



JAVA虚拟机-实战篇

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



JCMD

- 导出堆、查看Java进程、导出线程信息、执行GC、还可以进行采样分析。【功能齐全】
- `jcmd -h` 查看帮助信息

```
[root@dev-yd-jyh01 /]# jcmd -h
```

```
Usage: jcmd <pid | main class> <command ...|PerfCounter.print|-f file>  
or: jcmd -l  
or: jcmd -h
```

command must be a valid jcmd command for the selected jvm.
Use the command "help" to see which commands are available.
If the pid is 0, commands will be sent to all Java processes.
The main class argument will be used to match (either partially
or fully) the class used to start Java.
If no options are given, lists Java processes (same as -p).

PerfCounter.print display the counters exposed by this process

-f read and execute commands from the file
-l list JVM processes on the local machine
-h this help





JCMD – 命令格式

```
jcmbd <pid | main class> <command ... | PerfCounter.print | -f file>  
jcmbd -l  
jcmbd -h
```



- pid：接收诊断命令请求的进程ID。
- main class：接收诊断命令请求的进程的main类。匹配进程时，main类名称中包含指定子字符串的任何进程均是匹配的。如果多个正在运行的Java进程共享同一个main类，诊断命令请求将会发送到所有的这些进程中。
- command：命令
- **Perfcounter.print：打印目标Java进程上可用的性能计数器。性能计数器的列表可能会随着Java进程的不同而产生变化。**
- -f file：从文件file中读取命令，然后在目标Java进程上调用这些命令。在file中，每个命令必须写在单独的一行。以“#”开头的行会被忽略。当所有行的命令被调用完毕后，或者读取到含有stop关键字的命令，将会终止对file的处理。(使用较少)
- **-l：查看所有的进程列表信息。**
- **-h：查看帮助信息。（同 -help）**



JCMD – 查看java进程

```
[root@dev-yd-jyh01 /]# jcmd
14128 sun.tools.jcmd.JCmd
6835 org.apache.catalina.startup.Bootstrap start
```

```
[root@dev-yd-jyh01 /]# jcmd -l
6835 org.apache.catalina.startup.Bootstrap start
14173 sun.tools.jcmd.JCmd -l
```

```
[root@dev-yd-jyh01 /]# jps
6835 Bootstrap
14195 Jps-l
jcmd -h
```





JCMD – 查看性能统计数据

```
[root@dev-yd-jyh01 /]# jcmd 6835 PerfCounter.print
```

```
6835:
```

```
java.ci.totalTime=99466356310
```

```
java.cls.loadedClasses=9384
```

```
java.cls.sharedLoadedClasses=0
```

```
java.cls.sharedUnloadedClasses=0
```

```
java.cls.unloadedClasses=177
```

```
java.property.java.class.path="/data/tomcat/tomcat_jyh_app/bin/bootstrap.jar:/data/tomcat/tomcat_jyh_app/bin/tomcat-juli.jar"
```

```
java.property.java.endorsed.dirs="/data/tomcat/tomcat_jyh_app/endorsed"
```

```
java.property.java.ext.dirs="/usr/java/jdk1.8.0_45/jre/lib/ext:/usr/java/packages/lib/ext"
```

```
java.property.java.home="/usr/java/jdk1.8.0_45/jre"
```

```
java.property.java.library.path="/usr/java/packages/lib/amd64:/usr/lib64:/lib64:/lib:/usr/lib"
```

```
java.property.java.version="1.8.0_45"
```

```
java.property.java.vm.info="mixed mode"
```

```
java.property.java.vm.name="Java HotSpot(TM) 64-Bit Server VM"
```

```
java.property.java.vm.specification.name="Java Virtual Machine Specification"
```

```
java.property.java.vm.specification.vendor="Oracle Corporation"
```

```
.....
```





JCMD – 进程可执行的命令

```
[root@dev-yd-jyh01 /]# jcmd 6835 help
```

6835:

The following commands are available:

JFR.stop

JFR.start

JFR.dump

JFR.check

VM.native_memory

VM.check_commercial_features

VM.unlock_commercial_features

ManagementAgent.stop

ManagementAgent.start_local

ManagementAgent.start

GC.rotate_log

Thread.print

GC.class_stats

GC.class_histogram

GC.heap_dump

GC.run_finalization

GC.run

VM.uptime

VM.flags

VM.system_properties

VM.command_line

VM.version

help





JCMD – 进程可执行的命令-参数

- 查看可执行命令还有哪些参数选项，以及这些参数选项的使用方法

```
[root@dev-yd-jyh01 /]# jcmd 6835 help JFR.stop
```

```
6835:
```

```
JFR.stop
```

```
Stops a JFR recording
```

```
Impact: Low
```

```
Permission: java.lang.management.ManagementPermission(monitor)
```

```
Syntax : JFR.stop [options]
```

```
Options: (options must be specified using the <key> or <key>=<value> syntax)
```

```
name : [optional] Recording name,.e.g \"My Recording\" (STRING, no default value)
```

```
recording : [optional] Recording number, see JFR.check for a list of available recordings (JLONG, -1)
```

```
discard : [optional] Skip writing data to previously specified file (if any) (BOOLEAN, false)
```

```
filename : [optional] Copy recording data to file, e.g. \"/home/user/My Recording.jfr\" (STRING, no default value)
```

```
compress : [optional] GZip-compress "filename" destination (BOOLEAN, false)
```





VM参数

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



JCMD-VM参数

VM.uptime

虚拟机已启动时长



VM.system_properties

系统属性



VM.native_memory

本地内存分配情况



VM.flags

VM的标志选项和他们的当前值



VM.command_line

VM实例启动时使用的命令行



VM.version

版本信息



VM参数

虚拟机相关信息查看



JCMD – VM.uptime 虚拟机启动时长



```
[root@dev-yd-jyh01 /]# jcmd 6835 help VM.uptime
```

```
6835:
```

```
VM.uptime
```

```
Print VM uptime.
```

```
Impact: Low
```

```
Syntax : VM.uptime [options]
```

```
Options: (options must be specified using the <key> or <key>=<value> syntax)
```

```
    -date : [optional] Add a prefix with current date (BOOLEAN, false) #增加日期前缀
```

```
[root@dev-yd-jyh01 /]# jcmd 6835 VM.uptime
```

```
6835:
```

```
1372062.259 s
```

```
[root@dev-yd-jyh01 /]# jcmd 6835 VM.uptime -date
```

```
6835:
```

```
2019-06-20T11:31:30.794+0800: 1372072.575 s
```

```
[root@dev-yd-jyh01 /]# jcmd 6835 VM.uptime -date=true
```

```
6835:
```

```
2019-06-20T11:31:37.330+0800: 1372079.106 s
```

```
[root@dev-yd-jyh01 /]# jcmd 6835 VM.uptime -date=false
```

```
6835:
```



JCMD – VM.flags虚拟机标志选项



```
[root@dev-yd-jyh01 /]# jcmd 6835 help VM.flags
```

```
6835:
```

```
VM.flags
```

```
Print VM flag options and their current values. #打印VM的标志选项和他们的当前值
```

```
Impact: Low
```

```
Permission: java.lang.management.ManagementPermission(monitor)
```

```
Syntax : VM.flags [options]
```

```
Options: (options must be specified using the <key> or <key>=<value> syntax)
```

```
-all : [optional] Print all flags supported by the VM (BOOLEAN, false) #打印所有支持的参数
```

```
[root@dev-yd-jyh01 /]# jcmd 6835 VM.flags #打印当前参数
```

```
6835:
```

```
-XX:CICompilerCount=4 -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=heapdump.hprof -XX:InitialHeapSize=1073741824 -
```

```
XX:+ManagementServer -XX:MaxHeapSize=1073741824 -XX:MaxNewSize=536870912 -XX:MinHeapDeltaBytes=524288 -
```

```
XX:NewSize=536870912 -XX:OldSize=536870912 -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -
```

```
XX:+UseFastUnorderedTimeStamps -XX:+UseParallelGC
```

```
[root@dev-yd-jyh01 /]# jcmd 6835 VM.flags -all #打印VM所有支持的参数
```

```
6835:
```

```
[Global flags]
```

```
uintx AdaptiveSizeDecrementScaleFactor      = 4                {product}
```

```
uintx AdaptiveSizeMajorGCDecayTimeScale      = 10                {product}
```

```
uintx AdaptiveSizePausePolicy                 = 0                {product}
```

```
uintx AdaptiveSizePolicyCollectionCostMargin = 50                {product}
```



JCMD – VM.system_properties系统属性



```
[root@dev-yd-jyh01 /]# jcmd 6835 help VM.system_properties
```

```
6835:
```

```
VM.system_properties
```

```
Print system properties.
```

```
Impact: Low
```

```
Permission: java.util.PropertyPermission(*, read)
```

```
Syntax: VM.system_properties
```

```
[root@dev-yd-jyh01 /]# jcmd 6835 VM.system_properties #打印虚拟机的系统属性
```

```
6835:
```

```
#Thu Jun 20 11:47:59 CST 2019
```

```
com.sun.management.jmxremote.authenticate=false
```

```
java.runtime.name=Java(TM) SE Runtime Environment
```

```
sun.boot.library.path=/usr/java/jdk1.8.0_45/jre/lib/amd64
```

```
java.vm.version=25.45-b02
```

```
shared.loader=
```

```
java.vm.vendor=Oracle Corporation
```

```
java.vendor.url=http://java.oracle.com/
```

```
path.separator=\:
```

```
java.rmi.server.randomIDs=true
```

```
java.vm.name=Java HotSpot(TM) 64-Bit Server VM
```

```
tomcat.util.buf.StringCache.byte.enabled=true
```

```
file.encoding.pkg=sun.io
```

```
java.util.logging.config.file=/data/tomcat/tomcat_jyh_app/conf/logging.properties
```



