Java dump

Last Updated: 2025-02-14

Java™ dumps, sometimes referred to as *Java cores*, are produced when the VM ends unexpectedly because of an operating system signal, `OutOfMemoryEr`
`Xdump:java` option on the command line.

If your Java application crashes or hangs, Java dumps can provide useful information to help you diagnose the root cause.

- If your application crashes, Java dumps are generated automatically for the following types of failure:
  - the VM receives an unexpected signal or an assertion failure
  - the VM runs out of memory
- If your application hangs, you can trigger the generation of a Java dump by sending a SIGQUIT signal (`kill -3`) to the VM.

Note: On Windows®, if you started the VM in a console window you can force the VM to produce a Java dump in response to a SIGBREAK signal (Ctrl-Brea
trigger a full system dump by finding the VM process in the **Processes** tab of the Windows Task Manager and clicking **Create dump file**.

To help you understand how a Java dump can help you with problem diagnosis, this topic includes a few scenarios to help you interpret the data:

- A crash caused by a general protection fault (gpf)
- A Java heap OutOfMemoryError (OOM)
- A native OutOfMemoryError (OOM)
- A deadlock situation
- A hang

Java dump contents

Java dumps summarize the state of the VM when the event occurs, with most of the information relating to components of the VM. The file is made up of a nu

TITLE

The first section of the Java dump file provides information about the event that triggered the production of the dump. In the following example, you can see

```
0SECTION       TITLE subcomponent dump routine
NULL           ===============================
1TICHARSET     UTF-8
1TISIGINFO     Dump Event "vmstop" (00000002) Detail "#0000000000000000" received
1TIDATETIMEUTC Date: 2021/04/23 at 18:02:44:017 (UTC)
1TIDATETIME    Date: 2021/04/23 at 14:02:44:017
1TITIMEZONE    Timezone: UTC-4 (EDT)
1TINANOTIME    System nanotime: 379202644260787
1TIFILENAME    Javacore filename:    /home/doc-javacore/javacore.20210423.140244.1175.0001.txt
1TIREQFLAGS    Request Flags: 0x81 (exclusive+preempt)
1TIPREPSTATE   Prep State: 0x106 (vm_access+exclusive_vm_access+trace_disabled)
```

GPINFO

The GPINFO section provides general information about the system that the VM is running on. The following example is taken from a Java dump that was ger

```
NULL           ------------------------------------------------------------------------
0SECTION       GPINFO subcomponent dump routine
NULL           ================================
2XHOSLEVEL     OS Level         : Linux 3.10.0-862.11.6.el7.x86_64
2XHCPUS        Processors -
3XHCPUARCH       Architecture   : amd64
3XHNUMCPUS       How Many       : 4
3XHNUMASUP       NUMA is either not supported or has been disabled by user
NULL
```

```
1XHERROR2       Register dump section only produced for SIGSEGV, SIGILL or SIGFPE.
NULL
```

The content of this section can vary, depending on the cause of the dump. For example, if the dump was caused by a general protection fault (gpf), the library been involved. Look for the following line in the output:

```
1XHFLAGS        VM flags:0000000000000000
```

The hexadecimal number that is recorded for `VM flags` ends in MSSSS, where M is the VM component and SSSS is component-specific code as shown in th

Component
INTERPRETER
GC
GROW_STACK
JNI
JIT_CODEGEN
BCVERIFY
RTVERIFY
SHAREDCLASSES

A value of `0000000000000000` (0x00000) indicates that a crash occurred outside of the VM.

ENVINFO

This section contains useful information about the environment in which the crash took place, including the following data:

- Java version (`1CIJAVAVERSION`)
- Eclipse OpenJ9™ VM and subcomponent version information (`1CIVMVERSION`, `1CIJ9VMVERSION`, `1CIJITVERSION`, `1CIOMRVERSION`, `1CIJCLVERSION`
- VM start time (`1CISTARTTIME`) and process information (`1CIPROCESSID`)
- Java home (`1CIJAVAHOMEDIR`) and DLL (`1CIJAVADLLDIR`) directories
- User arguments passed on the command line (`1CIUSERARGS`), identifying those that are ignored (`1CIIGNOREDARGS`)
- User limits imposed by the system (`1CIUSERLIMITS`)
- Environment variables in place (`1CIENVVARS`)
- System information (`1CISYSINFO`)
- CPU information (`1CICPUINFO`)
- Control group (Cgroup) information (`1CICGRPINFO`)

