即失败。在某些情况下，采用 **count=1** 可以更可靠地重现故障。
2. 对 JIT 编译器参数添加 **disableInlining**。 **disableInlining** 禁止生成较大较复杂的代码。 如果问题不再发生，请在 Java 服务团队分析并解决编译器问题期间使用 **disableInlining** 作为变通方法。
3. 通过添加 **optLevel** 参数降低优化级别，并再次运行该程序，直到故障不再发生为止，或者使用“noOpt”级别。对于 JIT 编译器问题，请从“scorching”入手并依次使用下一个列表项。 这些优化级别按降序排列为:
   a. scorching
   b. veryHot
   c. hot
   d. warm
   e. cold
   f. noOpt

下一步做什么

如果其中一项设置导致故障消失，表明存在可以采用的变通方法。此变通方法可以在 Java 服务团队分析并解决编译器问题期间暂时采用。如果从 JIT 参数列表中除去 **disableInlining** 并不会导致故障重新出现，那么可以执行该操作以提高性能。遵循『查找失败的方法』中的指示信息，提高变通方法的性能。

如果在“noOpt”优化级别仍发生故障，那么作为变通方法，您必须禁用 JIT 编译器。

查找失败的方法:

确定将会导致 JIT 或 AOT 编译器编译方法时触发故障的最低优化级别后，可以确定哪一部分的 Java 程序在编译时将会引起故障。随后，您可以指示编译器将变通方法限制为某一特定方法、类或程序包，从而允许编译器正常编译程序的其余部分。对于 JIT 编译器故障，如果故障与 **-Xjit:optLevel=noOpt** 有关，那么您还可以指示编译器根本不编译导致故障的方法。

开始之前

如果您看到类似如下示例的错误输出，那么可以使用它来确定失败的方法:

```
Unhandled exception
Type=Segmentation error vmState=0x00000000
Target=2_30_20050520_01866_BHdSMr (Linux 2.4.21-27.0.2.EL)
CPU=s390x (2 logical CPUs) (0x7b6a8000 RAM)
```

```
J9Generic_Signal_Number=00000004 Signal_Number=0000000b Error_Value=4148bf20 Signal_Code=00000001
Handler1=00000100002ADB14 Handler2=00000100002F480C InaccessibleAddress=0000000000000000
gpr0=0000000000000006 gpr1=0000000000000006 gpr2=0000000000000000 gpr3=0000000000000006
gpr4=0000000000000001 gpr5=0000000080056808 gpr6=0000010002BCCA20 gpr7=0000000000000000
......
Compiled_method=java/security/AccessController.toArrayOfProtectionDomains([Ljava/lang/Object;
Ljava/security/AccessControlContext;)[Ljava/security/ProtectionDomain;
```

重要的行为:

vmState=0x00000000
指明失败的代码不是 JVM 运行时代码。

Module= or Module_base_address=
不出现在输出中（可能为空或 0），这是因为代码是由 JIT 编译的，位于任何 DDL 或库之外。

Compiled_method=
指明为其生成所编译代码的 Java 方法。

关于此任务

如果输出未指明失败的方法，那么可以按照以下步骤来确定失败的方法:

过程

1. 在对 **-Xjit** 或 **-Xaot** 选项添加了 JIT 参数 **verbose** 和 **vlog=**<*filename*> 的情况下运行 Java 程序。 利用这些参数，编译器会在名为 <*filename*>.<*date*>.<*time*>.<*pid*>的日志文件（也称为*界限文件*）中列出已编译的方法。一般的界限文件包含与所编译方法相对应的行，类似如下:

   ```
   + (hot) java/lang/Math.max(II)I @ 0x10C11DA4-0x10C11DDD
   ```

   编译器在接下来的步骤中忽略未以加号开头的行，您可以从文件中除去这些行。AOT 编译器编译的方法以 `+ (AOT cold)` 开头。为其从共享类高速缓存装入 AOT 代码的方法以 `+ (AOT load)` 开头。
2. 在指定了 JIT 或 AOT 参数 **limitFile=**(<*filename*>,<*m*>,<*n*>) 的情况下再次运行程序，其中 <*filename*> 是限制文件的路径，<*m*> 和 <*n*> 是行号，用于指示限制文件中第一个和最后一个应该编译的方法。 编译器只编译界限文件中 <*m*> 行到 <*n*> 行上所列出的方法。不会编译未列示在限制文件中的方法以及范围外部的行中列示的方法，并且不会装入共享数据高速缓存中这些方法的 AOT 代码。 如果程序不再失败，那么您在最后一次迭代中除去的一个或多个方法必然是导致故障的原因。
3. 可选: 如果您正在诊断 AOT 问题，那么可以使用同样的选项再次运行程序，以便从共享数据高速缓存中装入已编译的方法。您还可以添加 **–Xaot:scount=0** 选项，以确保首次调用方法时使用存储在共享数据高速缓存中的 AOT 编译的方法。 一些 AOT 编译故障只在从共享数据高速缓存中装入 AOT 编译的代码时发生。要帮助诊断这些问题，请使用 **–Xaot:scount=0** 选项，以确保首次调用该方法时使用存储在共享数据高速缓存中的 AOT 编译的方法，这样更便于重现问题。请注意，如果 **scount** 选项设为 0，它将强制装入 AOT 代码，并暂停所有等待执行该方法的应用程序线程。因而，这样的设置只能用于诊断目的。**–Xaot:scount=0** 选项可以产生更长的暂停时间。
4. 使用不同的 <*m*> 和 <*n*> 值按需多次重复该进程，以找到导致故障的必须编译的最少数量方法集。 通过每次将选定的行数减半，您可以对失败的方法执行对分搜索。一般来说，您可以将文件减少到一行。

