Java SE 6 对性能进行了深入研究，使用增强的工具管理和监视应用程序并且诊断常见问题。这些改进包括：

* 监视和管理 API 增强
* 正式支持增强的图形监视工具 JConsole
* 提供增强的 Java 虚拟机（JVM）测试工具

本文介绍了 Java SE 平台的监视和管理基本知识，并详细介绍了最新版本中的性能监视和管理增强。还介绍了 Java SE 6 平台提供的诊断和故障排除工具。

要从本文获益，您应当非常熟悉较早 Java SE 版本中引入的监视和管理功能。更多背景信息，请参阅 [参考资料](http://www.ibm.com/developerworks/cn/java/j-java6perfmon/index.html#resources)。

**监视和管理 API**

Java SE 5 中引入的 java.lang.management 包定义了 9 个 MBean，称为*平台 MBean* 或 *MXBean*（请参阅 [参考资料](http://www.ibm.com/developerworks/cn/java/j-java6perfmon/index.html#resources)）。每个 MXBean 封装了 JVM 的一个单独的功能区。从 Java SE 5 开始，JVM 包含了一个内置的 MBean 服务器，称为*平台 MBean 服务器*。MBeans 位于其中并由其管理。表 1 列出了 Java 平台中的 9 个 MXBeans：

**表 1. 平台 MBean**

|  |  |
| --- | --- |
| **管理接口** | **托管的资源** |
| ClassLoadingMXBean | 类装载器 |
| CompilationMXBean | 编译器 |
| MemoryMXBean | 内存 |
| ThreadMXBean | 线程 |
| RuntimeMXBean | 运行时 |
| OperatingSystemMXBean | 操作系统 |
| GarbageCollectorMXBean | 垃圾收集器 |
| MemoryManagerMXBean | 内存管理器 |
| MemoryPoolMXBean | 内存池 |

任何应用程序都可获得并使用 JVM 提供的平台 MBean，方法是获得目标 bean 的实例并调用合适的方法。MXBean 可用来监视本地和远程 JVM 的行为并获得相关信息。

平台 MBean 可提供对信息的访问，例如装载的类的数量、JVM 正常运行时间、内存消耗量、正在运行的线程的数量，以及线程竞争统计信息。

您可以使用以下两种方式之一监视和管理 JVM 资源：

* 直接访问 MXBean 接口
* 使用 MBeanServer 接口进行间接访问

**使用 MXBean 接口直接访问**

您可以从一个静态工厂方法获得一个 MXBean 实例，该方法可以使您直接访问本地运行的 JVM 的 MXBean 接口。ManagementFactory 类提供静态工厂方法获得 MXBean。清单 1 演示了如何使用该工厂获得 RuntimeMXBean 并获得其中一个标准属性 VmVendor 的值：

**清单 1. 直接访问 MXBean**

|  |
| --- |
| RuntimeMXBean mxbean = ManagementFactory.getRuntimeMXBean();  // Get the standard attribute "VmVendor"  String vendor = mxbean.getVmVendor(); |

**使用 MBeanServer 接口进行间接访问**

平台 MBeanServer 接口使用 MXBeanServerConnection，使您能够连接到远程 JVM 并访问运行在这些平台上的 MXBean。您可以使用ManagementFactory 类的 getPlatformMBeanServer 方法访问平台 MBean 服务器。清单 2 演示了如何获得运行在远程 JVM 上的RuntimeMXBean 并获得 VmVendor 属性的值：

**清单 2. 间接访问 MXBean**

|  |
| --- |
| MBeanServerConnection serverConn;  try {  //connect to a remote VM using JMX RMI  JMXServiceURL url = new JMXServiceURL( "service:jmx:rmi:///jndi/rmi://<addr>");  JMXConnector jmxConnector = JMXConnectorFactory.connect(url);  serverConn = jmxConnector.getMBeanServerConnection();  ObjectName objName = new  ObjectName(ManagementFactory.RUNTIME\_MXBEAN\_NAME);  // Get standard attribute "VmVendor"  String vendor =  (String) serverConn.getAttribute(objName, "VmVendor");  } catch (...) { } |