For clarity, the following example shows a shortened version of this section, where `...` indicates that lines are removed:

```
NULL           ------------------------------------------------------------------------
0SECTION       ENVINFO subcomponent dump routine
NULL           ================================
1CIJAVAVERSION JRE 9 Linux amd64-64 (build 9.0.4-internal+0-adhoc..openj9-openjdk-jdk9)
1CIVMVERSION   20180830_000000
1CIJ9VMVERSION 8e7c6ec
1CIJITVERSION  8e7c6ec
1CIOMRVERSION  553811b_CMPRSS
1CIJCLVERSION  ec1d223 based on jdk-9.0.4+12
1CIJITMODES    JIT enabled, AOT enabled, FSD disabled, HCR enabled
1CIRUNNINGAS   Running as a standalone JVM
1CIVMIDLESTATE VM Idle State: ACTIVE
```

```
1CICONTINFO    Running in container : FALSE
1CICGRPINFO    JVM support for cgroups enabled : TRUE
1CISTARTTIME   JVM start time: 2018/08/30 at 21:55:47:387
1CISTARTNANO   JVM start nanotime: 22012135233549
1CIPROCESSID   Process ID: 30285 (0x764D)
1CICMDLINE     [not available]
1CIJAVAHOMEDIR Java Home Dir:   /home/me/openj9-openjdk-jdk9/build/linux-x86_64-normal-server-release/images/jdk
1CIJAVADLLDIR  Java DLL Dir:    /home/me/openj9-openjdk-jdk9/build/linux-x86_64-normal-server-release/images/jdk/bin
1CISYSCP       Sys Classpath:
1CIUSERARGS    UserArgs:
2CIUSERARG              -Xoptionsfile=/home/me/openj9-openjdk-jdk9/build/linux-x86_64-normal-server-release/images/jdk/lib/options.default
...
1CIIGNOREDARGS Ignored Args:
2CIIGNOREDARG           -XX:+UseCompressedOop
2CIIGNOREDARG           -XX:CompressedClassSpaceSize=528482304
NULL
1CIUSERLIMITS  User Limits (in bytes except for NOFILE and NPROC)
NULL           ------------------------------------------------------------------------
NULL           type                            soft limit           hard limit
2CIUSERLIMIT   RLIMIT_AS                        unlimited            unlimited
2CIUSERLIMIT   RLIMIT_CORE                              0            unlimited
2CIUSERLIMIT   RLIMIT_CPU                       unlimited            unlimited
2CIUSERLIMIT   RLIMIT_DATA                      unlimited            unlimited
2CIUSERLIMIT   RLIMIT_FSIZE                     unlimited            unlimited
2CIUSERLIMIT   RLIMIT_LOCKS                     unlimited            unlimited
2CIUSERLIMIT   RLIMIT_MEMLOCK                       65536                65536
2CIUSERLIMIT   RLIMIT_NOFILE                         4096                 4096
2CIUSERLIMIT   RLIMIT_NPROC                          4096                30592
2CIUSERLIMIT   RLIMIT_RSS                       unlimited            unlimited
2CIUSERLIMIT   RLIMIT_STACK                       8388608            unlimited
2CIUSERLIMIT   RLIMIT_MSGQUEUE                    819200               819200
2CIUSERLIMIT   RLIMIT_NICE                              0                    0
2CIUSERLIMIT   RLIMIT_RTPRIO                            0                    0
2CIUSERLIMIT   RLIMIT_SIGPENDING                   30592                30592
NULL
1CIENVVARS     Environment Variables
NULL           ------------------------------------------------------------------------
2CIENVVAR      XDG_VTNR=1
2CIENVVAR      SSH_AGENT_PID=2653
...
NULL
1CISYSINFO     System Information
NULL           ------------------------------------------------------------------------
2CISYSINFO     /proc/sys/kernel/core_pattern = core
2CISYSINFO     /proc/sys/kernel/core_uses_pid = 1
NULL
1CICPUINFO     CPU Information
NULL           ------------------------------------------------------------------------
2CIPHYSCPU     Physical CPUs: 8
2CIONLNCPU     Online CPUs: 8
2CIBOUNDCPU    Bound CPUs: 8
2CIACTIVECPU   Active CPUs: 0
2CITARGETCPU   Target CPUs: 8
2CIJITFEATURE  CPU features (JIT): fpu cx8 cmov mmx sse sse2 ssse3 fma sse4_1 popcnt aesni osxsave avx avx2 rdt_m
2CIAOTFEATURE  CPU features (AOT): fpu cx8 cmov mmx sse sse2 ssse3 fma sse4_1 popcnt aesni osxsave avx avx2 rdt_m
NULL
```

```
1CICGRPINFO    Cgroup Information
NULL           ------------------------------------------------------------------------
2CICGRPINFO    subsystem : cpu
2CICGRPINFO    cgroup name : /
3CICGRPINFO    CPU Period : 100000 microseconds
3CICGRPINFO    CPU Quota : Not Set
3CICGRPINFO    CPU Shares : 1024
3CICGRPINFO    Period intervals elapsed count : 0
3CICGRPINFO    Throttled count : 0
3CICGRPINFO    Total throttle time : 0 nanoseconds
2CICGRPINFO    subsystem : cpuset
2CICGRPINFO    cgroup name : /
3CICGRPINFO    CPU exclusive : 1
3CICGRPINFO    Mem exclusive : 1
3CICGRPINFO    CPUs : 0-7
3CICGRPINFO    Mems : 0
2CICGRPINFO    subsystem : memory
2CICGRPINFO    cgroup name : /
3CICGRPINFO    Memory Limit : Not Set
3CICGRPINFO    Memory + Swap Limit : Not Set
3CICGRPINFO    Memory Usage : 5363396608 bytes
3CICGRPINFO    Memory + Swap Usage : 5363396608 bytes
3CICGRPINFO    Memory Max Usage : 0 bytes
3CICGRPINFO    Memory + Swap Max Usage : 0 bytes
3CICGRPINFO    Memory limit exceeded count : 0
3CICGRPINFO    Memory + Swap limit exceeded count : 0
3CICGRPINFO    OOM Killer Disabled : 0
3CICGRPINFO    Under OOM : 0
NULL
```