下一步做什么

找到失败的方法后，可以仅针对失败的方法禁用 JIT 或 AOT 编译器。例如，如果 JIT 通过 **optLevel=hot** 进行编译时，java/lang/Math.max(II)I 方法导致程序失败，那么您可以通过以下选项运行程序:

```
-Xjit:{java/lang/Math.max(II)I}(optLevel=warm,count=0)
```

以便只在『warm』优化级别编译失败的方法，但正常编译所有其他方法。

如果 JIT 在『noOpt』优化级别编译方法时失败，您可以使用 **exclude**={<*method*>} 参数将其全部从编译中排除:

```
-Xjit:exclude={java/lang/Math.max(II)I}
```

如果编译或从共享数据高速缓存中装入 AOT 代码时某方法导致程序失败，那么可以使用 **exclude**={<*method*>} 参数将该方法从 AOT 编译和 AOT 装入中排除。

```
-Xaot:exclude={java/lang/Math.max(II)I}
```

AOT 方法只在『cold』优化级别进行编译。防止 AOT 编译或 AOT 装入是针对这些方法的最佳途径。

确定 JIT 编译故障:

对于 JIT 编译器故障，分析错误输出以确定当 JIT 编译器尝试编译方法时是否发生错误。

如果 JVM 崩溃，并且您可以看到故障发生在 JIT 库 (libj9jit26.so) 中，则 JIT 编译器可能在尝试编译某个方式时失败。

如果您看到类似如下示例的错误输出，那么可以使用它来确定失败的方法:

```
Unhandled exception
Type=Segmentation error vmState=0x00050000
Target=2_30_20051215_04381_BHdSMr (Linux 2.4.21-32.0.1.EL)
CPU=ppc64 (4 logical CPUs) (0xebf4e000 RAM)
J9Generic_Signal_Number=00000004 Signal_Number=0000000b Error_Value=00000000 Signal_Code=00000001
Handler1=0000007FE05645B8 Handler2=0000007FE0615C20
R0=E8D4001870C00001 R1=0000007FF49181E0 R2=0000007FE2FBCEE0 R3=0000007FF4E60D70
R4=E8D4001870C00000 R5=0000007FE2E02D30 R6=0000007FF4C0F188 R7=0000007FE2F8C290
......
Module=/home/test/sdk/jre/bin/libj9jit26.so
Module_base_address=0000007FE29A6000
......
Method_being_compiled=com/sun/tools/javac/comp/Attr.visitMethodDef(Lcom/sun/tools/javac/tree/
JCTree$JCMethodDecl;)
```

重要的行为:

vmState=0x00050000
指明 JIT 编译器正在编译代码。有关 vmState 代码编号的列表，请参阅《IBM SDK for Java V7 用户指南》、http://publib.boulder.ibm.com/infocenter/java7sdk/v7r0/topic/com.ibm.java.lnx.70.doc/diag/tools/javadump_tags_info.html中的 Javadump 标记表。

Module=/home/test/sdk/jre/bin/libj9jit26.so
指示错误发生在 JIT 编译器模块 libj9jit26.so 中。

Method_being_compiled=
指明正在编译的 Java 方法。

如果输出未指明失败的方法，那么使用 **verbose** 选项以及以下一些额外设置:

```
-Xjit:verbose={compileStart|compileEnd}
```

这些 **verbose** 设置会报告 JIT 开始编译方法的时间以及结束时间。如果 JIT 在特定方法上失败（即它开始编译，但在可以结束之前崩溃），那么使用 **exclude** 参数，将其排除在编译之外（请参阅第 56 页的『查找失败的方法』）。 如果排除该方法能够防止崩溃，那么您可以在服务团队纠正问题时采用一种变通方法。

短时间运行的应用程序的性能

IBM JIT 编译器针对服务器上通常使用的长时间运行的应用程序进行了调优。您可以使用 **-Xquickstart** 命令行选项来提高短时间运行的应用程序的性能，对于未将处理集中到几个方法中的应用程序而言尤其如此。

-Xquickstart 使 JIT 编译器在缺省情况下使用较低的优化级别，并编译较少的方法。更快速地执行少量编译可以减少应用程序的启动时间。当 AOT 编译器处于活动状态（同时启用了共享类和 AOT 编译）时，**-Xquickstart** 将导致所有选择编译的方法由 AOT 编译，这将缩短后续运行的启动时间。将 **-Xquickstart** 与那些包含使用了大量处理资源的方法的长时间运行应用程序配合使用时，可能会导致性能下降。**-Xquickstart** 的执行会根据将来发行版的变化而相应调整。

您还可以尝试通过调整 JIT 阈值（使用试错法）来改进启动时间。请参阅 第 55 页的『选择性地禁用 JIT 编译器』以获取更多信息。

空闲期间 JVM 的行为

通过使用 **-XsamplingExpirationTime** 选项关闭 JIT 采样线程，可以减少空闲 JVM 所使用的 CPU 周期。

JIT 采样线程对运行中的 Java 应用程序进行概要分析，以发现常用的方法。采样线程的内存和处理器使用情况可忽略不计，而且当 JVM 空闲时，概要分析的频率将自动降低。