有关 MXBeans 和 java.lang.management API 的更详细信息，请参阅 [参考资料](http://www.ibm.com/developerworks/cn/java/j-java6perfmon/index.html#resources) 。

**Java SE 6 中的 API 增强**

Java SE 5 引入了 java.util.concurrent.locks 包，它为锁定和等待条件提供了一种框架。这种框架有别于 Java 的内置同步支持并允许更加灵活地使用锁。

Java SE 6 为 java.lang.management 包的 java.util.concurrent.locks 添加了支持。这包括可提供锁信息的新类以及对ThreadInfo、ThreadMXBean 和 OperatingSystemMXBean 接口的增强。

Java SE 6 引入了两个新类：

* LockInfo 包含有关锁的信息。
* MonitorInfo 扩展了 LockInfo 并包含有关对象监视锁的信息。

ThreadInfo 类利用了这些新的对象以及引入的三种新方法：

* getLockInfo() 返回 LockInfo 对象，给定线程将被阻塞以等待该对象。
* getLockedMonitors() 返回当前被给定线程锁定的 MonitorInfo 对象。
* getLockedSynchronizers() 返回 LockInfo 对象，提供当前由给定线程锁定的可拥有的同步程序。

在 Java SE 5 中，ThreadMXBean.getThreadInfo 方法只报告线程正在等待获取的对象监视器或是被阻塞的监视器。Java SE 6 对这些方法进行了增强，从而可以报告线程正在等待获取的 AbstractOwnableSynchronizer。

4 个新的方法被添加到 ThreadMXBean 接口中：

* isObjectMonitorUsageSupported() 将测试虚拟机是否支持对对象监视器的使用情况进行监视。
* isSynchronizerUsageSupported() 测试虚拟机是否对可拥有的同步程序使用情况进行监视。
* findDeadlockedThreads() 返回处于死锁状态的线程的 ID 数组。死锁线程被阻塞，防止进入对象监视器或同步程序。
* dumpAllThreads() 为所有活动线程返回堆栈跟踪和同步信息。

最后，OperatingSystemMXBean 接口将进行更新，以包括 getSystemLoadAverage() 方法，它可以返回前一分钟的系统负载平均值。

除了提供编程支持以外，Java SE 6 还包括了一些诊断和故障排除工具，可用来检测问题并监视 JVM 资源使用情况。接下来两节将介绍并演示其中几个可用的诊断工具。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-java6perfmon/index.html#ibm-pcon)

**Java 监视和管理控制台（JConsole）**

Java SE 6 提供了对 JConsole 的正式支持，这是 Java 5 SE 中引入的监视和管理控制台。JConsole 使您能够在运行时监视各种 JVM 资源统计信息。这种特性特别适用于检测死锁、锁竞争、内存泄漏和循环线程。它可以连接到一个本地或远程 JVM 并可用来进行监视：

* 线程状态（包括相关的锁）
* 内存使用情况
* 垃圾收集
* 运行时信息
* JVM 信息

以下小节将介绍 Java SE 6 对 JConsole 做出的增强。有关如何启动和使用 JConsole 的更多信息，请参阅 [参考资料](http://www.ibm.com/developerworks/cn/java/j-java6perfmon/index.html#resources)。

**Attach API 支持**

从 Java SE 6 开始，JConsole 实现了新的 Attach API。该 API 由两个包组成 — com.sun.tools.attach 和com.sun.tools.attach.spi — 可将应用程序的实现过程动态连接到目标虚拟机并在 JVM 内运行代理。

过去，对于要使用 JConsole 进行监视的应用程序，要求使用 -Dcom.sun.management.jmxremote 选项启动；现在，应用程序不再需要使用这个选项启动。对动态连接的支持使 JConsole 能够监视任何支持 Attach API 的应用程序。JConsole 在启动时将自动检测兼容的应用程序。