NATIVEMEMINFO

This section records information about native memory that is requested by using library functions such as `malloc()` and `mmap()`. Values are provided as a l
memory are allocated (but not yet freed) to VM Classes, which correspond to 141 allocations.

```
NULL           ------------------------------------------------------------------------
0SECTION       NATIVEMEMINFO subcomponent dump routine
NULL           ================================
0MEMUSER
1MEMUSER       JRE: 2,569,088,312 bytes / 4653 allocations
1MEMUSER       |
2MEMUSER       +--VM: 2,280,088,336 bytes / 2423 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Classes: 4,682,840 bytes / 141 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Memory Manager (GC): 2,054,966,784 bytes / 433 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Java Heap: 2,014,113,792 bytes / 1 allocation
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Other: 40,852,992 bytes / 432 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Threads: 10,970,016 bytes / 156 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Java Stack: 197,760 bytes / 16 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Native Stack: 10,616,832 bytes / 17 allocations
```

```
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Other: 155,424 bytes / 123 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Trace: 180,056 bytes / 263 allocations
2MEMUSER       |  |
3MEMUSER       |  +--JVMTI: 17,776 bytes / 13 allocations
2MEMUSER       |  |
3MEMUSER       |  +--JNI: 36,184 bytes / 52 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Port Library: 208,179,632 bytes / 72 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Unused <32bit allocation regions: 208,168,752 bytes / 1 allocation
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Other: 10,880 bytes / 71 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Other: 1,055,048 bytes / 1293 allocations
1MEMUSER       |
2MEMUSER       +--JIT: 288,472,816 bytes / 140 allocations
2MEMUSER       |  |
3MEMUSER       |  +--JIT Code Cache: 268,435,456 bytes / 1 allocation
2MEMUSER       |  |
3MEMUSER       |  +--JIT Data Cache: 2,097,216 bytes / 1 allocation
2MEMUSER       |  |
3MEMUSER       |  +--Other: 17,940,144 bytes / 138 allocations
1MEMUSER       |
2MEMUSER       +--Class Libraries: 13,432 bytes / 25 allocations
2MEMUSER       |  |
3MEMUSER       |  +--VM Class Libraries: 13,432 bytes / 25 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--sun.misc.Unsafe: 3,184 bytes / 13 allocations
4MEMUSER       |  |  |  |
5MEMUSER       |  |  |  +--Direct Byte Buffers: 1,056 bytes / 12 allocations
4MEMUSER       |  |  |  |
5MEMUSER       |  |  |  +--Other: 2,128 bytes / 1 allocation
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Other: 10,248 bytes / 12 allocations
1MEMUSER       |
2MEMUSER       +--Unknown: 513,728 bytes / 2065 allocations
NULL
```

This section does not record memory that is allocated by application or JNI code and is typically a little less than the value recorded by operating system too

MEMINFO

This section relates to memory management, providing a breakdown of memory usage in the VM for the object heap, internal memory, memory used for clas

The object memory area (1STHEAPTYPE) records each memory region in use, its start and end address, and region size. Further information is recorded abou the segment control data structure, the start and end address of the native memory segment, as well as the segment size.