在一些情况下，您可能希望空闲的 JVM 不占用 CPU 周期。要执行此操作，请指定 **-XsamplingExpirationTime**<*time*> 选项。将 <*time*> 设置为希望采样线程运行的秒数。请慎重使用该选项；关闭该选项后，您将不能重新激活采样线程。允许采样线程长时间运行，以确定重要的优化。

诊断收集器

“诊断收集器”收集针对问题事件的 Java 诊断文件。

收集 IBM 服务所需要的文件，可以减少解决报告的问题所需要的时间。《IBM SDK for Java V7 用户指南》包含有关使用“诊断收集器”的详细信息。

您可以在此处查找信息: IBM SDK for Java 7 - 诊断收集器。

垃圾收集器诊断数据

本部分描述了如何诊断垃圾收集问题。

《IBM SDK for Java V7 用户指南》包含有关诊断垃圾收集器问题的有用指导，包括:

- 详细垃圾回收日志记录
- 使用 **-Xtgc** 跟踪垃圾回收

您可以在此处查找信息: IBM SDK for Java 7 - 垃圾收集器诊断数据。

以下部分提供了有关 IBM WebSphere Real Time for Linux Metronome 垃圾回收器的补充信息。

Metronome 垃圾回收器故障诊断

通过使用命令行选项，您可以控制 Metronome 垃圾回收的频率、内存耗尽异常以及显式系统调用时的 Metronome 行为。

使用 verbose:gc 信息:

您可以结合使用 **-verbose:gc** 选项和 **-Xgc:verboseGCCycleTime=N** 选项以将关于 Metronome 垃圾回收器活动的信息写到控制台。并非标准 JVM 的 **-verbose:gc** 输出中的 XML 属性，并非所有都会予以创建，也并非全部适用于 Metronome 垃圾回收器的输出。

使用 **-verbose:gc** 选项以查看堆中的最小、最大和平均可用空间。这样，您便可以检查堆的活动级别和使用情况，然后在必要时调整这些值。**-verbose:gc** 选项用于将 Metronome 统计信息写到控制台。

-Xgc:verboseGCCycleTime=N 选项用于控制检索信息的频率。它决定转储摘要的时间（毫秒）。N 的缺省值是 1000 毫秒。周期时间不表示刚好在这一时间转储摘要，而是表示在符合该时间标准的最后一次垃圾回收事件过去时进行转储。对这些统计信息的收集和显示可能会增加 Metronome 垃圾回收器暂停次数，并且随着 N 变小，暂停次数可能会变多。

定量是 Metronome 垃圾回收器活动的单个时间段，会导致应用程序的中断或暂停时间。

verbose:gc 输出示例

输入:

```
java -Xgcpolicy:metronome -verbose:gc -Xgc:verboseGCCycleTime=N myApplication
```

触发垃圾回收时，将发生 `trigger start` 事件，后跟任意数量的 `heartbeat` 事件，然后在触发器得到满足时发生 `trigger end` 事件。该示例以 verbose:gc 输出方式显示已触发的垃圾回收周期:

```
<trigger-start id="25" timestamp="2011-07-12T09:32:04.503" />

<cycle-start id="26" type="global" contextid="26" timestamp="2011-07-12T09:32:04.503" intervalms="984.285" />

<gc-op id="27" type="heartbeat" contextid="26" timestamp="2011-07-12T09:32:05.209">
  <quanta quantumCount="321" quantumType="mark" minTimeMs="0.367" meanTimeMs="0.524" maxTimeMs="1.878"
    maxTimestampMs="598704.070" />
  <exclusiveaccess-info minTimeMs="0.006" meanTimeMs="0.062" maxTimeMs="0.147" />
  <free-mem type="heap" minBytes="99143592" meanBytes="114374153" maxBytes="134182032" />
  <thread-priority maxPriority="11" minPriority="11" />
</gc-op>

<gc-op id="28" type="heartbeat" contextid="26" timestamp="2011-07-12T09:32:05.458">
  <quanta quantumCount="115" quantumType="sweep" minTimeMs="0.430" meanTimeMs="0.471" maxTimeMs="0.511"
    maxTimestampMs="599475.654" />
  <exclusiveaccess-info minTimeMs="0.007" meanTimeMs="0.067" maxTimeMs="0.173" />
  <classunload-info classloadersunloaded=9 classesunloaded=156  />
  <references type="weak" cleared="660" />
  <free-mem type="heap" minBytes="24281568" meanBytes="55456028" maxBytes="87231320" />
  <thread-priority maxPriority="11" minPriority="11" />
</gc-op>
```

```
<gc-op id="29" type="syncgc" timems="136.945" contextid="26" timestamp="2011-07-12T09:32:06.046">
  <syncgc-info reason="out of memory" exclusiveaccessTimeMs="0.006" threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="21290752" bytesAfter="171963656" />
</gc-op>

<cycle-end id="30" type="global" contextid="26" timestamp="2011-07-12T09:32:06.046" />

<trigger-end id="31" timestamp="2011-07-12T09:32:06.046" />
```

可能会发生以下事件类型:

<trigger-start ...>
垃圾回收周期的开始（当已用内存量变得高于触发器阈值时）。缺省阈值为堆的 50%。intervalms 属性是上一个 trigger end 事件（id 为 -1）与此 trigger start 事件之间的时间间隔。

<trigger-end ...>
垃圾回收周期已成功将已用内存量降低至触发器阈值以下。如果垃圾回收周期已结束，但已用内存未降低到触发器阈值以下，那么将使用同一上下文标识开始新的垃圾回收周期。对于每个 trigger start 事件，都有一个具有同一上下文标识的匹配 trigger end 事件。intervalms 属性是上一个 trigger start 事件与当前 trigger end 事件之间的时间间隔。在这段时间内，一个或多个垃圾回收周期将会完成，直到已用内存降低到触发器阈值以下。