JCMD – VM.command_line启动参数



```
[[root@dev-yd-jyh01 /]# jcmd 6835 help VM.command_line
```

6835:

VM.command_line

Print the command line used to start this VM instance. #打印VM实例启动时使用的参数

Impact: Low

Permission: java.lang.management.ManagementPermission(monitor)

Syntax: VM.command_line #没有额外参数

```
[root@dev-yd-jyh01 /]# jcmd 6835 VM.command_line
```

6835:

VM Arguments:

jvm_args: -Djava.util.logging.config.file=/data/tomcat/tomcat_jyh_app/conf/logging.properties -

Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Xms1024m -Xmx1024m -XX:PermSize=1024m -

XX:MaxPermSize=1024m -XX:NewSize=256m -XX:MaxNewSize=512m -XX:+HeapDumpOnOutOfMemoryError -

XX:HeapDumpPath=heapdump.hprof -Dfile.encoding=utf-8 -Dcom.sun.management.jmxremote -

Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -

Djava.endorsed.dirs=/data/tomcat/tomcat_jyh_app/endorsed -Dcatalina.base=/data/tomcat/tomcat_jyh_app -

Dcatalina.home=/data/tomcat/tomcat_jyh_app -Djava.io.tmpdir=/data/tomcat/tomcat_jyh_app/temp

java_command: org.apache.catalina.startup.Bootstrap start

java_class_path (initial): /data/tomcat/tomcat_jyh_app/bin/bootstrap.jar:/data/tomcat/tomcat_jyh_app/bin/tomcat-juli.jar



JCMD – VM.version版本信息



```
[root@dev-yd-jyh01 ~]# jcmd 6835 help VM.version
```

```
6835:
```

```
VM.version
```

```
Print JVM version information.
```

```
Impact: Low
```

```
Permission: java.util.PropertyPermission(java.vm.version, read)
```

```
Syntax: VM.version #没有额外参数
```

```
[root@dev-yd-jyh01 ~]# jcmd 6835 VM.version #输出VM版本
```

```
6835:
```

```
Java HotSpot(TM) 64-Bit Server VM version 25.45-b02
```

```
JDK 8.0_45
```

(Kevin)

版权所有 侵权必究



JCMD – VM.native_memory本地内存分配情况



```
[root@dev-yd-jyh01 ~]# jcmd 6835 help VM.native_memory
```

```
6835:
```

```
VM.native_memory
```

```
Print native memory usage #打印本地内存分配情况
```

```
Impact: Medium #影响：中等
```

```
Permission: java.lang.management.ManagementPermission(monitor)
```

```
Syntax : VM.native_memory [options]
```

```
Options: (options must be specified using the <key> or <key>=<value> syntax)
```

summary : [optional] request runtime to report current memory summary, which includes total reserved and committed memory, along with memory usage summary by each subsystem. (BOOLEAN, false) #请求运行时报告当前内存摘要，其中包括保留和提交的总内存，以及每个子系统的内存使用摘要。

detail : [optional] request runtime to report memory allocation >= 1K by each callsite. (BOOLEAN, false) #请求运行时报告每个调用的内存分配大于等于1K。

baseline : [optional] (BOOLEAN, false) request runtime to baseline current memory usage, so it can be compared against in later time. #请求运行时将当前内存使用情况作为基线，以便在以后将其与之进行比较。

summary.diff : [optional] request runtime to report memory summary comparison against previous baseline. (BOOLEAN, false) #请求运行时报告与上一基线的内存摘要比较。

detail.diff : [optional] request runtime to report memory detail comparison against previous baseline, which shows the memory allocation activities at different callsites. (BOOLEAN, false) #请求运行时报告与上一基线的内存详细信息比较，该基线显示不同调用站点的内存分配活动

shutdown : [optional] request runtime to shutdown itself and free the memory used by runtime. (BOOLEAN, false) #请求运行时关闭自身并释放运行时使用的内存。

statistics : [optional] print tracker statistics for tuning purpose. (BOOLEAN, false) #打印跟踪统计信息以进行调整

scale : [optional] Memory usage in which scale, KB, MB or GB (STRING, KB) #内存使用率，其中的刻度、KB、MB或GB

```
[root@dev-yd-jyh01 ~]# jcmd 6835 VM.native_memory
```

```
6835:
```



Native Memory Tracking (NMT)

- Java8的HotSpot VM引入了Native Memory Tracking (NMT)特性，可以用于追踪JVM的内部内存使用。
 - NMT默认是关闭的，需要使用-XX:NativeMemoryTracking=[off|summary|detail] 参数去开启，
 - 默认是off状态
 - summary：仅收集系统聚合的内存使用情况。
 - detail：收集各个调用站点的内存使用情况。
 - 启用此选项将导致5-10%的开销。
-
- jcmd 9684 VM.native_memory summary scale=MB 显示摘要：打印按照类别汇总的摘要
 - jcmd 9684 VM.native_memory detail scale=MB 显示明细：按类别聚合的打印内存使用情况,打印虚拟内存映射,按调用站点聚合的打印内存使用情况
 - jcmd 9684 VM.native_memory baseline 创建基线(内存使用快照)用于比较
 - jcmd 9684 VM.native_memory summary.diff scale=MB 显示摘要差异,与基线比较
 - jcmd 9684 VM.native_memory detail.diff scale=MB 显示明细差异，与基线比较
 - jcmd 9684 VM.native_memory shutdown 关闭MNT, 关闭后不可再跟踪,必须重启服务



Native Memory Tracking (NMT)

Native Memory Tracking:

Total: reserved=3124MB, committed=454MB

- Java Heap (reserved=1676MB, committed=265MB) #JAVA堆
 - (mmap: reserved=1676MB, committed=265MB)
- Class (reserved=1074MB, committed=57MB) #类元数据
 - (classes #10170)
 - (malloc=6MB #11336)
 - (mmap: reserved=1068MB, committed=51MB)
- Thread (reserved=37MB, committed=37MB) #线程使用的内存，包括线程数据结构、资源区和句柄区等。
 - (thread #38)
 - (stack: reserved=37MB, committed=37MB)
- Code (reserved=245MB, committed=10MB) #生成的代码
 - (malloc=2MB #4369)
 - (mmap: reserved=244MB, committed=8MB)
- GC (reserved=67MB, committed=62MB) #GC使用的数据
 - (malloc=6MB #211)
 - (mmap: reserved=61MB, committed=57MB)
- Internal (reserved=7MB, committed=7MB) #不符合上述类别的内存，如命令行解析器使用的内存、jvmti、属性等。
 - (malloc=7MB #13865)
- Symbol (reserved=14MB, committed=14MB) #符号
 - (malloc=12MB #118823)
 - (arena=2MB #1)



Thread参数

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



JCMD – 线程打印命令的使用

```
[root@dev-yd-jyh01 ~]# jcmd 6835 help Thread.print
```

6835:

Thread.print

Print all threads with stacktraces. #打印所有线程堆栈信息

Impact: Medium: Depends on the number of threads.

Permission: java.lang.management.ManagementPermission(monitor)

Syntax : Thread.print [options]

Options: (options must be specified using the <key> or <key>=<value> syntax)

-l : [optional] print java.util.concurrent locks (BOOLEAN, false) #打印java.util.concurrent锁





JCMD – 线程堆栈打印



```
[root@dev-yd-jyh01 ~]# jcmd 6835 Thread.print
```

```
6835: #进程ID
```

```
2019-06-26 08:23:16 #打印时间
```

```
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.45-b02 mixed mode): #虚拟机版本描述
```

```
“Attach Listener“ #线程名称 #209 #线程号 daemon #守护进程 prio=9 #线程优先级 os_prio=0 #操作系统内优先级 tid=0x00007fa55402a800 #线程ID nid=0x547 #线程对应的本地线程ID waiting on condition [0x0000000000000000] #等待[某ID]的锁
```

```
java.lang.Thread.State: RUNNABLE #线程状态
```

```
"http-nio-8002-exec-128" #208 daemon prio=5 os_prio=0 tid=0x00007fa498079800 nid=0x1c60 waiting on condition [0x00007fa475ddc000]
```

```
java.lang.Thread.State: WAITING (parking)
```

```
at sun.misc.Unsafe.park(Native Method)
```

```
- parking to wait for <0x00000000dce0eb08> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
```

```
at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
```

```
at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039)
```

```
at java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:442)
```

```
at org.apache.tomcat.util.threads.TaskQueue.take(TaskQueue.java:103)
```

```
at org.apache.tomcat.util.threads.TaskQueue.take(TaskQueue.java:31)
```

```
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1067)
```

```
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)
```

```
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
```

```
at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
```

```
at java.lang.Thread.run(Thread.java:745)
```

(Kevin)