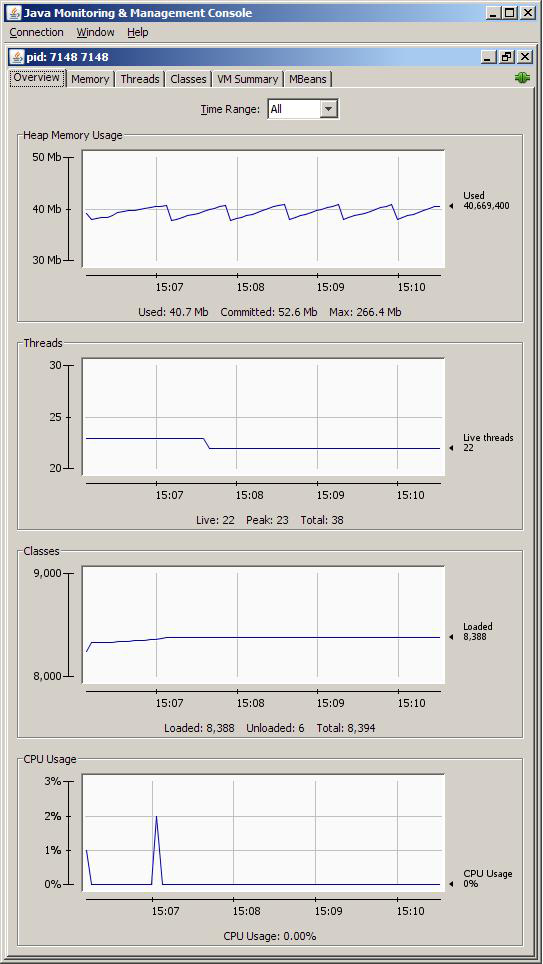
**增强的 UI 和 MBean 表示**

Java SE 6 对 JConsole 进行了更新，使它具有类似于 Windows® 操作系统或 GNOME 桌面的感观（取决于所运行的平台）。后文中显示的屏幕截图使用的是 Windows XP 并展示了与之前版本不同的 UI 特性。

一旦启动并与应用程序相关联，将显示由 6 个选项卡组成的 JConsole 视图。每个选项卡表示一个不同的 JVM 资源或一组资源：

* Overview
* Memory
* Threads
* Classes
* VM Summary
* MBeans

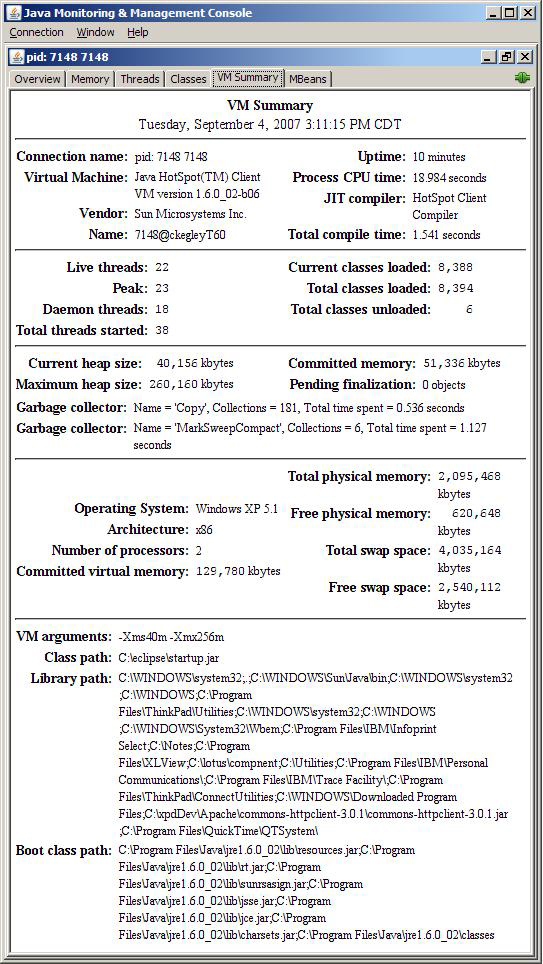
Overview 选项卡以图形的格式显示有关内存使用、线程、类和 CPU 使用情况的信息。Overview 选项卡在一个页面中显示了一组相关信息，而在以前需要在多个选项卡之间进行切换才能显示。图 1 展示了一个示例应用程序的 Overview 选项卡：