<gc-op id="28" type="heartbeat"...>
一个定期事件，它收集所涵盖时间内与所有垃圾回收定量有关的信息（关于内存和时间）。heartbeat 事件只能在一对匹配的 trigger start 与 trigger end 事件之间发生，即在活动垃圾回收周期正在进行期间发生。intervalms 属性是上一个 heartbeat 事件（id 为 -1）与该 heartbeat 事件之间的时间间隔。

<gc-op id="29" type="syncgc"...>
同步（非确定性）垃圾回收事件。请参阅第 62 页的『同步垃圾回收』

该示例中的 XML 标记具有以下含义:

<quanta ...>
脉动信号间隔期间定量暂停时间的摘要，包括暂停的长度（毫秒）。

<free-mem type="heap" ...>
脉动信号间隔期间可用堆空间量的摘要，在每个垃圾回收定量的末尾采样。

<classunload-info classloadersunloaded=9 classesunloaded=156 />
脉动信号间隔期间卸载的类装入器和类的数量。

<references type="weak" cleared="660 />
脉动信号间隔期间已清除的 Java 引用对象的数量和类型。

注:

- 如果两次脉动信号之间的时间间隔中只发生了一个垃圾回收定量，那么将仅在这一个定量的末尾对可用内存进行采样。因此，该脉动信号摘要中给出的最小、最大和平均量全都相等。
- 如果堆没有填满到足以要求执行垃圾回收活动的程度，那么两次 heartbeat 事件之间的时间间隔可能会比指定的周期时间长得多。例如，如果您的程序只是需要每几秒执行一次垃圾回收活动，那么您可能每几秒只会看到一次脉动信号。

- 时间间隔可能会由于以下原因而比指定的周期时间长得多: 垃圾回收对于没有填满到足以有理由进行垃圾回收活动的堆无工作。例如，如果您的程序只是需要每几秒执行一次垃圾回收活动，那么您可能每几秒只会看到一次脉动信号。

  如果发生诸如同步垃圾回收或优先级更改的事件，那么该事件以及任何暂挂事件（如 heartbeat 事件）的详细信息会立即生成为输出。
- 如果给定时间段内的最大垃圾回收定量过大，那么您可能希望使用 **-Xgc:targetUtilization** 选项来降低目标利用率。该操作给予垃圾回收器更多的工作时间。或者，您也可能希望使用 **-Xmx** 选项来增加堆大小。类似地，如果您的应用程序能够容忍比当前所报告的延迟更长的延迟，那么可以提高目标利用率或降低堆大小。
- 通过 **-Xverbosegclog:<file>** 选项可以将输出重定向到日志文件而非控制台；例如，**-Xverbosegclog:out** 将 **-verbose:gc** 输出写到文件 *out*。
- thread-priority 中所列的优先级是底层操作系统线程优先级，而非 Java 线程优先级。

同步垃圾回收

当发生同步（非确定性）垃圾回收时，也会向 **-verbose:gc** 日志中写入一个条目。该事件有三种可能的原因:

- 代码中有显式 System.gc() 调用。
- JVM 耗尽内存，然后执行同步垃圾回收以避免 OutOfMemoryError 情况。
- JVM 在持续垃圾回收期间关闭。JVM 无法取消此回收，因此同步完成此回收，然后退出。

System.gc() 条目的示例如下:

```
<gc-op id="9" type="syncgc" timems="12.92" contextid="8" timestamp=
"2011-07-12T09:41:40.808">
  <syncgc-info reason="system GC" totalBytesRequested="260" exclusiveaccessTimeMs="0.009"
  threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="22085440" bytesAfter="136023450" />
  <classunload-info classloadersunloaded="54" classesunloaded="234" />
  <references type="soft" cleared="21" dynamicThreshold="29" maxThreshold="32" />
  <references type="weak" cleared="523" />
  <finalization enqueued="124" />
</gc-op>
```

因 JVM 关闭而产生的同步垃圾回收条目的示例如下:

```
<gc-op id="24" type="syncgc" timems="6.439" contextid="19" timestamp="2011-07-12T09:43:14.524">
  <syncgc-info reason="VM shut down" exclusiveaccessTimeMs="0.009" threadPriority="11" />
  <free-mem-delta type="heap" bytesBefore="56182430" bytesAfter="151356238" />
  <classunload-info classloadersunloaded="14" classesunloaded="276" />
  <references type="soft" cleared="154" dynamicThreshold="29" maxThreshold="32" />
  <references type="weak" cleared="53" />
  <finalization enqueued="34" />
</gc-op>
```

该示例中的 XML 标记和属性具有以下含义:

<gc-op id="9" type="syncgc" timems="6.439" ...
: 该行指示事件类型是同步垃圾回收。timems 属性是同步垃圾回收的持续时间（毫秒）。

<syncgc-info reason="..."/>
: 同步垃圾回收的原因。

<free-mem-delta.../>
同步垃圾回收前后的可用 Java 堆内存（字节）。

<finalization .../>
等待最终化的对象的数量。

<classunload-info .../>
脉动信号间隔期间卸载的类装入器和类的数量。

<references type="weak" cleared="53" .../>
脉动信号间隔期间已清除的 Java 引用对象的数量和类型。

因内存耗尽情况或 JVM 关闭而进行的同步垃圾回收只能在垃圾回收器处于活动状态的情况下发生。这之前必须先发生 `trigger start` 事件，尽管两者不必前后紧密相连。某些 heartbeat 事件可能在 `trigger start` 事件和 `synchgc` 事件之间发生。 `System.gc()` 所引起的同步垃圾回收可随时发生。