版权所有 侵权必究



JCMD – 线程堆栈打印 –I参数



```
[root@dev-yd-jyh01 ~]# jcmd 6835 Thread.print -l
```

```
6835: #进程ID
```

```
2019-06-26 08:23:16 #打印时间
```

```
Full thread dump Java HotSpot(TM) 64-Bit Server VM (25.45-b02 mixed mode): #虚拟机版本描述
```

```
"http-nio-8002-exec-128" #208 daemon prio=5 os_prio=0 tid=0x00007fa498079800 nid=0x1c60 waiting on condition [0x00007fa475ddc000]
```

```
java.lang.Thread.State: WAITING (parking)
```

```
at sun.misc.Unsafe.park(Native Method)
```

```
- parking to wait for <0x00000000dce0eb08> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject) #挂起等待ID为0x00000000dce0eb08的锁对象,以及对象对应的类型
```

```
at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
```

```
at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039)
```

```
at java.util.concurrent.LinkedBlockingQueue.take(LinkedBlockingQueue.java:442)
```

```
at org.apache.tomcat.util.threads.TaskQueue.take(TaskQueue.java:103)
```

```
at org.apache.tomcat.util.threads.TaskQueue.take(TaskQueue.java:31)
```

```
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1067)
```

```
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)
```

```
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
```

```
at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
```

```
at java.lang.Thread.run(Thread.java:745)
```

```
Locked ownable synchronizers: #当前线程持有的锁对象, 类型和地址
```

```
- None #None代表没有持有锁
```

```
- <0x22be64e0> (a java.util.concurrent.ThreadPoolExecutor$Worker) #如果持有锁,则显示形式为这样
```



JCMD – 线程堆栈分析

堆栈分析：《jcmt Thread.print 解析.pdf》

Demo：mima-jvm-server项目：com.mimaxueyuan.jvm.controller.MainController

线程状态

- NEW：未启动状态
- RUNNABLE：可运行状态：处于可运行状态的线程正在 Java 虚拟机中执行，但也可能是正在等待来自操作系统资源，例如 CPU。
- BLOCKED：阻塞状态
- WAITING：等待状态：线程调用Object.wait()、Thread.join()、LockSupport.park()进入等待状态，其他线程调用Object.notify()、Object.notifyAll()、Thread.join()则线程继续执行。
- TIMED_WAITING：具有指定时间的等待状态，Thread.sleep()、Object.wait()、Thread.join()、LockSupport.parkNanos()、LockSupport.parkUntil()
- TERMINATED：已终止线程的线程状态或线程已完成执行

Demo：mima-jvm-server项目：com.mimaxueyuan.jvm.jcmt.ThreadStateDemo



GC参数

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



JCMD – GC参数

GC.class_stats

类的元数据统计信息



GC.class_histogram

打印java堆的使用统计



GC.run

执行System.gc()



GC.heap_dump

转储堆快照文件



GC.run_finalization

执行System.runFinalization()



GC.heap_info

堆信息



GC
GC相关参数



JCMD – GC.class_stats打印class元数据统计



```
[root@dev-yd-jyh01 ~]# jcmd 6835 help GC.class_stats
```

6835:

GC.class_stats

Provide statistics about Java class meta data. Requires -XX:+UnlockDiagnosticVMOptions. #提供关于java类元数据统计信息，需要开启UnlockDiagnosticVMOptions参数

Impact: High: Depends on Java heap size and content. #这个命令影响性能较大,取决于堆的大小和内容

Syntax : GC.class_stats [options] [<columns>]

Arguments:

columns : [optional] Comma-separated list of all the columns to show. If not specified, the following columns are shown:

InstBytes,ClassBytes,CpAll,annotations,MethodCount,Bytecodes,MethodAll,ROAll,RWAll,Total (STRING, no default value) #列：[可选]要显示的所有列的逗号分隔列表。如果未指定，将显示以下列：instbytes、klassbytes、cpall、annotations、methodcount、bytecodes、methodall、roall、rwall、total (字符串，无默认值)

Options: (options must be specified using the <key> or <key>=<value> syntax)

-all : [optional] Show all columns (BOOLEAN, false) #显示所有列

-csv : [optional] Print in CSV (comma-separated values) format for spreadsheets (BOOLEAN, false) #csv格式输出

-help : [optional] Show meaning of all the columns (BOOLEAN, false) #显示列的含义

● jcmd 6224 GC.class_stats -all > d:/tmp.txt #信息量太大输出到文件里

● jcmd 6224 GC.class_stats -csv > d:/tmp.csv #输出到CSV文件里

● jcmd 6224 GC.class_stats "InstSize,InstCount,InstBytes" #常用显示列,多个列要用引号和逗号

● Demo辅助：mima-jvm-server 项目，本地windows环境演示



JCMD – GC.class_histogram打印java堆的使用统计



```
[root@dev-yd-jyh01 ~]# jcmd 6835 help GC.class_histogram #histogram 直方图
```

```
6835:
```

```
GC.class_histogram
```

```
Provide statistics about the Java heap usage. #提供有关Java堆使用的统计信息
```

```
Impact: High: Depends on Java heap size and content. #这个命令影响性能较大,取决于堆的大小和内容
```

```
Permission: java.lang.management.ManagementPermission(monитор)
```

```
Syntax : GC.class_histogram [options]
```

```
Options: (options must be specified using the <key> or <key>=<value> syntax)
```

```
-all : [optional] Inspect all objects, including unreachable objects (BOOLEAN, false) #检查所有对象,包括无法访问的对象
```

```
[root@dev-yd-jyh01 ~]# jcmd 6835 GC.class_histogram
```

```
6835:
```

```
num    #instances    #bytes  class name----- #占用空间从大大小排列, 第一列序号, 第二列对象数, 第三列类名
```

```
1:      3145728      301989888  com.tianrong.core.dao.mybatis.TransLogPOWithBLOBs
```

```
2:      3145728       75497472  com.tianrong.core.disruptor.log.LogEvent
```

```
3:       214996       29533360  [C
```

```
4:        7748       15006168  [B
```

```
5:        7375       14751784  [I
```

```
6:       33220       14462144  [Ljava.lang.Object;
```

```
7:       63391       5578408   java.lang.reflect.Method
```

```
[root@dev-yd-jyh01 ~]# jcmd 6835 GC.class_histogram -all #打印所有
```



JCMD – GC.run执行System.gc()

```
[root@dev-yd-jyh01 ~]# jcmd 6835 help GC.run
```

```
6835:
```

```
GC.run
```

Call java.lang.System.gc(). #对 JVM 执行 java.lang.System.gc()，告诉垃圾收集器打算进行垃圾收集，而垃圾收集器进不进行收集是不确定的。
相当于：System.gc()。

Impact: Medium: Depends on Java heap size and content. #影响：中等：取决于Java堆大小和内容。

Syntax: GC.run

```
[root@dev-yd-jyh01 ~]# jcmd 6835 GC.run
```

```
6835:
```

```
Command executed successfully #执行成功
```





JCMD – GC.run_finalization执行System.runFinalization()



```
[root@dev-yd-jyh01 ~]# jcmd 6835 help GC.run_finalization
```

```
6835:
```

```
GC.run_finalization
```

```
Call java.lang.System.runFinalization(). #对 JVM 执行 java.lang.System.runFinalization()。执行一次finalization操作，相当于：  
System.runFinalization()。
```

```
Impact: Medium: Depends on Java content. #影响：中等：取决于Java堆大小和内容。
```

```
Syntax: GC.run_finalization
```

```
[root@dev-yd-jyh01 ~]# jcmd 6835 GC.run_finalization
```

```
6835:
```

```
Command executed successfully #执行成功
```

● System.gc()和System.runFinalization()区别

- System.gc(); 告诉垃圾收集器打算进行垃圾收集，而垃圾收集器进不进行收集是不确定的
- System.runFinalization();强制调用已经失去引用的对象的finalize方法
- 当垃圾收集器认为没有指向对象实例的引用时，会在销毁该对象之前调用finalize()方法。该方法最常见的作用是确保释放实例占用的全部资源。java并不保证定时为对象实例调用该方法，甚至不保证方法会被调用，所以该方法不应该用于正常内存处理。



JCMD – GC.heap_info堆信息

8680:

```
PSYoungGen    total 240640K, used 88874K [0x00000000d5d00000, 0x00000000ecc80000, 0x0000000100000000)
 eden space 226304K, 32% used [0x00000000d5d00000,0x00000000da5d0ba0,0x00000000e3a00000)
 from space 14336K, 99% used [0x00000000e3a00000,0x00000000e47f9dc8,0x00000000e4800000)
 to   space 18944K, 0% used [0x00000000eba00000,0x00000000eba00000,0x00000000ecc80000)
ParOldGen     total 93184K, used 29085K [0x0000000081600000, 0x0000000087100000, 0x00000000d5d00000)
 object space 93184K, 31% used [0x0000000081600000,0x0000000083267740,0x0000000087100000)
Metaspace     used 53394K, capacity 56058K, committed 56360K, reserved 1097728K
 class space   used 7427K, capacity 7902K, committed 7976K, reserved 1048576K
```





JCMD – GC.heap_dump转储堆快照文件



```
[root@dev-yd-jyh01 ~]# jcmd 6835 help GC.heap_dump #dump转储
```

6835:

GC.heap_dump

Generate a HPROF format dump of the Java heap. #JAVA堆转储为HPROF格式文件

Impact: High: Depends on Java heap size and content. Request a full GC unless the ‘-all’ option is specified. #对性能影响高

Permission: java.lang.management.ManagementPermission(monitor)

Syntax : GC.heap_dump [options] <filename>

Arguments:

filename : Name of the dump file (STRING, no default value) #参数：转储文件名

Options: (options must be specified using the <key> or <key>=<value> syntax)

-all : [optional] Dump all objects, including unreachable objects (BOOLEAN, false) #-all 转储所有对象包括无法访问的对象

```
[root@dev-yd-jyh01 ~]# jcmd 6835 GC.heap_dump -all /data/heap_dump_all.hprof #导出所有对象,指定路径和文件名
```