For clarity, the following example shows a shortened version of this section, where `...` indicates that lines are removed:

```
NULL           ------------------------------------------------------------------------
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
NULL
1STHEAPTYPE    Object Memory
NULL           id                start             end               size              space/region
1STHEAPSPACE   0x00007FF4F00744A0         --                --                --        Generational
```

```
1STHEAPREGION  0x00007FF4F0074CE0 0x0000000087F40000 0x0000000088540000 0x0000000000600000 Generational/Tenured Region
1STHEAPREGION  0x00007FF4F0074930 0x00000000FFE00000 0x00000000FFF00000 0x0000000000100000 Generational/Nursery Region
1STHEAPREGION  0x00007FF4F0074580 0x00000000FFF00000 0x0000000100000000 0x0000000000100000 Generational/Nursery Region
NULL
1STHEAPTOTAL   Total memory:                    8388608 (0x0000000000800000)
1STHEAPINUSE   Total memory in use:             2030408 (0x00000000001EFB48)
1STHEAPFREE    Total memory free:               6358200 (0x00000000006104B8)
NULL
1STSEGTYPE     Internal Memory
NULL           segment           start             alloc             end               type       size
1STSEGMENT     0x00007FF4F004CBC8 0x00007FF4CD33C000 0x00007FF4CD33C000 0x00007FF4CE33C000 0x01000440 0x0000000001000000
1STSEGMENT     0x00007FF4F004CB08 0x00007FF4DE43D030 0x00007FF4DE517770 0x00007FF4DE53D030 0x00800040 0x0000000000100000
NULL
1STSEGTOTAL    Total memory:                  17825792 (0x0000000001100000)
1STSEGINUSE    Total memory in use:             894784 (0x00000000000DA740)
1STSEGFREE     Total memory free:             16931008 (0x00000000010258C0)
NULL
1STSEGTYPE     Class Memory
NULL           segment           start             alloc             end               type       size
1STSEGMENT     0x00007FF4F03B5638 0x0000000001053D98 0x000000000105BD98 0x000000000105BD98 0x00010040 0x0000000000008000
1STSEGMENT     0x00007FF4F03B5578 0x0000000001048188 0x0000000001050188 0x0000000001050188 0x00010040 0x0000000000008000
...
NULL
1STSEGTOTAL    Total memory:                    3512520 (0x00000000003598C8)
1STSEGINUSE    Total memory in use:             3433944 (0x00000000003465D8)
1STSEGFREE     Total memory free:                 78576 (0x00000000000132F0)
NULL
1STSEGTYPE     JIT Code Cache
NULL           segment           start             alloc             end               type       size
1STSEGMENT     0x00007FF4F00961F8 0x00007FF4CE43D000 0x00007FF4CE445790 0x00007FF4DE43D000 0x00000068 0x0000000010000000
NULL
1STSEGTOTAL    Total memory:                  268435456 (0x0000000010000000)
1STSEGINUSE    Total memory in use:              34704 (0x0000000000008790)
1STSEGFREE     Total memory free:             268400752 (0x000000000FFF7870)
1STSEGLIMIT    Allocation limit:              268435456 (0x0000000010000000)
NULL
1STSEGTYPE     JIT Data Cache
NULL           segment           start             alloc             end               type       size
1STSEGMENT     0x00007FF4F0096668 0x00007FF4CC553030 0x00007FF4CC753030 0x00007FF4CC753030 0x00000048 0x0000000000200000
NULL
1STSEGTOTAL    Total memory:                    2097152 (0x0000000000200000)
1STSEGINUSE    Total memory in use:             2097152 (0x0000000000200000)
1STSEGFREE     Total memory free:                     0 (0x0000000000000000)
1STSEGLIMIT    Allocation limit:              402653184 (0x0000000018000000)
NULL
1STGCHTYPE     GC History
NULL
```

In the example, the GC History (1STGCHTYPE) section is blank. This section is populated if a garbage collection cycle occurred in a VM that is being diagnose

LOCKS

This section of the Java dump provides information about locks, which protect shared resources from being accessed by more than one entity at a time. The the threads that are causing the problem, which enables you to identify the root cause.

The following example shows a typical LOCKS section, where no deadlocks existed at the time the dump was triggered. For clarity, the following example sho

```
NULL           ------------------------------------------------------------------------
0SECTION       LOCKS subcomponent dump routine
NULL           ===============================
NULL
1LKPOOLINFO    Monitor pool info:
2LKPOOLTOTAL     Current total number of monitors: 3
NULL
1LKMONPOOLDUMP Monitor Pool Dump (flat & inflated object-monitors):
2LKMONINUSE      sys_mon_t:0x00007FF4B0001D78 infl_mon_t: 0x00007FF4B0001DF8:
3LKMONOBJECT       java/lang/ref/ReferenceQueue@0x00000000FFE26A10: <unowned>
3LKNOTIFYQ            Waiting to be notified:
3LKWAITNOTIFY             "Common-Cleaner" (J9VMThread:0x0000000000FD0100)
NULL
1LKREGMONDUMP  JVM System Monitor Dump (registered monitors):
2LKREGMON          Thread global lock (0x00007FF4F0004FE8): <unowned>
2LKREGMON          &(PPG_mem_mem32_subAllocHeapMem32.monitor) lock (0x00007FF4F0005098): <unowned>
2LKREGMON          NLS hash table lock (0x00007FF4F0005148): <unowned>
...
NULL
```

THREADS

The THREADS section of a Java dump file provides summary information about the VM thread pool and detailed information about Java threads, native threa

A Java thread runs on a native thread. Several lines are recorded for each Java thread in the `Thread Details` subsection, which include the following key p

– 3XMTHREADINFO: The thread name, address information for the VM thread structures and Java thread object, the thread state, and thread priority.
– 3XMJAVALTHREAD: The Java thread ID and daemon status from the thread object.
– 3XMTHREADINFO1: The native operating system thread ID, priority, scheduling policy, internal VM thread state, and VM thread flags.
– 3XMTHREADINFO2: The native stack address range.
– 3XMTHREADINFO3: Java call stack information (4XESTACKTRACE) or Native call stack information (4XENATIVESTACK).
– 5XESTACKTRACE: This line indicates whether locks were taken by a specific method.