**图 1. JConsole Overview 选项卡**  
 

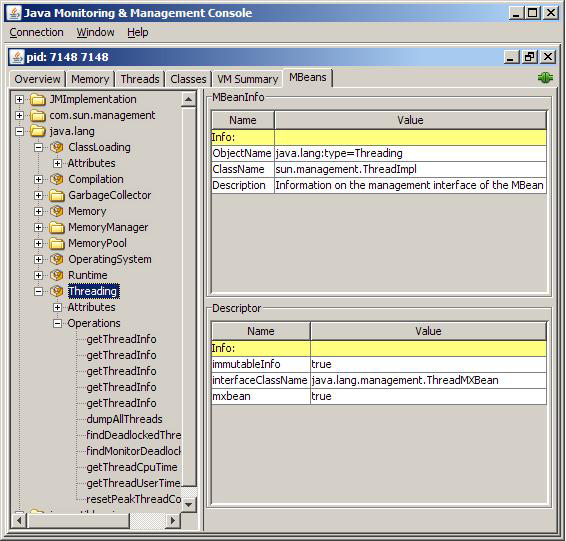
Overview 选项卡显示了 4 个反映 VM 资源使用情况的图形以及一个挑选列表，可以修改显示结果的时间范围。第一幅图像 Heap Memory Usage 显示堆内存量，随时间流逝以 GB 的单位增加。该图有助于检测内存泄漏。如果您的应用程序出现了内存泄漏，堆内存使用量将随时间稳步增长。

Threads 图形随时间的增长描绘活动线程的数量，而 Classes 图形描绘所装载的类的数量。CPU Usage 图表描述应用程序在其生命周期的不同时间点的 CPU 使用百分比。

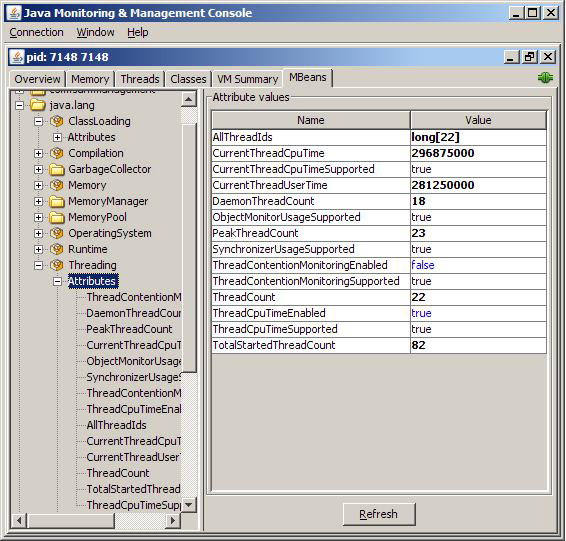
图 2 显示的 VM Summary 选项卡是对 Java SE 6 版本的另一个新的增强。它提供了有关 JVM 的详细信息，包括总的运行时间、线程信息、装载的类、内存统计信息、垃圾收集和操作系统信息。