6835:

Heap dump file created

- -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/data/logs/heap_dump.hprof
- 在发生内存溢出时转储堆快照文件、指定快照文件的存储位置
- 生产环境下快照文件很大，需要放置在大数据卷
- 快照文件可以使用Memory Analyzer Tool (MAT) 分析、也可以使用JProfiler分析。
- MAT下载地址：<http://www.eclipse.org/mat/downloads.php>



JProfiler内存泄漏分析实战

实战演示

1. 运行mima-jvm-server服务
2. 请求接口使服务内存溢出，并生成快照文件；`curl localhost:8080/demo/outmem`
3. 使用JProfiler工具分析



ManagementAgent参数

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



JCMD – 管理代理

ManagementAgent.start

启动管理代理



ManagementAgent.stop

停止管理代理

JMX

Java

Management Extension

JMX是管理系统和资源之间的一个接口，它定义了管理系统和资源之间交互的标准

ManagementAgent.start_local

开启本地管理代理





JCMD – 开启远程管理代理



```
[root@dev-yd-jyh01 ~]# jcmd 12379 help ManagementAgent.start
```

```
12379:
```

```
ManagementAgent.start
```

```
Start remote management agent. #开启远程管理代理
```

```
Impact: Low: No impact #对性能影响低
```

```
Syntax : ManagementAgent.start [options]
```

```
Options: (options must be specified using the <key> or <key>=<value> syntax)
```

```
    config.file : [optional] set com.sun.management.config.file (STRING, no default value)
```

```
    jmxremote.port : [optional] set com.sun.management.jmxremote.port (STRING, no default value)
```

```
    jmxremote.rmi.port : [optional] set com.sun.management.jmxremote.rmi.port (STRING, no default value)
```

```
    jmxremote.ssl : [optional] set com.sun.management.jmxremote.ssl (STRING, no default value)
```

```
    jmxremote.registry.ssl : [optional] set com.sun.management.jmxremote.registry.ssl (STRING, no default value)
```

```
    jmxremote.authenticate : [optional] set com.sun.management.jmxremote.authenticate (STRING, no default value)
```

```
    jmxremote.password.file : [optional] set com.sun.management.jmxremote.password.file (STRING, no default value)
```

```
    jmxremote.access.file : [optional] set com.sun.management.jmxremote.access.file (STRING, no default value)
```

```
    jmxremote.login.config : [optional] set com.sun.management.jmxremote.login.config (STRING, no default value)
```

```
    jmxremote.ssl.enabled.cipher.suites : [optional] set com.sun.management.jmxremote.ssl.enabled.cipher.suite (STRING, no default value)
```

```
    jmxremote.ssl.enabled.protocols : [optional] set com.sun.management.jmxremote.ssl.enabled.protocols (STRING, no default value)
```

```
    jmxremote.ssl.need.client.auth : [optional] set com.sun.management.jmxremote.need.client.auth (STRING, no default value)
```

```
    jmxremote.ssl.config.file : [optional] set com.sun.management.jmxremote.ssl_config_file (STRING, no default value)
```

```
    jmxremote.autodiscovery : [optional] set com.sun.management.jmxremote.autodiscovery (STRING, no default value)
```

```
    jdp.port : [optional] set com.sun.management.jdp.port (INT, no default value)
```

```
    jdp.address : [optional] set com.sun.management.jdp.address (STRING, no default value)
```

```
    jdp.source_addr : [optional] set com.sun.management.jdp.source_addr (STRING, no default value)
```



JCMD – 开启本地管理代理

```
[root@dev-yd-jyh01 ~]# jcmd 12379 help ManagementAgent.start_local
```

```
12379:
```

```
ManagementAgent.start_local
```

```
Start local management agent. #启动本地管理代理
```

```
Impact: Low: No impact #影响低
```

```
Syntax: ManagementAgent.start_local
```





JCMD – 停止远程管理代理

```
[root@dev-yd-jyh01 ~]# jcmd 12379 help ManagementAgent.stop
```

```
12379:
```

```
ManagementAgent.stop
```

```
Stop remote management agent. #停止远程代理
```

```
Impact: Low: No impact #影响低
```

```
Syntax: ManagementAgent.stop
```





使用jconsole和jvisualvm

使用jconsole、 jvisualvm 进行可视化监控

- 远程连接失败：
- 1.关闭防火墙（ CentOS为例 ）： firewall-cmd --state 查看防火墙状态、 systemctl stop firewalld.service 停止
- 2.修改hosts文件为真实IP（ vim /etc/hosts ） ， 使用hostname -i 查看
- 3.重启应用， 否则hosts修改了也没用
- 4.启动JMX监听： jcmd XXX端口 ManagementAgent.start jmxremote.port=7777 jmxremote.rmi.port=7777
jmxremote.ssl=false jmxremote.authenticate=false
- 5.使用jconsole、 jvisualvm使用ip+端口进行连接



使用jconsole与JMX结合演练



实操演练



使用jvisualvm与JMX结合演练



实操演练



使用jmc与JMX结合演练



实操演练



JFR

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



JCMD – 飞行记录

JFR.start

启动一个新的JFR录制



JFR.dump

将JFR录制内容转出为文件



JFR.check

检查正在运行的JFR录制



JFR.stop

停止一个JFR录制



Java Flight Recorder

记录一段时间内JVM的
发生的所有事，类似于
飞机上的黑匣子，深入
分析JVM的工具



JCMD – 启动飞行记录



```
[root@dev-yd-jyh01 ~]# jcmd 12379 help JFR.start
12379:
JFR.start
Starts a new JFR recording #开启一个新的JFR
```

Impact: Medium: Depending on the settings for a recording, the impact can range from low to high. #影响：中等：根据一次录制的设置不同，影响可以从低到高

Permission: java.lang.management.ManagementPermission(monitor)

Syntax : JFR.start [options]

Options: (options must be specified using the <key> or <key>=<value> syntax)

name : [optional] Name that can be used to identify recording, e.g. \"My Recording\" (STRING, no default value)

defaultrecording : [optional] Starts the default recording, can only be combined with settings. (BOOLEAN, false)

dumponexit : [optional] Dump running recording when JVM shuts down (BOOLEAN, no default value)

settings : [optional] Settings file(s), e.g. profile or default. See JRE_HOME/lib/jfr (STRING SET, no default value)

delay : [optional] Delay recording start with (s)econds, (m)inutes, (h)ours, or (d)ays, e.g. 5h. (NANOTIME, 0)

duration : [optional] Duration of recording in (s)econds, (m)inutes, (h)ours, or (d)ays, e.g. 300s. (NANOTIME, 0)

filename : [optional] Resulting recording filename, e.g. \"/home/user/My Recording.jfr\" (STRING, no default value)

compress : [optional] GZip-compress the resulting recording file (BOOLEAN, false)

maxage : [optional] Maximum time to keep recorded data (on disk) in (s)econds, (m)inutes, (h)ours, or (d)ays, e.g. 60m, or 0 for no limit (NANOTIME, 0)

maxsize : [optional] Maximum amount of bytes to keep (on disk) in (k)B, (M)B or (G)B, e.g. 500M, or 0 for no limit (MEMORY SIZE, 0)



JCMD – 检查运行中的JFR记录



```
[root@dev-yd-jyh01 ~]# jcmd 12379 help JFR.check
```

```
12379:
```

```
JFR.check
```

```
Checks running JFR recording(s) #检查运行中的飞行记录
```

```
Impact: Low #影响低
```

```
Permission: java.lang.management.ManagementPermission(monitor)
```

```
Syntax : JFR.check [options]
```

```
Options: (options must be specified using the <key> or <key>=<value> syntax)
```

```
    name : [optional] Recording name, e.g. \"My Recording\" or omit to see all recordings (STRING, no default value)
```

```
    recording : [optional] Recording number, or omit to see all recordings (JLONG, -1)
```

```
    verbose : [optional] Print event settings for the recording(s) (BOOLEAN, false)
```



JCMD – 转储JFR记录文件



```
[root@dev-yd-jyh01 ~]# jcmd 12379 help JFR.dump
```

```
12379:
```

```
JFR.dump
```

Copies contents of a JFR recording to file. Either the name or the recording id must be specified. #将JFR记录的内容复制到文件中。必须指定名称或录制ID。

Impact: Low #影响低

Permission: java.lang.management.ManagementPermission(monitor)

Syntax : JFR.dump [options]

Options: (options must be specified using the <key> or <key>=<value> syntax)

name : [optional] Recording name, e.g. \"My Recording\" (STRING, no default value)

recording : [optional] Recording number, use JFR.check to list available recordings (JLONG, -1)

filename : Copy recording data to file, i.e. \"/home/user/My Recording.jfr\" (STRING, no default value)

compress : [optional] GZip-compress "filename" destination (BOOLEAN, false)



JCMD – 停止JFR录制

```
[root@dev-yd-jyh01 ~]# jcmd 12379 help JFR.stop
```

```
12379:
```

```
JFR.stop
```

```
Stops a JFR recording #停止一个JFR录制
```

```
Impact: Low #影响低
```

```
Permission: java.lang.management.ManagementPermission(monitor)
```

```
Syntax : JFR.stop [options]
```

```
Options: (options must be specified using the <key> or <key>=<value> syntax)
```

```
name : [optional] Recording name,.e.g \"My Recording\" (STRING, no default value)
```

```
recording : [optional] Recording number, see JFR.check for a list of available recordings (JLONG, -1)
```

```
discard : [optional] Skip writing data to previously specified file (if any) (BOOLEAN, false)
```

```
filename : [optional] Copy recording data to file, e.g. \"/home/user/My Recording.jfr\" (STRING, no default value)
```

```
compress : [optional] GZip-compress "filename" destination (BOOLEAN, false)
```