Java thread priorities are mapped to operating system priority values. Thread states are shown in the following table:

Thread state value	Status	Descriptio
R	Runnable	The thread
CW	Condition Wait	The thread
S	Suspended	The thread
Z	Zombie	The thread
P	Parked	The thread
B	Blocked	The thread

For threads that are parked (P), blocked (B), or waiting (CW), an additional line (3XMTHREADBLOCK) is included in the output that shows what the thread is pa
includes the name of the thread that is currently working to progress the initialization of the class. You can use this information to diagnose deadlocks that ar

For clarity, the following example shows a shortened version of a typical THREADS section, where `...` indicates that lines are removed:

```
NULL           ------------------------------------------------------------------------
0SECTION       THREADS subcomponent dump routine
NULL           ================================
NULL
```

```
1XMPOOLINFO    JVM Thread pool info:
2XMPOOLTOTAL       Current total number of pooled threads: 19
2XMPOOLLIVE        Current total number of live threads: 18
2XMPOOLDAEMON      Current total number of live daemon threads: 15
NULL
1XMTHDINFO     Thread Details
NULL
...
3XMTHREADINFO      "JIT Diagnostic Compilation Thread-007 Suspended" J9VMThread:0x0000000000035200, omrthread_t:0x00007F3F8C0D02C8, java/lang/Thread:0x000
3XMJAVALTHREAD            (java/lang/Thread getId:0x9, isDaemon:true)
3XMJAVALTHRCCL            sun/misc/Launcher$AppClassLoader(0x00000000FFF3BF98)
3XMTHREADINFO1            (native thread ID:0x618F, native priority:0xB, native policy:UNKNOWN, vmstate:CW, vm thread flags:0x00000081)
3XMTHREADINFO2            (native stack address range from:0x00007F3F879C5000, to:0x00007F3F87AC5000, size:0x100000)
3XMCPUTIME               CPU usage total: 0.052410771 secs, current category="JIT"
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=0 (0x0)
3XMTHREADINFO3           No Java callstack associated with this thread
...
NULL
...
3XMTHREADINFO      "Class Initialization Thread 2" J9VMThread:0x0000000000124D00, omrthread_t:0x00007F3F8C1494C8, java/lang/Thread:0x00000000FFF53EE8, sta
3XMJAVALTHREAD            (java/lang/Thread getId:0x13, isDaemon:false)
3XMJAVALTHRCCL            sun/misc/Launcher$AppClassLoader(0x00000000FFF3BF98)
3XMTHREADINFO1            (native thread ID:0x6199, native priority:0x5, native policy:UNKNOWN, vmstate:CW, vm thread flags:0x00000181)
3XMTHREADINFO2            (native stack address range from:0x00007F3F74AB4000, to:0x00007F3F74AF4000, size:0x40000)
3XMCPUTIME               CPU usage total: 0.008712260 secs, current category="Application"
3XMTHREADBLOCK     Waiting on: java/lang/J9VMInternals$ClassInitializationLock@0x00000000FFF61C90 Owned by: <unowned> Initializing thread: "Class Initiali
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=4096 (0x1000)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at java/lang/Class.forNameImpl(Native Method)
4XESTACKTRACE                at java/lang/Class.forName(Class.java:339)
4XESTACKTRACE                at ClassInitLockBug$ClassInitThread.run(ClassInitLockBug.java:16)
...
NULL
...
NULL
3XMTHREADINFO      "Class Initialization Thread 1" J9VMThread:0x0000000000124100, omrthread_t:0x00007F3F8C148F50, java/lang/Thread:0x00000000FFF53D80, sta
3XMJAVALTHREAD            (java/lang/Thread getId:0x12, isDaemon:false)
3XMJAVALTHRCCL            sun/misc/Launcher$AppClassLoader(0x00000000FFF3BF98)
3XMTHREADINFO1            (native thread ID:0x6198, native priority:0x5, native policy:UNKNOWN, vmstate:CW, vm thread flags:0x00000481)
3XMTHREADINFO2            (native stack address range from:0x00007F3F74AF5000, to:0x00007F3F74B35000, size:0x40000)
3XMCPUTIME               CPU usage total: 0.010221701 secs, current category="Application"
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=12736 (0x31C0)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at java/lang/Thread.sleepImpl(Native Method)
4XESTACKTRACE                at java/lang/Thread.sleep(Thread.java:983)
4XESTACKTRACE                at java/lang/Thread.sleep(Thread.java:966)
4XESTACKTRACE                at TestClass.<clinit>(ClassInitLockBug.java:29)
4XESTACKTRACE                at java/lang/Class.forNameImpl(Native Method)
4XESTACKTRACE                at java/lang/Class.forName(Class.java:339)
4XESTACKTRACE                at ClassInitLockBug$ClassInitThread.run(ClassInitLockBug.java:16)
...
NULL
...
NULL
1XMTHDSUMMARY  Threads CPU Usage Summary
NULL           =========================
```

```
NULL
1XMTHDCATINFO  Warning: to get more accurate CPU times for the GC, the option -XX:-ReduceCPUMonitorOverhead can be used. See the user guide for more info
NULL
1XMTHDCATEGORY All JVM attached threads: 0.698865000 secs
1XMTHDCATEGORY |
2XMTHDCATEGORY +--System-JVM: 0.653723000 secs
2XMTHDCATEGORY |  |
3XMTHDCATEGORY |  +--GC: 0.047248000 secs
2XMTHDCATEGORY |  |
3XMTHDCATEGORY |  +--JIT: 0.512971000 secs
1XMTHDCATEGORY |
2XMTHDCATEGORY +--Application: 0.045142000 secs
```