跟踪所有 GC 定量

对于单独 GC 定量，可通过启用 `GlobalGCStart` 和 `GlobalGCEnd` 跟踪点来进行跟踪。这些跟踪点在所有 Metronome 垃圾回收器活动（包括同步垃圾回收）的开始和结束时生成。这些跟踪点的输出类似于:

```
03:44:35.281 0x833cd00 j9mm.52 - GlobalGC start: globalcount=3

03:44:35.284 0x833cd00 j9mm.91 - GlobalGC end: workstackoverflow=0 overflowcount=0
```

内存耗尽条目

当堆耗尽可用空间时，会在抛出 OutOfMemoryError 异常之前向 **-verbose:gc** 日志中写入条目。该输出的示例如下:

```
<out-of-memory id="71" timestamp="2011-07-12T10:21:50.135" memorySpaceName="Metronome"
  memorySpaceAddress="0806DFDC"/>
```

缺省情况下，会因 OutOfMemoryError 异常而生成 Java 转储。该转储包含与程序所用的内存有关的信息。

```
NULL
1STSEGTOTAL    Total memory:           4066080 (0x003E0B20)
1STSEGINUSE    Total memory in use:    3919440 (0x003BCE50)
1STSEGFREE     Total memory free:       146640 (0x00023CD0)
```

内存耗尽情况下的 Metronome 垃圾回收器行为:

缺省情况下，Metronome 垃圾回收器在 JVM 耗尽内存时触发不受限的非确定性垃圾回收。要避免非确定性行为，请使用 **-Xgc:noSynchronousGCOnOOM** 选项以在 JVM 耗尽内存时抛出 OutOfMemoryError。

缺省的不受限回收将一直运行，直到在一次操作中回收完所有可能的垃圾为止。所需暂停时间通常比普通 Metronome 递增定量长许多毫秒。

相关信息:

使用 -Xverbose:gc 来分析同步垃圾回收

显式 System.gc() 调用时的 Metronome 垃圾回收器行为:

如果垃圾回收周期正在进行中，那么在调用 System.gc() 时 Metronome 垃圾回收器会以同步方式完成该周期。如果未在进行任何垃圾回收，那么在调用 System.gc() 时将执行完全同步周期。使用 System.gc() 可通过受控方式来清理堆。这是非确定性操作，因为它在返回之前执行完全垃圾回收。

某些应用程序会调用具有 System.gc() 调用的供应商软件，其中不允许创建这些非确定性延迟。要禁用所有 System.gc() 调用，请使用 **-Xdisableexplicitgc** 选项。

System.gc() 调用的详细垃圾回收输出具有“系统垃圾回收”原因，而可能具有较长的持续时间:

```
<gc-op id="9" type="syncgc" timems="6.439" contextid="8" timestamp="2011-07-12T09:41:40.808">
  <syncgc-info reason="VM shut down" exclusiveaccessTimeMs="0.009" threadPriority="11"/>
  <free-mem-delta type="heap" bytesBefore="126082300" bytesAfter="156085440"/>
  <classunload-info classloadersunloaded="14" classesunloaded="276"/>
  <references type="soft" cleared="154" dynamicThreshold="29" maxThreshold="32"/>
  <references type="weak" cleared="53"/>
  <finalization enqueued="34"/>
</gc-op>
```

共享类诊断数据

了解如何诊断可能出现的问题，将有助于使用共享类方式。

《IBM SDK for Java V7 用户指南》包含有关诊断共享类问题的有用指导，包括:

- 部署共享类
- 处理运行时字节码修改
- 了解动态更新
- 使用 Java Helper API
- 了解共享类诊断输出
- 调试有关共享类的问题

您可以在此处查找信息: IBM SDK for Java 7 - 共享类诊断数据。

使用 JVMTI

JVMTI 是一个双路接口，支持 JVM 与本机代理程序之间进行通信。它取代了 JVMDI 和 JVMPI 接口。

JVMTI 支持第三方为 JVM 开发调试、概要分析和监视工具。此接口包含用于通知 JVM 所需的各类信息的代理程序机制。此接口还提供了接收相关通知的方式。任何时候都可以将几个代理程序连接至 JVM。

《IBM SDK for Java V7 用户指南》包含有关使用 JVMTI 的详细信息，包括有关 IBM 对 JVMTI 的扩展的 API 参考部分。

您可以在此处查找信息: IBM SDK for Java 7 - 使用 JVMTI。

使用 Diagnostic Tool Framework for Java

Diagnostic Tool Framework for Java (DTFJ) 是 IBM 提供的 Java 应用程序编程接口 (API)，用于为构建 Java 诊断工具提供支持。DTFJ 可以处理来自系统转储或 Javadump 的数据。

《IBM SDK for Java V7 用户指南》包含有关 DTFJ 的详细信息。跟随此链接：使用 Diagnostic Tool Framework for Java

第 10 章 参考

这一组主题列出了可用于 WebSphere Real Time for Linux 的选项和类库

命令行选项

在您启动 Java 时，可以在命令行中指定选项。最常用的选项已选作缺省选项。

指定 Java 选项和系统属性

有三种方法可指定 Java 属性和系统属性。

关于此任务

您可使用这些方法指定 Java 选项和系统属性。按照优先顺序，这些方法依次为:

1. 在命令行上指定选项或属性。例如:

   ```
   java -Dmysysprop1=tcpip -Dmysysprop2=wait -Xdisablejavadump MyJavaClass
   ```

2. 创建包含这些选项的文件，并使用 **-Xoptionsfile=**<*filename*> 选项在命令行上指定该文件。

   在该选项文件中，在新的一行中指定每个选项；如果需要单个选项跨多个行，那么可以使用“\”字符作为续行符。使用“#”字符来定义注释行。不能在选项文件中指定 **-classpath**。以下是选项文件的示例:

   ```
   #My options file
   -X<option1>
   -X<option2>=\
   <value1>,\
   <value2>
   -D<sysprop1>=<value1>
   ```

3. 创建包含这些选项的名为 **IBM_JAVA_OPTIONS** 的环境变量。例如:

   ```
   export IBM_JAVA_OPTIONS="-Dmysysprop1=tcpip -Dmysysprop2=wait -Xdisablejavadump"
   ```

您在命令行中指定的最后一个选项的优先顺序高于第一个选项。例如，如果您指定选项 **-Xint -Xjit myClass**，那么 **-Xjit** 的优先顺序高于 **-Xint**。

系统属性

为应用程序提供了系统属性，帮助提供有关运行时环境的信息。

com.ibm.jvm.realtime

该属性使 Java 应用程序能够确定是否在 WebSphere Real Time for Linux 环境下运行。

如果您的应用程序正在 IBM WebSphere Real Time for RT Linux 运行时环境下运行，且使用 **-Xrealtime** 选项启动，那么 **com.ibm.jvm.realtime** 属性的值为“hard”。

如果您的应用程序正在 IBM WebSphere Real Time for RT Linux 运行时环境下运行，但未使用 **-Xrealtime** 选项启动，那么不会设置 **com.ibm.jvm.realtime** 属性。

如果应用程序在 IBM WebSphere Real Time 运行时环境下运行，那么 **com.ibm.jvm.realtime** 属性的值为“soft”。

标准选项

标准选项的定义。

-agentlib:<*libname*>**[=**<*options*>**]**
装入本机代理程序库 <*libname*>；例如 **-agentlib:hprof**。有关更多信息，请在命令行上指定 **-agentlib:jdwp=help** 和 **-agentlib:hprof=help**。

-agentpath:*libname***[=**<*options*>**]**
以完整路径名装入本机代理程序库。

-assert
打印有关与断言相关的选项的帮助。

-cp 或 -classpath <*由 : 隔开的目录和 .zip 或 .jar 文件*>
设置应用程序类和资源的搜索路径。如果不使用 **-classpath** 和 **-cp**，也不设置 **CLASSPATH**，那么缺省情况下，用户的类路径是当前目录（.）。

-D<property_name>=<*value*>
设置系统属性。

-help 或 -?
打印使用消息。

-javaagent:<*jarpath*>**[=**<*options*>**]**
装入 Java 编程语言代理程序。有关更多信息，请参阅 java.lang.instrument API 文档。

-jre-restrict-search
在版本搜索中包含用户专用的 JRE。

-no-jre-restrict-search
在版本搜索中排除用户专用的 JRE。

-showversion
打印产品版本并继续。

-verbose:[*class***,***gc***,***dynload***,***sizes***,***stack***,***jni***]**
启用冗余输出。

-verbose:class
为装入的每个类的 stderr 写入一个条目。

-verbose:gc
请参阅第 60 页的『使用 verbose:gc 信息』。

-verbose:dynload
在 JVM 装入每个类时提供详细信息，包括:
- 类名和程序包
- 对于 .jar 文件中的类文件，.jar 的名称和目录路径
- 类大小和装入类所用时间的详细信息

数据将写入 stderr。输出示例如下所示:

```
<Loaded java/lang/String from /myjdk/sdk/jre/lib/i386/
softrealtime/jclSC160/vm.jar>
<Class size 17258; ROM size 21080; debug size 0>
<Read time 27368 usec; Load time 782 usec; Translate time 927 usec>
```

注: **-verbose:dynload** 输出中不列出从共享类高速缓存装入的类。 使用 **-verbose:class** 获取有关这些类的信息。

-verbose:sizes
向 stderr 写入信息，描述用于 JVM 中堆栈和堆的内存量

-verbose:stack
向 stderr 写入信息，描述 Java 和 C 堆栈使用情况。

-verbose:jni
向 stderr 写入信息，描述由应用程序和 JVM 调用的 JNI 服务。

-version
打印输出非实时方式的版本信息。

-version:<*value*>
需要运行指定版本。

-X 打印关于非标准选项的帮助。

非标准选项

带有 **-X** 前缀的选项是非标准选项；它们可能发生更改，而不另行通知。

《IBM SDK for Java V7 用户指南》包含有关非标准选项的详细信息。您可在以下地址查找该信息: IBM SDK for Java 7 - 命令行选项。

以下部分提供有关 IBM WebSphere Real Time for Linux 的补充信息。

Metronome 垃圾回收器选项

Metronome 垃圾回收器选项的定义。

-Xgc:synchronousGCOnOOM | -Xgc:nosynchronousGCOnOOM
执行垃圾回收的一种情况是堆内存耗尽。如果堆中没有更多的可用空间，使用 **-Xgc:synchronousGCOnOOM** 可停止应用程序，同时垃圾回收除去未使用的对象。如果可用空间再次耗尽，考虑降低目标利用率以为垃圾回收的完成提供更多时间。设置 **-Xgc:nosynchronousGCOnOOM** 意为当堆内存已满时，应用程序停止，并发出内存耗尽消息。缺省值为 **-Xgc:synchronousGCOnOOM**。