JCMD – JFR操作演练



实操演练



JPS

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



JPS

- JPS的全称java Virtual Machine Process Status Tool，用来查看JVM状态的工具

```
[root@dev~]# jps -h
usage: jps [-help]
       jps [-q] [-mlvV] [<hostid>]
```



Definitions:

<hostid>: <hostname>[:<port>]

- 当不指定hostid时，默认查看本机jvm进程，当指定指定的hostid机器上的jvm进程，此时hostid所指机器必须开启statd服务。
- 针对java进程，我们可以放弃操作系统提供的ps命令

```
[root@dev-yd-jyh01 ~]# jps
4632 Bootstrap
3481 Jps
```



- 不使用任何参数，列出PID和简单的class或者jar名称



JPS

```
[root@dev-yd-jyh01 ~]# jps -q  
4632  
4139
```



- jps -q 只显示pid，而不显示其他信息，功能十分鸡肋，对于写shell脚本，读取pid或许有一些作用

```
[root@dev-yd-jyh01 ~]# jps -m  
4632 Bootstrap start start  
4735 Jps -m
```



- jps -m 显示pid，以及传入main函数的参数内容

```
[root@dev-yd-jyh01 ~]# jps -l  
5204 sun.tools.jps.Jps  
4632 org.apache.catalina.startup.Bootstrap
```



- jps -l 显示pid，以及main方法的完整路径或者jar包名称，常用

```
[root@dev-yd-jyh01 ~]# jps -v  
8664 RemoteMavenServer -Djava.awt.headless=true -Didea.version==2019.1.3 -Xmx768m -Didea.maven.embed  
der.version=3.3.9 -Dfile.encoding=UTF-8
```

- jps -v 显示pid，以及vm参数；感觉唯一有点用的方法



JPS&JSTATD

```
[root@dev-yd-jyh01 ~]# jps -lvm
7377 sun.tools.jps.Jps -lvm -Denv.class.path=./usr/java/jdk1.8.0_45/lib/dt.jar:/usr/java/jdk1.8.0_45/lib/tools.jar -
Dapplication.home=/usr/java/jdk1.8.0_45 -Xms8m
4632 org.apache.catalina.startup.Bootstrap start start -
Djava.util.logging.config.file=/data/tomcat/tomcat_jyh_app/conf/logging.properties
```



- jps -lvm 可以把多个选项叠加使用。如果使用-q，则-q优先生效

```
C:\Users\Administrator>jps -lvm 10.56.20.20
RMI Registry not available at 10.56.20.20:1099
Connection refused to host: 10.56.20.20; nested exception is:
    java.net.ConnectException: Connection refused: connect //无法链接, 没有开启statd服务
```



- jps命令如果输入远程服务器ip则可以查看远程服务器的jvm进程，但是远程服务必须开启jstatd服务监听。



JPS&JSTATD&jvisualvm

- 在\$JAVA_HOME/jre/lib/security文件夹下创建statd.all.policy文件，文件内容如下

```
grant codebase "file:${java.home}/../lib/tools.jar" {  
    permission java.security.AllPermission;  
};
```



- jstatd -J-Djava.security.policy=jstatd.all.policy 开启jstatd远程服务，默认端口1099。服务端处于阻塞状态
- jstatd -J-Djava.security.policy=jstatd.all.policy & 进入daemon模式
- jstatd -J-Djava.security.policy=jstatd.all.policy -p 9999 使用9999端口
- 使用jvisualvm工具进行连接，但是监控内容有缺失，感觉比较鸡肋
- 使用jps进行远程进程查看



JSTAT

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



jstat

- jstat全称Java Virtual Machine statistics monitoring tool 用于对JVM的资源性能进行实时监控

```
[D:\~]$ jstat //jstat -h jstat -help 3者效果相同
```

-h requires an integer argument

Usage: jstat -help|-options

```
jstat -<option> [-t] [-h<lines>] <vmid> [<interval> [<count>]] #-t可以在打印的列加上Timestamp列，用于显示系统运行的时间
```



Definitions:

<option> An option reported by the -options option //选择报告项目

<vmid> Virtual Machine Identifier. A vmid takes the following form:

```
<lvmid>[@<hostname>[:<port>]]
```

虚拟机标识符。vmid采用以下形式：

<lvmid>[@<hostname>[:<port>]]其中<lvmid>是目标的本地虚拟机标识符Java虚拟机，通常是进程ID；

Where <lvmid> is the local vm identifier for the target

Java virtual machine, typically a process id; <hostname> is

the name of the host running the target Java virtual machine;

and <port> is the port number for the rmiregistry on the

target host. See the jvmsat documentation for a more complete

description of the Virtual Machine Identifier.

<lines> Number of samples between header lines.

每隔多少行输出一次头部信息

<interval> Sampling interval. The following forms are allowed:

多少秒采样一次

```
<n>["ms"|"s"]
```

Where <n> is an integer and the suffix specifies the units as

milliseconds("ms") or seconds("s"). The default units are "ms".

<count> Number of samples to take before terminating.

在结束之前采样多少次

-J<flag> Pass <flag> directly to the runtime system.

直接向虚拟机传入参数

Kevin)

版权所有 侵权必究



jstat可以监控哪些信息

[D:\~]\$ jstat -options #查看jstat都支持查看哪些内容

-class #查看类的加载情况

-compiler #编译任务执行数量

-gc #gc相关的堆信息，查看gc的次数

-gccapacity #VM内存中三代（ young,old,perm ）对象的使用和占用大小

-gccause #显示垃圾回收的相关统计信息,同时显示最后一次或当前正在发生的垃圾回收的诱因。

-gcmetacapacity #metaspace 中对象的信息及其占用量。

-gcnew #年轻代对象的信息。

-gcnewcapacity #年轻代对象的信息及其占用量

-gcold #old代对象的信息

-gcoldcapacity #old代对象的信息及其占用量

-gcutil #统计gc信息

-printcompilation #输出JIT编译的方法信息





Jstat -class 与 jstat -compiler

```
[D:\~]$ jstat -class -t -h10 1204 ls 100
Timestamp    Loaded Bytes    Unloaded Bytes    Time
783.6        10207 18502.9      1      0.9      8.65
784.6        10207 18502.9      1      0.9      8.65
785.6        10207 18502.9      1      0.9      8.65
786.6        10207 18502.9      1      0.9      8.65
787.6        10207 18502.9      1      0.9      8.65
788.6        10207 18502.9      1      0.9      8.65
789.6        10207 18502.9      1      0.9      8.65
790.6        10207 18502.9      1      0.9      8.65
791.6        10207 18502.9      1      0.9      8.65
792.6        10207 18502.9      1      0.9      8.65
```

- Loaded 已经加载多少个类
- Bytes 已经加载类占用字节数
- Unloaded 已经卸载多少个类
- Bytes 已经卸载类占用字节数
- Time 装载和卸载类所花费的时间

```
[D:\~]$ jstat -compiler -t -h10 1204 ls 100
Timestamp    Compiled Failed Invalid    Time FailedType FailedMethod
961.8         3825     0      0      1.48      0
962.8         3825     0      0      1.48      0
963.8         3825     0      0      1.48      0
964.8         3825     0      0      1.48      0
965.8         3825     0      0      1.48      0
966.8         3825     0      0      1.48      0
967.8         3825     0      0      1.48      0
968.8         3825     0      0      1.48      0
969.8         3825     0      0      1.48      0
970.8         3825     0      0      1.48      0
```

- Compiled : 编译任务执行数量
- Failed : 编译任务执行失败数量
- Invalid : 编译任务执行失效数量
- Time : 编译任务消耗时间
- FailedType : 最后一个编译失败任务的类型
- FailedMethod : 最后一个编译失败任务所在的类及方法



Jstat -gc

```
[D:\~]$ jstat -gc -t -h10 1204 1s 100
Timestamp      S0C      S1C      S0U      S1U      EC      EU      OC      OU      MC      MU      CCSC      CCSU      YGC      YGCT      FGC      FGCT      GCT
1182.7 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
1183.7 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
1184.7 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
1185.8 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
1186.7 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
1187.8 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
1188.7 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
1189.8 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
1190.7 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
1191.8 6144.0 8192.0 6120.1 0.0 17408.0 6574.2 61440.0 17060.7 51368.0 48690.8 7336.0 6847.4 30 0.315 2 0.228 0.543
```

- S0C 年轻代中第一个survivor (幸存区) 的容量
- S1C 年轻代中第二个survivor (幸存区) 的容量
- S0U 年轻代中第一个survivor (幸存区) 目前已使用空间
- S1U 年轻代中第二个survivor (幸存区) 目前已使用空间
- EC 年轻代中Eden (伊甸园) 的容量
- EU 年轻代中Eden (伊甸园) 目前已使用空间
- OC Old代的容量
- OU : Old代目前已使用空间
- MC : (元空间)的容量
- MU : metmetaspaceaspace(元空间)目前已使用空间
- YGC : 从应用程序启动到采样时年轻代中gc次数
- YGCT : 从应用程序启动到采样时年轻代中gc所用时间(s)
- FGC : 从应用程序启动到采样时old代(full gc)gc次数
- FGCT : 从应用程序启动到采样时old代(full gc)gc所用时间(s)
- GCT : 从应用程序启动到采样时gc用的总时间(s)



jstat -gccapacity分析

```
[root@dev-yd-jyh01 security]# jstat -gccapacity -h10 15836 1s 100
```