**图 2. JConsole VM Summary 选项卡**  
 

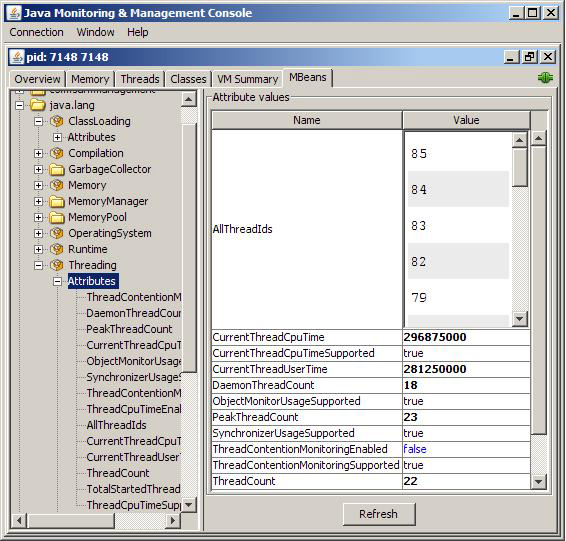
MBeans 选项卡进行了增强，可以更轻松地访问 MBean 的操作和属性。它显示所有注册到平台中的 MBean 的信息，通过该选项卡可以访问所有平台 MBeans。左侧的树结构显示当前运行的所有 MBean。当选择一个 MBean 时，其 MBeanInfo 和描述符将显示在右侧的表中，如图 3 所示：

**图 3. JConsole MBean 选项卡**  
 

选择 Attributes 节点将显示 MBean 的所有属性，如图 4 所示的 Threading MBean：

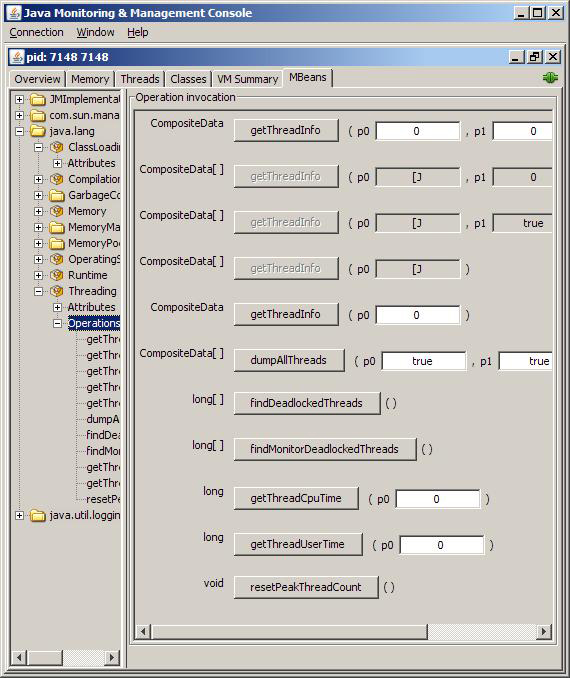
**图 4. MBean 属性**  
 

注意，方框右侧的属性及其值对应于通过前文介绍的 java.lang.management 包的 ThreadMXBean API 获得的属性值，通过双击属性值可获得所列属性的其他信息，只有粗体显示的属性值可被展开。例如，双击 AllThreadIds 值将显示 22 个线程的线程 ID，如图 5 所示：

**图 5. 展开的属性值**  
 

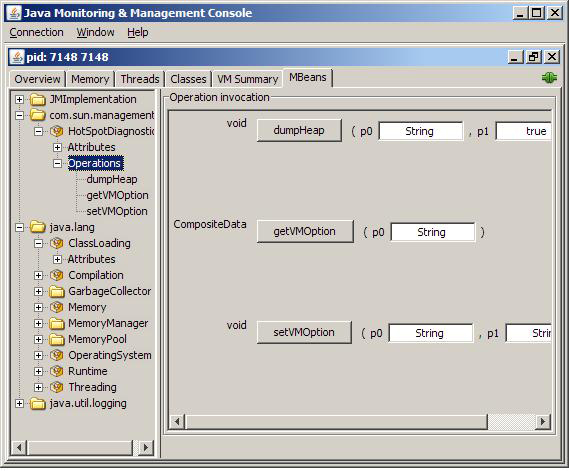
可写的属性以蓝色显示并可以通过单击它们进行编辑并输入新值。例如，图 5 所示的 ThreadContentionMonitoringAvailable 属性就可以以这种方式在 JConsole 视图中进行编辑。

选择左侧树结构中的 Operations 节点将显示与该 MBean 有关的操作。MBean 操作以按钮的形式显示在右侧的区域中，并且当单击时将调用指定的方法。图 6 显示了可用于 ThreadMXBean 的操作：