HOOKS

This section shows internal VM event callbacks, which are used for diagnosing performance problems in the VM. Multiple hook interfaces are listed, which in

The following example shows data for the `J9VMHookInterface`, including the total time for all previous events, the call site location (<source file>:<line nu

```
NULL           ------------------------------------------------------------------------
SECTION HOOK   subcomponent dump routine
NULL           ========================
1NOTE          These data are reset every time a javacore is taken
1HKINTERFACE   MM_OMRHookInterface
NULL           ------------------------------------------------------------------------
1HKINTERFACE   MM_PrivateHookInterface
NULL           ------------------------------------------------------------------------
1HKINTERFACE   MM_HookInterface
NULL           ------------------------------------------------------------------------
1HKINTERFACE   J9VMHookInterface
NULL           ------------------------------------------------------------------------
2HKEVENTID     1
3HKCALLCOUNT       1239
3HKTOTALTIME       219564us
3HKLAST            Last Callback
4HKCALLSITE            trcengine.c:395
4HKSTARTTIME           Start Time: 2019-10-18T00:15:14.664
4HKDURATION            Duration : 16us
3HKLONGST          Longest Callback
4HKCALLSITE            trcengine.c:395
4HKSTARTTIME           Start Time: 2019-10-18T21:28:34.895
4HKDURATION            Duration : 5012us
NULL
...
1HKINTERFACE   J9VMZipCachePoolHookInterface
NULL           ------------------------------------------------------------------------
1HKINTERFACE   J9JITHookInterface
NULL           ------------------------------------------------------------------------
2HKEVENTID     3
3HKCALLCOUNT       3113
3HKTOTALTIME       4904us
3HKLAST            Last Callback
4HKCALLSITE            common/mgmtinit.c:193
4HKSTARTTIME           Start Time: 2019-10-18T16:04:15.320
4HKDURATION            Duration : 3us
3HKLONGST          Longest Callback
4HKCALLSITE            common/mgmtinit.c:193
```

```
4HKSTARTTIME          Start Time: 2019-10-18T16:37:17.633
4HKDURATION           Duration : 27us
NULL
...
```

SHARED CLASSES

If the shared classes cache is enabled at run time, the information that is provided in a Java dump file describes settings that were used when creating the ca

In the following example, the shared classes cache was created with a Class Debug Area (`-Xnolinenumbers=false`). Byte code instrumentation (BCI) is e

The `Cache Summary` shows a cache size (2SCLTEXTCSZ) of 16776608 bytes, with a soft maximum size (2SCLTEXTSMB) also of 16776608 bytes, which lea