NGCMN	NGCMX	NGC	S0C	S1C	EC	OGCMN	OGCMX	OGC	OC	MCMN	MCMX	MC	CCSMN	CCSMX	CCSC	YGC	FGC
786432.0	786432.0	786432.0	131072.0	15360.0	589824.0	1310720.0	1310720.0	1310720.0	1310720.0	0.0	1085440.0	40368.0	0.0	1048576.0	4784.0	5	2
786432.0	786432.0	786432.0	131072.0	15360.0	589824.0	1310720.0	1310720.0	1310720.0	1310720.0	0.0	1085440.0	40368.0	0.0	1048576.0	4784.0	5	2
786432.0	786432.0	786432.0	131072.0	15360.0	589824.0	1310720.0	1310720.0	1310720.0	1310720.0	0.0	1085440.0	40368.0	0.0	1048576.0	4784.0	5	2
786432.0	786432.0	786432.0	131072.0	15360.0	589824.0	1310720.0	1310720.0	1310720.0	1310720.0	0.0	1085440.0	40368.0	0.0	1048576.0	4784.0	5	2

- NGCMN:年轻代(young)中初始化(最小)的大小(字节)
- NGCMX:年轻代(young)的最大容量(字节)
- NGC:年轻代(young)中当前的容量(字节)
- S0C : 年轻代中第一个survivor (幸存区) 的容量(字节)
- S1C : 年轻代中第二个survivor (幸存区) 的容量(字节)
- EC:年轻代中Eden (伊甸园) 的容量(字节)
- OGCMN:old代中初始化(最小)的大小(字节)
- OGCMX:old代的最大容量(字节)
- OGC : old代当前新生成的容量(字节)
- OC:Old代的容量(字节)
- MCMN : metaspace(元空间)中初始化(最小)的大小(字节)
- MCMX : metaspace(元空间)的最大容量(字节)
- MC : metaspace(元空间)当前新生成的容量(字节)
- CCSMN : 最小压缩类空间大小
- CCSMX : 最大压缩类空间大小
- CCSC : 当前压缩类空间大小
- YGC:从应用程序启动到采样时年轻代中gc次数
- FGC : 从应用程序启动到采样时old代(全gc)gc次数



jstat -gcutil分析

```
[root@dev-yd-jyh01 security]# jstat -gcutil -h10 15836 1s 100
S0      S1      E      O      M      CCS      YGC      YGCT      FGC      FGCT      GCT
0.00    97.41    8.98    6.26    96.99    95.21      5      0.481      2      0.395    0.876
0.00    97.41    8.98    6.26    96.99    95.21      5      0.481      2      0.395    0.876
0.00    97.41    8.98    6.26    96.99    95.21      5      0.481      2      0.395    0.876
0.00    97.41    8.98    6.26    96.99    95.21      5      0.481      2      0.395    0.876
0.00    97.41    8.98    6.26    96.99    95.21      5      0.481      2      0.395    0.876
```

- S0：年轻代中第一个survivor（幸存区）已使用的占当前容量百分比
- S1：年轻代中第二个survivor（幸存区）已使用的占当前容量百分比
- E：年轻代中Eden（伊甸园）已使用的占当前容量百分比
- O：old代已使用的占当前容量百分比
- P：perm代已使用的占当前容量百分比
- YGC：从应用程序启动到采样时年轻代中gc次数
- YGCT：从应用程序启动到采样时年轻代中gc所用时间(s)
- FGC：从应用程序启动到采样时old代(全gc)gc次数
- FGCT：从应用程序启动到采样时old代(全gc)gc所用时间(s)
- GCT：从应用程序启动到采样时gc用的总时间(s)



jstat -gccause分析

```
[root@dev-yd-jyh01 security]# jstat -gccause -h10 15836 1s 100
```

S0	S1	E	O	M	CCS	YGC	YGCT	FGC	FGCT	GCT	LGCC	GCC
0.00	97.41	7.53	6.26	96.99	95.21	5	0.481	2	0.395	0.876	Allocation Failure	No GC
0.00	97.41	7.53	6.26	96.99	95.21	5	0.481	2	0.395	0.876	Allocation Failure	No GC
0.00	97.41	7.53	6.26	96.99	95.21	5	0.481	2	0.395	0.876	Allocation Failure	No GC
0.00	97.41	7.53	6.26	96.99	95.21	5	0.481	2	0.395	0.876	Allocation Failure	No GC

- 显示垃圾回收的相关信息（通-gcutil），同时显示最后一次或当前正在发生的垃圾回收的诱因。
- 最后两列如下：
- LGCC：最后一次GC原因
- GCC：当前GC原因（No GC 为当前没有执行GC）



jstat -gcmetacapacity分析

```
[root@dev-yd-jyh01 security]# jstat -gcmetacapacity -h10 15836 1s 100
```

MCMN	MCMX	MC	CCSMN	CCSMX	CCSC	YGC	FGC	FGCT	GCT
0.0	1085440.0	40368.0	0.0	1048576.0	4784.0	5	2	0.395	0.876
0.0	1085440.0	40368.0	0.0	1048576.0	4784.0	5	2	0.395	0.876
0.0	1085440.0	40368.0	0.0	1048576.0	4784.0	5	2	0.395	0.876
0.0	1085440.0	40368.0	0.0	1048576.0	4784.0	5	2	0.395	0.876

- MCMN:最小元数据容量
- MCMX : 最大元数据容量
- MC : 当前元数据空间大小
- CCSMN : 最小压缩类空间大小
- CCSMX : 最大压缩类空间大小
- CCSC : 当前压缩类空间大小
- YGC : 从应用程序启动到采样时年轻代中gc次数
- FGC : 从应用程序启动到采样时old代(全gc)gc次数
- FGCT : 从应用程序启动到采样时old代(全gc)gc所用时间(s)
- GCT : 从应用程序启动到采样时gc用的总时间(s)



jstat -gcnew分析

```
[root@dev-yd-jyh01 security]# jstat -gcnew -h10 15836 1s 100
S0C    S1C    S0U    S1U    TT MTT    DSS    EC    EU    YGC    YGCT
131072.0 15360.0    0.0 14961.7 8    15 131072.0 589824.0 47605.6    5    0.481
131072.0 15360.0    0.0 14961.7 8    15 131072.0 589824.0 47605.6    5    0.481
131072.0 15360.0    0.0 14961.7 8    15 131072.0 589824.0 47605.6    5    0.481
131072.0 15360.0    0.0 14961.7 8    15 131072.0 589824.0 47605.6    5    0.481
131072.0 15360.0    0.0 14961.7 8    15 131072.0 589824.0 47605.6    5    0.481
```

- S0C : 年轻代中第一个survivor (幸存区) 的容量(字节)
- S1C : 年轻代中第二个survivor (幸存区) 的容量(字节)
- S0U : 年轻代中第一个survivor (幸存区) 目前已使用空间(字节)
- S1U : 年轻代中第二个survivor (幸存区) 目前已使用空间(字节)
- TT : 持有次数限制
- MTT : 最大持有次数限制

- DSS : 期望的幸存区大小
- EC : 年轻代中Eden (伊甸园) 的容量(字节)
- EU : 年轻代中Eden (伊甸园) 目前已使用空间(字节)
- YGC : 从应用程序启动到采样时年轻代中gc次数
- YGCT : 从应用程序启动到采样时年轻代中gc所用时间(s)



jstat -gcnewcapacity分析

```
[root@dev-yd-jyh01 security]# jstat -gcnewcapacity -h10 15836 1s 100
NGCMN    NGCMX      NGC      S0CMX      S0C      S1CMX      S1C      ECMX      EC      YGC      FGC
786432.0  786432.0    786432.0 262144.0 131072.0 262144.0 15360.0 785408.0 589824.0 5      2
786432.0  786432.0    786432.0 262144.0 131072.0 262144.0 15360.0 785408.0 589824.0 5      2
786432.0  786432.0    786432.0 262144.0 131072.0 262144.0 15360.0 785408.0 589824.0 5      2
```

- NGCMN : 年轻代(young)中初始化(最小)的大小(字节)
- NGCMX : 年轻代(young)的最大容量(字节)
- NGC : 年轻代(young)中当前的容量(字节)
- S0CMX : 年轻代中第一个survivor (幸存区) 的最大容量(字节)
- S0C : 年轻代中第一个survivor (幸存区) 的容量(字节)
- S1CMX : 年轻代中第二个survivor (幸存区) 的最大容量(字节)
- S1C : 年轻代中第二个survivor (幸存区) 的容量(字节)
- ECMX : 年轻代中Eden (伊甸园) 的最大容量(字节)
- EC : 年轻代中Eden (伊甸园) 的容量(字节)
- YGC : 从应用程序启动到采样时年轻代中gc次数
- FGC : 从应用程序启动到采样时old代(全gc)gc次数



jstat -gcold分析

```
[root@dev-yd-jyh01 security]# jstat -gcold -h10 15836 1s 100
```

MC	MU	CCSC	CCSU	OC	OU	YGC	FGC	FGCT	GCT
40368.0	39151.6	4784.0	4554.6	1310720.0	82051.3	5	2	0.395	0.876
40368.0	39151.6	4784.0	4554.6	1310720.0	82051.3	5	2	0.395	0.876
40368.0	39151.6	4784.0	4554.6	1310720.0	82051.3	5	2	0.395	0.876