-Xnoclassgc
禁用类垃圾回收。此选项关闭与 JVM 不再使用的 Java 类关联的存储器的垃圾回收。缺省行为是 **-Xnoclassgc**。

-Xgc:targetPauseTime=*N*
设置垃圾回收暂停时间，其中 *N* 是时间（单位为毫秒，ms）。指定该选项后，GC 运行时的暂停时间不会超出指定的值。如果未指定该选项，那么缺省暂停时间设置为 3 毫秒。例如，使用 **-Xgc:targetPauseTime=20** 运行会导致 GC 运行期间 GC 暂停时间不超过 20 毫秒。

-Xgc:targetUtilization=*N*

将应用程序利用率设置为 N%；垃圾回收器尝试最多使用每个时间间隔的 (100-N)%。合理值范围为 50-80%。低分配率的应用程序可能以 90% 运行。缺省值为 70%。

本示例显示堆内存最大大小为 30 MB。垃圾回收器尝试使用每个时间间隔的 25%，原因是应用程序的目标利用率为 75%。

```
java -Xgcpolicy:metronome -Xmx30m -Xgc:targetUtilization=75 Test
```

-Xgc:threads=*N*

指定要运行的 GC 线程数目。缺省值为可用于进程的处理器内核数目。您可指定的最大值为可用于操作系统的处理器数目。

-Xgc:verboseGCCycleTime=N

N 为应转储摘要信息的时间（单位为毫秒）。

注: 周期时间不表示刚好在这一时间转储摘要信息，而是表示在符合该时间标准的最后一次垃圾回收事件过去时进行转储。

-Xmx<*size*>

指定 Java 堆大小。与其他垃圾回收策略不同，实时 Metronome GC 并不支持堆扩展。没有初始或最大堆大小选项。您只能指定最大堆大小。

JVM 的缺省设置

当 JVM 的运行环境未发生更改时，缺省设置会应用于实时 JVM 中。下面显示了常用设置以供参考。

在 JVM 启动时可以使用环境变量或命令行参数来更改缺省设置。下表显示了部分常用 JVM 设置。最后一列指示您可以如何更改行为，其中的如下关键字适用:

- **e** - 仅环境变量控制的设置
- **c** - 仅命令行参数控制的设置
- **ec** - 环境变量和命令行参数都可以控制的设置，其中命令行参数优先。

提供的信息仅供快速参考，并不全面。

JVM 设置	缺省值	影响设置的原因
Javadumps	已启用	ec
Javadumps on out of memory	已启用	ec
Heapdumps	已禁用	ec
Heapdumps on out of memory	已启用	ec
Sysdumps	已启用	ec
Where dump files are produced	Current®目录	ec
Verbose output	已禁用	c
Boot classpath search	已禁用	c
JNI checks	已禁用	c
Remote debugging	已禁用	c
Strict conformancy checks	已禁用	c
Quickstart	已禁用	c

JVM 设置	缺省值	影响设置的原因
Remote debug info server	已禁用	c
Reduced signalling	已禁用	c
Signal handler chaining	已启用	c
Classpath	未设置	ec
Class data sharing	已禁用	c
Accessibility support	已启用	e
JIT compiler	已启用	ec
AOT compiler (AOT is not used by the JVM unless shared classes are also enabled)	已启用	c
JIT debug options	已禁用	c
Java2D max size of fonts with algorithmic bold	14 磅	e
Java2D use rendered bitmaps in scalable fonts	已启用	e
Java2D freetype font rasterizing	已启用	e
Java2D use AWT fonts	已禁用	e
Default locale	无	e
Time to wait before starting plug-in	0	e
Temporary directory	/tmp	e
Plug-in redirection	无	e
IM switching	已禁用	e
IM modifiers	已禁用	e
Thread model	不适用	e
Initial stack size for Java Threads 32-bit. Use: **-Xiss<size>**	2 KB	c
Maximum stack size for Java Threads 32-bit. Use: **-Xss<size>**	256 KB	c
Stack size for OS Threads 32-bit. Use **-Xmso<size>**	256 KB	c
Initial heap size. Use **-Xms<size>**	64 MB	c
Maximum Java heap size. Use **-Xmx<size>**	可用内存的一半，最小为 16 MB，最大为 512 MB	c
应用程序的目标时间间隔利用率。垃圾回收器会尝试使用剩余部分。使用 **-Xgc:targetUtilization=<percentage>**	70%	c
要运行的垃圾回收器线程的数量。使用 **-Xgc:threads=<value>**	进程可用的处理器核心数量。	c
在 **-Xrealtime** 方式下可以分配到作用域内存的最大内容量。使用 **-Xgc:scopedMemoryMaximumSize=<size>**。	8 MB	c
在 **-Xrealtime** 方式下设置永久内存区域的大小。使用 **-Xgc:immortalMemorySize=<size>**	16 MB	c

注: “可用内存”可以是实际（物理）内存的数量，或者 **RLIMIT_AS** 值，取其中最小的值。

声明

本信息是为在美国提供的产品和服务编写的。IBM 可能在其他国家或地区不提供本文档中讨论的产品、服务或功能特性。有关您当前所在区域的产品和服务的信息，请向您当地的 IBM 代理咨询。任何对 IBM 产品、程序或服务的引用并非意在明示或暗示只能使用 IBM 的产品、程序或服务。只要不侵犯 IBM 的知识产 权，任何同等功能的产品、程序或服务，都可以代替 IBM 产品、程序或服务。但是，评估和验证任何非 IBM 产品、程序或服务，则由用户自行负责。

IBM 公司可能已拥有或正在申请与本文档中主题有关的各项专利。提供本文档并未授予用户使用这些专利的任何许可。您可以用书面方式将许可查询寄往:

IBM Director of Licensing IBM Corporation North Castle DriveArmonk, NY 10504-1785 U.S.A.