In the `Cache Memory Status` subsection, the line 2SCLTEXTCMDT indicates the name and location of the shared cache and `cr` indicates that the cache is

```
NULL          ------------------------------------------------------------------------
0SECTION      SHARED CLASSES subcomponent dump routine
NULL          ========================================
NULL
1SCLTEXTCRTW  Cache Created With
NULL          -----------------
NULL
2SCLTEXTXNL       -Xnolinenumbers       = false
2SCLTEXTBCI       BCI Enabled           = true
2SCLTEXTBCI       Restrict Classpaths   = false
NULL
1SCLTEXTCSUM  Cache Summary
NULL          -----------------
NULL
2SCLTEXTNLC       No line number content                  = false
2SCLTEXTLNC       Line number content                     = true
NULL
2SCLTEXTRCS       ROMClass start address                  = 0x00007F423061C000
2SCLTEXTRCE       ROMClass end address                    = 0x00007F42307B9A28
2SCLTEXTMSA       Metadata start address                  = 0x00007F42313D42FC
2SCLTEXTCEA       Cache end address                       = 0x00007F4231600000
2SCLTEXTRTF       Runtime flags                           = 0x00102001ECA6028B
2SCLTEXTCGN       Cache generation                        = 35
NULL
2SCLTEXTCSZ       Cache size                              = 16776608
2SCLTEXTSMB       Softmx bytes                            = 16776608
2SCLTEXTFRB       Free bytes                              = 12691668
2SCLTEXTRCB       ROMClass bytes                          = 1694248
2SCLTEXTAOB       AOT code bytes                          = 0
2SCLTEXTADB       AOT data bytes                          = 0
2SCLTEXTAHB       AOT class hierarchy bytes               = 32
2SCLTEXTATB       AOT thunk bytes                         = 0
2SCLTEXTARB       Reserved space for AOT bytes            = -1
2SCLTEXTAMB       Maximum space for AOT bytes             = -1
2SCLTEXTJHB       JIT hint bytes                          = 308
2SCLTEXTJPB       JIT profile bytes                       = 2296
2SCLTEXTJRB       Reserved space for JIT data bytes       = -1
2SCLTEXTJMB       Maximum space for JIT data bytes        = -1
2SCLTEXTNOB       Java Object bytes                       = 0
2SCLTEXTZCB       Zip cache bytes                         = 919328
2SCLTEXTSHB       Startup hint bytes                      = 0
2SCLTEXTRWB       ReadWrite bytes                         = 114080
2SCLTEXTJCB       JCL data bytes                          = 0
```

```
2SCLTEXTBDA        Byte data bytes                          = 0
2SCLTEXTMDA        Metadata bytes                           = 23448
2SCLTEXTDAS        Class debug area size                    = 1331200
2SCLTEXTDAU        Class debug area % used                  = 11%
2SCLTEXTDAN        Class LineNumberTable bytes              = 156240
2SCLTEXTDAV        Class LocalVariableTable bytes           = 0
NULL
2SCLTEXTNRC        Number ROMClasses                        = 595
2SCLTEXTNAM        Number AOT Methods                       = 0
2SCLTEXTNAD        Number AOT Data Entries                  = 0
2SCLTEXTNAH        Number AOT Class Hierarchy               = 1
2SCLTEXTNAT        Number AOT Thunks                        = 0
2SCLTEXTNJH        Number JIT Hints                         = 14
2SCLTEXTNJP        Number JIT Profiles                      = 20
2SCLTEXTNCP        Number Classpaths                        = 1
2SCLTEXTNUR        Number URLs                              = 0
2SCLTEXTNTK        Number Tokens                            = 0
2SCLTEXTNOJ        Number Java Objects                      = 0
2SCLTEXTNZC        Number Zip Caches                        = 5
2SCLTEXTNSH        Number Startup Hint Entries              = 0
2SCLTEXTNJC        Number JCL Entries                       = 0
2SCLTEXTNST        Number Stale classes                     = 0
2SCLTEXTPST        Percent Stale classes                    = 0%
NULL
2SCLTEXTCPF        Cache is 24% full
NULL
1SCLTEXTCMST   Cache Memory Status
NULL           ------------------
1SCLTEXTCNTD       Cache Name                    Feature                 Memory type             Cache path
NULL
2SCLTEXTCMDT       sharedcc_doc-javacore         CR                      Memory mapped file      /tmp/javasharedresources/C290M4F1A64P_sharedcc_doc-java
NULL
1SCLTEXTCMST   Cache Lock Status
NULL           ------------------
1SCLTEXTCNTD       Lock Name                     Lock type               TID owning lock
NULL
2SCLTEXTCWRL       Cache write lock              File lock               Unowned
2SCLTEXTCRWL       Cache read/write lock         File lock               Unowned
NULL
```

The following example shows information for a layered cache:

```
NULL           ------------------------------------------------------------------------
0SECTION       SHARED CLASSES subcomponent dump routine
NULL           ========================================
NULL
1SCLTEXTCSTL   Cache Statistics for Top Layer
NULL
1SCLTEXTCRTW       Cache Created With
NULL               ------------------
NULL
2SCLTEXTXNL            -Xnolinenumbers       = false
2SCLTEXTBCI            BCI Enabled           = true
2SCLTEXTBCI            Restrict Classpaths   = false
NULL
1SCLTEXTCSUM       Cache Summary
```

```
NULL              ------------------
NULL
2SCLTEXTNLC           No line number content                  = false
2SCLTEXTLNC           Line number content                     = false
NULL
2SCLTEXTRCS           ROMClass start address                  = 0x00007F0EDB567000
2SCLTEXTRCE           ROMClass end address                    = 0x00007F0EDB567000
2SCLTEXTMSA           Metadata start address                  = 0x00007F0EDC40241C
2SCLTEXTCEA           Cache end address                       = 0x00007F0EDC54B000
2SCLTEXTRTF           Runtime flags                           = 0x80102001ECA602BB
2SCLTEXTCGN           Cache generation                        = 41
2SCLTEXTCLY           Cache layer                             = 1
NULL
2SCLTEXTCSZ           Cache size                              = 16776608
2SCLTEXTSMB           Softmx bytes                            = 16776608
2SCLTEXTFRB           Free bytes                              = 15315996
2SCLTEXTARB           Reserved space for AOT bytes            = -1
2SCLTEXTAMB           Maximum space for AOT bytes             = -1
2SCLTEXTJRB           Reserved space for JIT data bytes       = -1
2SCLTEXTJMB           Maximum space for JIT data bytes        = -1
2SCLTEXTRWB           ReadWrite bytes                         = 114080
2SCLTEXTDAS           Class debug area size                   = 1331200
2SCLTEXTDAU           Class debug area % used                 = 0%
2SCLTEXTDAN           Class LineNumberTable bytes             = 0
2SCLTEXTDAV           Class LocalVariableTable bytes          = 0
NULL
2SCLTEXTCPF           Cache is 8% full
NULL
1SCLTEXTCMST      Cache Memory Status
NULL              ------------------
1SCLTEXTCNTD          Cache Name                  Feature                 Memory type             Cache path
NULL
2SCLTEXTCMDT          Cache1                      CR                      Memory mapped file      /tmp/javasharedresources/C290M4F1A64P_Cache1_G41L01
NULL
1SCLTEXTCMST      Cache Lock Status
NULL              ------------------
1SCLTEXTCNTD          Lock Name                   Lock type               TID owning lock
NULL
2SCLTEXTCWRL          Cache write lock            File lock               Unowned
2SCLTEXTCRWL          Cache read/write lock       File lock               Unowned
NULL
1SCLTEXTCSAL  Cache Statistics for All Layers
NULL
2SCLTEXTRCB           ROMClass bytes                          = 1459040
2SCLTEXTAOB           AOT code bytes                          = 57624
2SCLTEXTADB           AOT data bytes                          = 272
2SCLTEXTAHB           AOT class hierarchy bytes               = 1840
2SCLTEXTATB           AOT thunk bytes                         = 632
2SCLTEXTJHB           JIT hint bytes                          = 484
2SCLTEXTJPB           JIT profile bytes                       = 0
2SCLTEXTNOB           Java Object bytes                       = 0
2SCLTEXTZCB           Zip cache bytes                         = 1134016
2SCLTEXTSHB           Startup hint bytes                      = 0
2SCLTEXTJCB           JCL data bytes                          = 0
2SCLTEXTBDA           Byte data bytes                         = 0
NULL
```

```
2SCLTEXTNRC            Number ROMClasses                       = 503
2SCLTEXTNAM            Number AOT Methods                      = 16
2SCLTEXTNAD            Number AOT Data Entries                 = 1
2SCLTEXTNAH            Number AOT Class Hierarchy              = 28
2SCLTEXTNAT            Number AOT Thunks                       = 11
2SCLTEXTNJH            Number JIT Hints                        = 15
2SCLTEXTNJP            Number JIT Profiles                     = 0
2SCLTEXTNCP            Number Classpaths                       = 1
2SCLTEXTNUR            Number URLs                             = 0
2SCLTEXTNTK            Number Tokens                           = 0
2SCLTEXTNOJ            Number Java Objects                     = 0
2SCLTEXTNZC            Number Zip Caches                       = 21
2SCLTEXTNSH            Number Startup Hint Entries             = 0
2SCLTEXTNJC            Number JCL Entries                      = 0
2SCLTEXTNST            Number Stale classes                    = 0
2SCLTEXTPST            Percent Stale classes                   = 0%
```