- MC : metaspace(元空间)的容量(字节)
- MU : metaspace(元空间)目前已使用空间(字节)
- CCSC:压缩类空间大小
- CCSU:压缩类空间使用大小
- OC : Old代的容量(字节)
- OU : Old代目前已使用空间(字节)
- YGC : 从应用程序启动到采样时年轻代中gc次数
- FGC : 从应用程序启动到采样时old代(全gc)gc次数
- FGCT : 从应用程序启动到采样时old代(全gc)gc所用时间(s)
- GCT : 从应用程序启动到采样时gc用的总时间(s)



jstat -gcoldcapacity分析

```
[root@dev-yd-jyh01 security]# jstat -gcoldcapacity -h10 15836 1s 100
  OGCMN      OGCMX      OGC        OC      YGC    FGC    FGCT      GCT
1310720.0 1310720.0 1310720.0 1310720.0    5     2    0.395    0.876
1310720.0 1310720.0 1310720.0 1310720.0    5     2    0.395    0.876
1310720.0 1310720.0 1310720.0 1310720.0    5     2    0.395    0.876
```

- OGCMN : old代中初始化(最小)的大小(字节)
- OGCMX : old代的最大容量(字节)
- OGC : old代当前新生成的容量(字节)
- OC : Old代的容量(字节)
- YGC : 从应用程序启动到采样时年轻代中gc次数
- FGC : 从应用程序启动到采样时old代(全gc)gc次数
- FGCT : 从应用程序启动到采样时old代(全gc)gc所用时间(s)
- GCT : 从应用程序启动到采样时gc用的总时间(s)



jstat -printcompilation分析

```
[root@dev-yd-jyh01 security]# jstat -printcompilation -h10 15836 1s 100
Compiled   Size   Type Method
    4918      5     1 com/mysql/jdbc/ConnectionImpl supportsQuotedIdentifiers
    4918      5     1 com/mysql/jdbc/ConnectionImpl supportsQuotedIdentifiers
    4918      5     1 com/mysql/jdbc/ConnectionImpl supportsQuotedIdentifiers
    4918      5     1 com/mysql/jdbc/ConnectionImpl supportsQuotedIdentifiers
```

- Compiled：编译任务的数目
- Size：方法生成的字节码的大小
- Type：编译类型
- Method：类名和方法名用来标识编译的方法。类名使用/做为一个命名空间分隔符。方法名是给定类中的方法。上述格式是由-XX:+PrintCompilation选项进行设置的



JMAP 与 JHAT

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



jmap -h

```
[root@dev-yd-jyh01 security]# jmap -h
```

Usage:

jmap [option] <pid>

(to connect to running process)

jmap [option] <executable <core>

(to connect to a core file)

jmap [option] [server_id@]<remote server IP or hostname>

(to connect to remote debug server)

where <option> is one of:

<none> to print same info as Solaris pmap

-heap to print java heap summary

-histo[:live] to print histogram of java object heap; if the "live" suboption is specified, only count live objects

-clstats to print class loader statistics

-finalizerinfo to print information on objects awaiting finalization

-dump:<dump-options> to dump java heap in hprof binary format dump-options: live dump only live objects; if not specified, all objects in the heap are dumped. format=b binary format

file=<file> dump heap to <file> Example: jmap -dump:live,format=b,file=heap.bin <pid>

-F force. Use with -dump:<dump-options> <pid> or -histo to force a heap dump or histogram when <pid> does not respond. The "live" suboption is not supported in this mode.

-h | -help to print this help message

-J<flag> to pass <flag> directly to the runtime system



jmap

```
[root@dev-yd-jyh01 security]# jmap 15836
Attaching to process ID 15836, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.45-b02
0x0000000000400000      7K      /usr/java/jdk1.8.0_45/bin/java
0x00007f11a4462000     86K      /usr/lib64/libgcc_s-4.8.5-20150702.so.1
0x00007f11a4678000    250K      /usr/java/jdk1.8.0_45/jre/lib/amd64/libsunec.so
0x00007f11a4dc2000     90K      /usr/java/jdk1.8.0_45/jre/lib/amd64/libnio.so
0x00007f11a4fd3000     48K      /usr/java/jdk1.8.0_45/jre/lib/amd64/libmanagement.so
0x00007f11a53bd000    113K      /usr/java/jdk1.8.0_45/jre/lib/amd64/libnet.so
0x00007f11f334d000    121K      /usr/java/jdk1.8.0_45/jre/lib/amd64/libzip.so
0x00007f11f3568000     56K      /usr/lib64/libnss_files-2.17.so
0x00007f11f377a000    220K      /usr/java/jdk1.8.0_45/jre/lib/amd64/libjava.so
0x00007f11f39a6000     64K      /usr/java/jdk1.8.0_45/jre/lib/amd64/libverify.so
0x00007f11f3bb5000     43K      /usr/lib64/librt-2.17.so
0x00007f11f3dbd000    1114K      /usr/lib64/libm-2.17.so
0x00007f11f40bf000   16440K      /usr/java/jdk1.8.0_45/jre/lib/amd64/server/libjvm.so
0x00007f11f5087000    2058K      /usr/lib64/libc-2.17.so
0x00007f11f5448000     19K      /usr/lib64/libdl-2.17.so
0x00007f11f564c000    100K      /usr/java/jdk1.8.0_45/lib/amd64/jli/libjli.so
0x00007f11f5862000    138K      /usr/lib64/libpthread-2.17.so
0x00007f11f5a7e000    160K      /usr/lib64/ld-2.17.so
```

- 使用不带选项参数的jmap打印共享对象映射，将会打印目标虚拟机中加载的每个共享对象的起始地址、映射大小以及共享对象文件的路径全称。



jmap -heap

```
[root@dev-yd-jyh01 security]# jmap -heap 15836
Attaching to process ID 15836, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.45-b02

using thread-local object allocation.
Parallel GC with 8 thread(s)

Heap Configuration:
  MinHeapFreeRatio      = 0
  MaxHeapFreeRatio      = 100
  MaxHeapSize           = 2147483648 (2048.0MB)
  NewSize               = 805306368 (768.0MB)
  MaxNewSize            = 805306368 (768.0MB)
  OldSize               = 1342177280 (1280.0MB)
  NewRatio              = 2
  SurvivorRatio         = 8
  MetaspaceSize         = 21807104 (20.796875MB)
  CompressedClassSpaceSize = 1073741824 (1024.0MB)
  MaxMetaspaceSize      = 17592186044415 MB
  G1HeapRegionSize      = 0 (0.0MB)
```

```
Heap Usage:
PS Young Generation
Eden Space:
  capacity = 603979776 (576.0MB)
  used     = 75087912 (71.60941314697266MB)
  free     = 528891864 (504.39058685302734MB)
  12.432189782460531% used
From Space:
  capacity = 15728640 (15.0MB)
  used     = 15320800 (14.611053466796875MB)
  free     = 407840 (0.388946533203125MB)
  97.40702311197917% used
To Space:
  capacity = 134217728 (128.0MB)
  used     = 0 (0.0MB)
  free     = 134217728 (128.0MB)
  0.0% used
PS Old Generation
  capacity = 1342177280 (1280.0MB)
  used     = 84020544 (80.12823486328125MB)
  free     = 1258156736 (1199.8717651367188MB)
  6.260018348693848% used

21638 interned Strings occupying 2538088 bytes.
```

- 打印堆的信息、以及新生代、老年代、幸存区的内存容量和使用情况。
- -XX:MinHeapFreeRatio=<n> 指定 jvm heap 在使用率小于 n 的情况下,heap 进行收缩,Xmx==Xms 的情况下无效,如 :-XX:MinHeapFreeRatio=30 j: -XX:MaxHeapFreeRatio=<n> 指定 jvm heap 在使用率大于 n 的情况下,heap 进行扩张,Xmx==Xms 的情况下无效,如 :-XX:MaxHeapFreeRatio=70



jmap -heap 内存划分

NewRatio : 老年代与新生代的比值

- Old : New = 2 , 则老年代占整个Heap的2/3 , 新生代占1/3

SurvivorRatio : Eden与一个Survivor的比值

- Eden : 2Survivor = 8:2 , 则Eden占整个New的8/10 , 1个Survivor占New的1/10



jmap -histo:live

num	#instances	#bytes	class name
1:	204070	64185968	[B
2:	58602	29843984	[I
3:	218681	22881392	[C
4:	262144	20971520	org.apache.logging.log4j.core.async.RingBufferLogEvent
5:	262144	6291456	org.apache.logging.log4j.core.async.AsyncLoggerConfigHelper\$Log4jEventWrapper
6:	203503	4884072	java.lang.String
7:	24922	3499880	[Ljava.lang.Object;
8:	29346	2817216	java.util.jar.JarFile\$JarFileEntry
9:	73347	2347104	java.util.HashMap\$Node
10:	55833	2233320	java.util.TreeMap\$Entry
11:	47459	1898360	java.util.HashMap\$KeyIterator
12:	56936	1821952	java.io.File
13:	44418	1776720	java.util.HashMap\$ValueIterator
14:	14783	1300904	java.lang.reflect.Method
15:	11090	1242080	java.net.SocksSocketImpl
16:	44419	1066056	org.apache.catalina.LifecycleEvent
17:	3094	1017224	[Ljava.util.HashMap\$Node;
18:	61303	980848	java.lang.Object
19:	13459	969048	java.util.regex.Pattern