有关双字节 (DBCS) 信息的许可证查询，请与您所在国家或地区的 IBM 知识产权部门联系，或用书面方式将查询寄往:

Intellectual Property LicensingLegal and Intellectual Property LawIBM Japan Ltd.19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

本条款不适用英国或任何这样的条款与当地法律不一致的国家或地区:

INTERNATIONAL BUSINESS MACHINES CORPORATION“按现状”提供本出版物，不附有任何种类的（无论是明示的还是暗含的）保证，包括但不限于暗含的有关非侵权、适销和适用于某特定用途的保证。某些国家或地区在某些交易中不允许免除明示或暗含的保证。因此本条款可能不适用于您。

本信息中可能包含技术方面不够准确的地方或印刷错误。此处的信息将定期更改；这些更改将编入本资料的新版本中。IBM 可以随时对本信息中描述的产品和/或程序进行改进和/或更改，而不另行通知。

本信息中对非 IBM Web 站点的任何引用都只是为了方便起见才提供的，不以任何方式充当对那些 Web 站点的保证。那些 Web 站点中的资料不是本 IBM 产品资料的一部分，使用那些 Web 站点带来的风险将由您自行承担。

IBM 可以按它认为适当的任何方式使用或分发您所提供的任何信息而无须对您承担任何责任。

本程序的被许可方如果要了解有关程序的信息以达到如下目的: (i) 允许在独立创建的程序和其他程序（包括本程序）之间进行信息交换，以及 (ii) 允许对已经交换的信息进行相互使用，请与下列地址联系:

- JIMMAIL@uk.ibm.com [Hursley Java Technology Center (JTC) contact]

只要遵守适当的条件和条款，包括某些情形下的一定数量的付费，都可获得这方面的信息。

本资料中描述的许可程序及其所有可用的许可资料均由 IBM 依据 IBM 客户协议、IBM 国际程序许可协议或任何同等协议中的条款提供。

此处包含的任何性能数据都是在受控环境中测得的。因此，在其他操作环境中获得的数据可能会有明显的不同。有些测量可能是在开发集的系统上进行的，因此不保证与一般可用系统上进行的测量结果相同。此外，有些测量是通过推算而估计的，实际结果可能会有差异。本文档的用户应当验证其特定环境的适用数据。

涉及非 IBM 产品的信息可从这些产品的供应商、其出版说明或其他可公开获得的资料中获取。IBM 没有对这些产品进行测试，也无法确认其性能的精确性、兼容性或任何其他关于非 IBM 产品的声明。有关非 IBM 产品性能的问题应当向这些产品的供应商提出。

隐私策略注意事项

IBM 软件产品（包括软件即服务解决方案（“软件产品”））可能会使用 cookie 或其他技术收集产品使用信息，以帮助改善最终用户体验，定制与最终用户的交互或用于其他目的。在许多情况下，软件产品不会收集个人可标识信息 (PII)。我们的某些软件产品可以帮助您收集个人可标识信息 (PII)。如果本软件产品使用 cookie 收集个人可标识信息 (PII)，下面列出了有关本产品的具体 cookie 使用信息。

此软件产品不使用 cookie 或其他技术来收集个人可标识信息。

如果针对该软件产品部署的配置使您作为客户能够通过 cookie 和其他技术从最终用户处收集个人可标识信息，那么您应向自己的法律顾问了解有关适用于此类数据收集的任何法律，包括声明和同意的要求。

要获取有关针对这些目的使用各种技术（包括 cookies）的更多信息，请参阅 (i) IBM 的隐私策略：http://www.ibm.com/privacy；(ii) IBM 的在线隐私声明：http://www.ibm.com/privacy/details（尤其是题为“Cookies, Web Beacons and Other Technologies”的部分）；以及 (iii)“IBM Software Products and Software-as-a-Service Privacy Statement”：http://www.ibm.com/software/info/product-privacy。

商标

IBM、IBM 徽标和 ibm.com 是 International Business Machines Corporation 在美国和/或其他国家或地区的商标或注册商标。如果这些术语和其他 IBM 已注册商标的术语在本信息中首次出现时都使用商标符号（® 或 ™）标记，那么这些符号表示在本信息发布时由 IBM 在美国注册或拥有的普通法商标。这些商标也可能是在其他国家或地区的注册商标或普通法商标。 IBM 商标的最新列表在 Web 页面 http://www.ibm.com/legal/copytrade.shtml 的“Copyright and trademark information”部分中提供。

Adobe、Adobe 徽标、PostScript 和 PostScript 徽标是 Adobe Systems Incorporated 在美国和/或其他国家或地区的注册商标或商标。

Intel 和 Itanium 是 Intel Corporation 或其子公司在美国和其他国家或地区的商标。

Linux 是 Linus Torvalds 在美国和/或其他国家或地区的商标。

Java 和所有基于 Java 的商标和徽标是 Oracle 和/或其子公司的商标或注册商标。

其他公司、产品或服务名称可能是其他公司的商标或服务标记。

索引

Printed in China