**图 6. MBean 操作**  
 

**HotSpot Diagnostic MBean**

在 Java SE 6 中，JConsole 提供了对 HotSpot Diagnostic MBean 的支持。这个引入的 MBean 允许您执行 on-the-spot 诊断操作。其 API 允许用户在运行时执行堆转储并设置其他 VM 选项。您可以从 MBean 选项卡访问 HotSpot Diagnostic MBean，方法为展开com.sun.management 节点并选择 HotSpotDiagnostic。图 7 显示了可用于 HotSpot Diagnostic MBean 的方法：

**图 7. HotSpot Diagnostic MBean**  
 

**JConsole 插件支持**

从 Java SE 6 开始，JConsole 提供了插件支持，允许您构建自己的插件来和 JConsole 一起运行。例如，您可以向 JConsole 主视图添加一个自定义选项卡，用于访问特定于应用程序的 MBeans 并执行自己的监视活动。

您必须扩展抽象的 com.sun.tools.jconsole.JConsolePlugin 类，创建一个自定义的 JConsole 插件。您将为插件实现 2 个方法，使之正确显示在 JConsole 视图中：

* newSwingWorker() 返回 SwingWorker 对象，它将对插件执行 GUI 更新。
* getTabs() 返回被添加到 JConsole 窗口中的选项卡图。

JConsole 使用其服务提供者机制检测并装载所有插件类。因此，您必须为包含 META-INF/services/com.sun.tools.jconsole.JConsolePlugin 文件的 JAR 文件提供插件类。该文件应当包含一组完全符合规则的插件类名，其中每行显示一个类名。要将新插件装载到 JConsole 视图，使用以下命令在命令行中运行 JConsole：

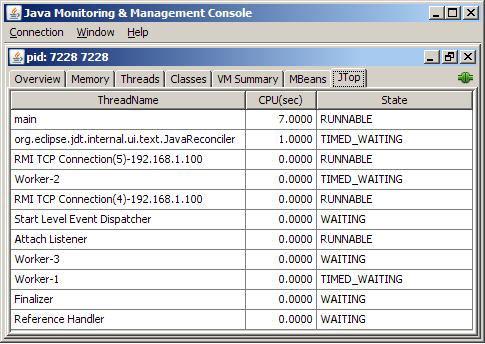
|  |
| --- |
| jconsole -pluginpath *plugin\_path* |

命令中的 *plugin\_path* 指目录路径或者 JConsole 插件的归档。您可以指定多条路径。

Java SE 6 附带了一个示例 JConsole 插件，称为 JTop。JTop 将显示当前应用程序内运行线程的 CPU 使用情况。要运行 JConsole 和 JTop，执行下面的命令：

|  |
| --- |
| jconsole -*pluginpath* JAVA\_HOME/demo/management/JTop/JTop.jar |

图 8 展示了 JConsole 的一个实例，它选择了 JTop 选项卡。左列显示了所有运行线程的名称。对于每个线程，将显示它的 CPU 使用情况和线程状态。当统计信息发生变化时，视图将自动刷新。JTop 插件可用于通过高 CPU 使用量识别线程。

**图 8. JConsole JTop 插件**  


[**回页首**](http://www.ibm.com/developerworks/cn/java/j-java6perfmon/index.html#ibm-pcon)

**监视和故障排除工具**

除 JConsole 外，Java SE 6 还提供了对其他命令行工具的支持。这些诊断工具可以连接到任何应用程序而不要求应用程序以特殊模式启动。它们使您能够获得更多应用程序信息，从而确定它是否按预期运行。注意，列出的这些工具仍处于实验性质，未来的 Java SE 版本未必能够提供完全的支持。

**监视工具**

Java SE 6 包括了三个命令行工具，如表 2 所示，这些工具可用于监视 JVM 性能统计信息：

**表 2. 监视工具**

|  |  |
| --- | --- |
| **工具** | **说明** |
| jps | JVM 进程状态工具 |
| jstat | JVM 统计信息监视工具 |
| jstatd | JVM jstat 守护程序 |

jps 工具为目标系统的当前用户列出虚拟机。这对于使用 JNI Invocation API 而不是标准 Java 启动程序启动 VM 的环境特别有用。在这些环境中，通常很难从进程列表中识别 Java 进程。jps 工具解决了这个问题。