- Jstat -histo:live 显示堆中的对象统计信息，其中包括每个Java类、对象数量、内存大小(单位：字节)、完全限定的类名。打印的虚拟机内部的类名称将会带有一个 '*' 前缀。如果指定了live子选项，则只计算活动的对象。



jmap -finalizerinfo

```
[root@dev-yd-jyh01 security]# jmap -finalizerinfo 15836
Attaching to process ID 15836, please wait...
Debugger attached successfully.
Server compiler detected.
JVM version is 25.45-b02
Number of objects pending for finalization: 0
```

- 示在F-Queue队列等待Finalizer线程执行finalizer方法的对象
- Number of objects pending for finalization: 0 说明当前F-QUEUE队列中并没有等待Finalizer线程执行final



jmap -dump:live,format=b,file=temp.hprof

```
[root@dev-yd-jyh01 security]# jmap -dump:live,format=b,file=jyh.hprof 24399
Dumping heap to /usr/java/jdk1.8.0_45/jre/lib/security/jyh.hprof ...
Heap dump file created
```

- 以hprof二进制格式转储Java堆到指定filename的文件中。live子选项是可选的。如果指定了live子选项，堆中只有活动的对象会被转储。想要浏览heap dump，你可以使用jhat(Java堆分析工具)读取生成的文件
- 执行的过程中为了保证dump的信息是可靠的，会暂停应用，生产系统不推荐使用。



jhat xx.hprof

```
D:\>jhat abc.hprof
Reading from abc.hprof...
Dump file created Thu Aug 15 12:02:34 CST 2019
Snapshot read, resolving...
Resolving 1330228 objects...
Chasing references, expect 266 dots.....
.....
.
Eliminating duplicate references.....
.....
Snapshot resolved.
Started HTTP server on port 7000
Server is ready.
```

- Java Heap Analyse Tool
- 是用来分析java堆的命令，可以将堆中的对象以html的形式显示出来，包括对象的数量，大小等等，并支持对象查询语。好处是不用把文件导入下来，直接可以从浏览器访问7000端口即可使用。
- 支持OQL（Object Query Language 对象查询语言）像使用sql一样查询对象
- 如果堆文件很大，则解析速度很慢，并且在这个过程中会耗费很大的内存。
- 由于快照文件一般都很大，所以即使解析完毕，在浏览器中查看也十分卡顿；提供的功能很少，比较鸡肋。



JSTACK

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



jstack

```
[root@dev-yd-jyh01 security]# jstack -h
Usage:
  jstack [-l] <pid>
          (to connect to running process)
  jstack -F [-m] [-l] <pid>
          (to connect to a hung process)
  jstack [-m] [-l] <executable> <core>
          (to connect to a core file)
  jstack [-m] [-l] [server_id@]<remote server IP or hostname>
          (to connect to a remote debug server)

Options:
  -F  to force a thread dump. Use when jstack <pid> does not respond (process is hung)
  -m  to print both java and native frames (mixed mode)
  -l  long listing. Prints additional information about locks
  -h or -help to print this help message
```

- jstack可以查看或导出 Java 应用程序中线程堆栈信息。
- **参数说明：**
- -l 长列表. 打印关于锁的附加信息,例如属于java.util.concurrent 的 ownable synchronizers列表.
- -F 当' jstack [-l] pid'没有相应的时候强制打印栈信息
- -m 打印java和native c/c++框架的所有栈信息.
- -h | -help 打印帮助信息



jstack自动死锁检测

Found one Java-level deadlock:

=====

"Thread-1":

waiting for `ownable synchronizer 0x00000000dd48c530`, (a `java.util.concurrent.locks.ReentrantLock$NonfairSync`),
which is held by "Thread-0"

"Thread-0":

waiting for `ownable synchronizer 0x00000000dd48c560`, (a `java.util.concurrent.locks.ReentrantLock$NonfairSync`),
which is held by "Thread-1"

Java stack information for the threads listed above:

=====

"Thread-1":

```
at sun.misc.Unsafe.park(Native Method)
- parking to wait for <0x00000000dd48c530> (a java.util.concurrent.locks.ReentrantLock$NonfairSync)
at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
at java.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(AbstractQueuedSynchronizer.java:836)
at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquireQueued(AbstractQueuedSynchronizer.java:870)
at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:1199)
at java.util.concurrent.locks.ReentrantLock$NonfairSync.lock(ReentrantLock.java:209)
at java.util.concurrent.locks.ReentrantLock.lock(ReentrantLock.java:285)
at com.mimaxueyuan.jvm.jstack.DeadLockDemo$2.run(DeadLockDemo.java:34)
at java.lang.Thread.run(Thread.java:748)
```

"Thread-0":

```
at sun.misc.Unsafe.park(Native Method)
- parking to wait for <0x00000000dd48c560> (a java.util.concurrent.locks.ReentrantLock$NonfairSync)
at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
at java.util.concurrent.locks.AbstractQueuedSynchronizer.parkAndCheckInterrupt(AbstractQueuedSynchronizer.java:836)
at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquireQueued(AbstractQueuedSynchronizer.java:870)
at java.util.concurrent.locks.AbstractQueuedSynchronizer.acquire(AbstractQueuedSynchronizer.java:1199)
at java.util.concurrent.locks.ReentrantLock$NonfairSync.lock(ReentrantLock.java:209)
at java.util.concurrent.locks.ReentrantLock.lock(ReentrantLock.java:285)
at com.mimaxueyuan.jvm.jstack.DeadLockDemo$1.run(DeadLockDemo.java:19)
at java.lang.Thread.run(Thread.java:748)
```

Found 1 deadlock.

● 死锁检测Demo : `com.mimaxueyuan.jvm.jstack.DeadLockDemo`



JINFO

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



jinfo -h

```
[root@dev-yd-jyh01 security]# jinfo
Usage:
  jinfo [option] <pid>
        (to connect to running process)
  jinfo [option] <executable <core>
        (to connect to a core file)
  jinfo [option] [server_id@]<remote server IP or hostname>
        (to connect to remote debug server)

where <option> is one of:
  -flag <name>          to print the value of the named VM flag
  -flag [+|-]<name>      to enable or disable the named VM flag
  -flag <name>=<value>   to set the named VM flag to the given value
  -flags                to print VM flags
  -sysprops              to print Java system properties
  <no option>            to print both of the above
  -h | -help            to print this help message
```

- -flag <name> 打印指定name的 JVM flag
- -flag [+|-]<name> 启用或者禁用指定name的 JVM flag
- -flag <name>=<value> 设置指定名称的name flag 为给定的值，很多参数是不允许调整的
- -flags 打印 JVM flags
- -sysprops 打印Java 系统属性
- <no option> 不指定任何option，则打印出所有的JVM flags和sysprops
- -h | -help 打印帮助信息



远程调用

- `jsadebugd pid server-id`
- 开启RMI远程服务, `jmap`、`jinfo`、`jstack`作为RMI客户端使用



性能调优及故障排除工具总结

工具	说明	重要程度
jcmd	最全面，基本涵盖所有功能，但是对于gc的跟踪缺失	强大
jconsole	配合JMX进行监控	可视化
jvisualvm	配合JMX或JSTATD服务监控	可视化
jmc	配合JMX监控及预警、配合JFR做分析	可视化、强
jps	查看本机和远程的jvm进行,支持远程调用	常用
jinfo	查看jvm信息、动态修改flag参数,支持远程调用	与jcmd选一
jstat	对于gc有强大的跟踪监视能力,支持远程调用	重要
jstack	打印线程堆栈、可以检测死锁（jcmd覆盖）支持远程调用	与jcmd选一
jmap	打印堆、实例、导出dump（jcmd覆盖）支持远程调用	与jcmd选一
jhat	解析Heap dump快照，使用http服务器方式展示	鸡肋
jstatd	开启服务端jstat监听	一般



Arthas

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



Arthas资料

- 官网地址：
- <https://alibaba.github.io/arthas/index.html>



故障排查

尹洪亮

Kevin.Yin

尹洪亮(Kevin)
版权所有 侵权必究



CPU占用过高-故障排查步骤

错误的程序引起的占用过高

- 使用top命令查看进程状态，查看CPU和内存的占用情况是否异常
- top命令查看cpu高的进程，然后使用top -H -p PID命令查看这个进程下的所有线程
- 如果所有线程的占用比例相差不大，并且都比较低，则是线程数量过大（启用线程过多或者请求量过大）考虑水平或垂直扩展。
- 如果是某一个线程占用比例特别大，则需要找到具体是哪个线程，记录TID，将TID转为十六进制（`printf "%x\n" TID`），这个16进制的数字就是JVM线程堆栈中的nid。
- 使用nid再去查看jstack -l PID，在线程堆栈中找出对应nid的线程，从而定位问题产生的位置。
- 也可以使用arthas直接定位问题

频繁GC引起的占用过高，与内存泄漏有关

- 结合jstat命令查看gc收集情况
- 三种CPU过高情况排查、图形化观察GC是否正常，请参见实操课程视频



GC日志查看工具

GCViewer

- github地址:<https://github.com/chewiebug/GCViewer> 需要自己打包
- 可以直接下载现成的jar,地址: <http://sourceforge.net/projects/gcviewer/files/gcviewer-1.35-SNAPSHOT.jar/download>
- 除了这个工具还有很多,例如GCLogViewer-0.3-win, google出品的工具,但是只支持32位JRE, 所以不建议采用。
- <https://gceasy.io/> 在线分析工具无需下载
- 都需要何止一些gc日志参数: `-verbose.gc -Xloggc:gc.log -XX:+PrintGCApplicationStoppedTime -XX:+PrintGCTimeStamps -XX:+PrintGCDetails`