快速故障排除技巧

Quick Troubleshooting Tips on Solaris OS and Linux for Java SE 7

This "Quick Start Guide" gives you some quick tips for troubleshooting. The subsections list some typical functions that can help you in troubleshooting, including one or more ways to get the information or perform the action.

These tips are organized as follows:

Hung, Deadlocked, or Looping Process 悬挂、死锁、循环进程

- Print thread stack for all Java threads: 打印所有Java线程的线程堆栈
  - Control-\
  - `kill -QUIT` *pid*
  - `jstack` *pid* (or `jstack -F` *pid* if `jstack` *pid* does not respond)
- Detect deadlocks: 检测死锁
  - Request deadlock detection: JConsole tool, Threads tab 请求死锁检测
  - Print information on deadlocked threads: Control-\
  - Print list of concurrent locks owned by each thread: `-XX:+PrintConcurrentLocks` set, then Control-\
  - Print lock information for a process: `jstack -l` *pid* 打印进程的锁信息
- Get a heap histogram for a process: 获取进程的堆历史
  - Start Java process with `-XX:+PrintClassHistogram`, then Control-\
  - `jmap -histo` *pid* (with `-F` option if *pid* does not respond)
- Dump Java heap for a process in binary format to file: 转储进程的Java堆 很耗机器CPU，线上机器请先下线
  - `jmap -dump:format=b,file=`*filename* *pid* (with `-F` option if *pid* does not respond)
- Print shared object mappings for a process: 打印进程的共享对象映射表
  - `jmap` *pid*
- Print heap summary for a process: 打印进程的堆概要信息
  - Control-\
  - `jmap -heap` *pid*
- Print finalization information for a process: 打印进程中对象等待终止的信息（资源连接Socket、线程）
  - `jmap -finalizerinfo` *pid*
- Attach the command-line debugger to a process: 附加命令行调试器到进程
  - `jdb -connect sun.jvm.hotspot.jdi.SAPIDAttachingConnector:pid=`*pid*

Post-mortem Diagnostics, Memory Leaks 事后剖析诊断（内存泄漏）

- Examine the fatal error log file. Default file name is `hs_err_pid`*pid*`.log` in the working-directory.
- Create a heap dump: 创建一个堆转储文件
  - Start the application with HPROF enabled: `java -agentlib:hprof=file=`*file*`,format=b` *application*; then Control-\
  - Start the application with HPROF enabled: `java -agentlib:hprof=heap=dump` *application*
  - JConsole tool, MBeans tab
  - Start VM with `-XX:+HeapDumpOnOutOfMemoryError`; if `OutOfMemoryError` is thrown, VM generates a heap dump. 如果OOM错误被抛出，则VM生成一个堆转储文件
- Browse Java heap dump: 浏览堆转储文件（HeapAnalyzer）
  - `jhat` *heap-dump-file* 很耗机器CPU，切勿在线上机器执行该命令
- Dump Java heap from core file in binary format to a file:
  - `jmap -dump:format=b,file=`*filename* *corefile*
- Get a heap histogram for a process:
  - Start Java process with `-XX:+PrintClassHistogram`, then Control-\
  - `jmap -histo` *pid* (with `-F` option if *pid* does not respond)
- Get a heap histogram from a core file:
  - `jmap -histo` *corefile*
- Print shared object mappings from a core file:
  - `jmap` *corefile*
- Print heap summary from a core file:
  - `jmap -heap` *corefile*
- Print finalization information from a core file:
  - `jmap -finalizerinfo` *corefile*
- Print Java configuration information from a core file:
  - `jinfo` *corefile*
- Print thread trace from a core file:
  - `jstack` *corefile*

- Print lock information from a core file:
  - `jstack -l` *corefile*
- Attach the command-line debugger to a core file on the same machine:
  - `jdb -connect sun.jvm.hotspot.jdi.SACoreAttachingConnector:javaExecutable=`*path*`,core=`*corefile*
- Attach the command-line debugger to a core file on a different machine:
  - On the machine with the core file: `jsadebugd` *path corefile*
    and on the machine with the debugger: `jdb -connect sun.jvm.hotspot.jdi.SADebugServerAttachingConnector:debugServerName=`*machine*
- `libumem` can be used to debug memory leaks.