下面的例子演示了 jps 工具的使用。在命令行输入 jps，该工具即可为具有访问权限的用户列出虚拟机和进程 ID，如清单 3 的示例所示：

**清单 3. 使用 jps 工具**

|  |
| --- |
| $ jps  16217 MyApplication  16342 jps |

jstat 工具使用 JVM 的内置测试工具，提供有关性能和所运行应用程序的资源消耗信息。该工具有助于诊断性能问题、与堆大小和垃圾收集有关的特殊问题。

jstatd 守护程序是一个 Remote Method Invocation (RMI) 服务器应用程序，它将监视 JVM 的创建和终止并提供接口以允许远程监视工具连接到运行在本地主机的 JVM。例如，这个守护程序允许 jps 工具列出远程系统中的进程。

有关这些工具的更多文档和使用示例，请参阅 [参考资料](http://www.ibm.com/developerworks/cn/java/j-java6perfmon/index.html#resources)。

**故障排除工具**

Java SE 6 还提供了一些故障排除工具，如表 3 所示，这些工具可帮助您找出应用程序中运行异常的部分：

**表 3. 故障排除工具**

|  |  |
| --- | --- |
| **工具** | **说明** |
| jinfo | 配置信息 |
| jhat | 堆转储浏览器 |
| jmap | 内存映射 |
| jsadebugd | 服务能力代理调试程序 |
| jstack | 堆栈跟踪 |

jinfo 命令行工具从运行中的 Java 进程或崩溃转储（crash dump）中获取配置信息，并对系统属性和用于启动虚拟机的标记进行打印。

jhat 工具提供了一种方便的方法，可以在堆快照中浏览对象结构。Java SE 6 版本中引入的这个工具可以取代 Heap Analysis Tool (HAT)，有助于检测内存泄漏。

jmap 命令行工具为运行中的 VM 或核心文件打印与内存有关的统计信息。该工具还可以使用 jsadebugd 守护程序查询远程机器中的进程或核心文件。jmap 工具可检测出是否过度使用了完成器，后者可导致出现 OutOfMemoryError 错误。

Serviceability Agent Debug Daemon (jsadebugd) 连接到一个 Java 进程或一个核心文件，并充当一个调试服务器。该工具当前只能用于 Solaris OS 和 Linux®。诸如 jstack、jmap 和 jinfo 这样的远程客户机可以通过 Java RMI 连接到这种服务器上。

jstack 命令行工具连接到指定的进程或核心文件，并打印所有连接到虚拟机的线程的堆栈跟踪信息，包括 Java 线程和 VM 内部线程，有时也包括本地堆栈框架。该工具还执行死锁检测。它使用 jsadebugd 守护程序查询远程机器上的进程或核心文件。jstack 工具可用于检测死锁问题。

关于这些工具的更多文档和使用示例，请参考 [参考资料](http://www.ibm.com/developerworks/cn/java/j-java6perfmon/index.html#resources)。

[**回页首**](http://www.ibm.com/developerworks/cn/java/j-java6perfmon/index.html#ibm-pcon)

**结束语**

Java 6 平台对 VM 测试、管理 API 和 JDK 工具提供了一些增强，可帮助有效地确定和诊断 Java 应用程序中的性能和内存问题。本文介绍了对 Java SE 监视和管理框架添加的增强以及为开发人员提供的诊断命令行工具。

Java 应用程序的平均性能随时间稳步提高。现在，使用 Java SE 6 版本，Java 性能可与 C 或 C++ 相媲美。在很多情况下，Java 代码的运行速度显著提升。您还可以使用本文介绍的工具实现更好的性能优化。请您亲自尝试。我们可以确保帮您找到以前从未进行过的应用程序优化。