Monitoring 监视

Note: The *vmID* argument for the `jstat` command is the virtual machine identifier. See the `jstat` man page for a detailed explanation.

- Print statistics on the class loader: 打印类加载器的统计数据（动态语言）
  - `jstat -class` *vmID*
- Print statistics on the compiler:
  - Compiler behavior: `jstat -compiler` *vmID*
  - Compilation method statistics: `jstat -printcompilation` *vmID*
- Print statistics on garbage collection: 打印垃圾收集的统计数据
  - Summary of statistics: `jstat -gcutil` *vmID* 统计概述：jstat -gcutil vmID 1s
  - Summary of statistics, with causes: `jstat -gccause` *vmID*
  - Behavior of the gc heap: `jstat -gc` *vmID* GC 堆的行为
  - Capacities of all the generations: `jstat -gccapacity` *vmID* 所有后代的能力
  - Behavior of the new generation: `jstat -gcnew` *vmID* Eden 区：新生代的行为
  - Capacity of the new generation: `jstat -gcnewcapacity` *vmID*
  - Behavior of the old and permanent generations: `jstat -gcold` *vmID* Old、Perm 区：老年代和永久代的行为
  - Capacity of the old generation: `jstat -gcoldcapacity` *vmID*
  - Capacity of the permanent generation: `jstat -gcpermcapacity` *vmID*
- Monitor objects awaiting finalization: 监视等待终止的对象（资源连接泄漏）
  - JConsole tool, VM Summary tab
  - `jmap -finalizerinfo` *pid*
  - `getObjectPendingFinalizationCount` method in `java.lang.management.MemoryMXBean` class
- Monitor memory: 监视内存
  - Heap allocation profiles via HPROF: `java -agentlib:hprof=heap=sites`
  - JConsole tool, Memory tab
  - Control-\ prints generation information.

  Monitor CPU usage: 监视CPU使用率（线程堆栈、方法）
  - By thread stack: `java -agentlib:hprof=cpu=samples` *application*
  - By method: `java -agentlib:hprof=cpu=times` *application*
  - JConsole tool, Overview and VM Summary tabs
- Monitor thread activity: 监视活动线程
  - JConsole tool, Threads tab
- Monitor class activity: 监视已加载的类
  - JConsole tool, Classes tab

Actions on a Remote Debug Server

First, attach the debug daemon `jsadebugd`, then execute the command:

- Dump Java heap in binary format to a file: `jmap -dump:format=b,file=`*filename hostID*
- Print shared object mappings: `jmap` *hostID*
- Print heap summary : `jmap -heap` *hostID*
- Print finalization information : `jmap -finalizerinfo` *hostID*
- Print lock information : `jstack -l` *hostID*
- Print thread trace : `jstack` *hostID*
- Print Java configuration information: `jinfo` *hostID*

Other Functions

- Interface with the instrumented Java virtual machines:
  - Monitor for the creation and termination of instrumented VMs: `jstatd` daemon
  - List the instrumented VMs: `jps` 列出检测到虚拟机实例：jps -l 详细启动参数：jps -lv
  - Provide interface between remote monitoring tools and local VMs: `jstatd` daemon
  - Request garbage collection: JConsole tool, Memory tab
- Print Java configuration information from a running process:

  - `jinfo` *pid*
- Dynamically set, unset, or change the value of certain Java VM flags for a process:
  - `jinfo -flag` *flag*

动态更改某些虚拟机标识的值

- Print command-line flags passed to the VM:
  - `jinfo -flags`

-D<name>=<value>

- Print Java system properties:
  - `jinfo -sysprops`
- Pass a Java VM flag to the virtual machine:
  - `jconsole -J`*flag* ...
  - `jhat -J`*flag* ...
  - `jmap -J`*flag* ...
- Print statistics of permanent generation of Java heap, by class loader:
  - `jmap -permstat`
- Report on monitor contention.
  - `java -agentlib:hprof=monitor=y` *application*
- Evaluate or execute a script in interactive or batch mode:
  - `jrunscript`
- Interface dynamically with an MBean, via JConsole tool, MBean tab:
  - Show tree structure.
  - Set an attribute value.
  - Invoke an operation.
  - Subscribe to notification.
- Run interactive command-line debugger:
  - Launch a new VM for the class: `jdb` *class*
  - Attach debugger to a running VM: `jdb -attach` *address*