CLASSES

The classes section shows information about class loaders. The first part is a summary that records each available class loader (2CLTEXTCLLOADER) followe loaded.

In the example you can see that the `java/lang/InternalAnonymousClassLoader` loaded two classes, `jdk/internal/loader/BuiltinClassLoac jdk/internal/loader/BuiltinClassLoader$$Lambda$1/00000000F00D2460(0x0000000001018A00)`.

```
NULL           ------------------------------------------------------------------------
0SECTION       CLASSES subcomponent dump routine
NULL           ================================
1CLTEXTCLLOS    Classloader summaries
1CLTEXTCLLSS        12345678: 1=primordial,2=extension,3=shareable,4=middleware,5=system,6=trusted,7=application,8=delegating
2CLTEXTCLLOADER     p---st-- Loader *System*(0x00000000FFE1D258)
3CLNMBRLOADEDLIB        Number of loaded libraries 5
3CLNMBRLOADEDCL         Number of loaded classes 638
2CLTEXTCLLOADER     -x--st-- Loader jdk/internal/loader/ClassLoaders$PlatformClassLoader(0x00000000FFE1D4F0), Parent *none*(0x0000000000000000)
3CLNMBRLOADEDLIB        Number of loaded libraries 0
3CLNMBRLOADEDCL         Number of loaded classes 0
2CLTEXTCLLOADER     ----st-- Loader java/lang/InternalAnonymousClassLoader(0x00000000FFE1DFD0), Parent *none*(0x0000000000000000)
3CLNMBRLOADEDLIB        Number of loaded libraries 0
3CLNMBRLOADEDCL         Number of loaded classes 2
2CLTEXTCLLOADER     -----ta- Loader jdk/internal/loader/ClassLoaders$AppClassLoader(0x00000000FFE1DAD0), Parent jdk/internal/loader/ClassLoaders$Platform
3CLNMBRLOADEDLIB        Number of loaded libraries 0
3CLNMBRLOADEDCL         Number of loaded classes 0
1CLTEXTCLLIB    ClassLoader loaded libraries
2CLTEXTCLLIB        Loader *System*(0x00000000FFE1D258)
3CLTEXTLIB              /home/me/openj9-openjdk-jdk9/build/linux-x86_64-normal-server-release/images/jdk/lib/compressedrefs/jclse9_29
3CLTEXTLIB              /home/me/openj9-openjdk-jdk9/build/linux-x86_64-normal-server-release/images/jdk/lib/java
3CLTEXTLIB              /home/me/openj9-openjdk-jdk9/build/linux-x86_64-normal-server-release/images/jdk/lib/compressedrefs/j9jit29
3CLTEXTLIB              /home/me/openj9-openjdk-jdk9/build/linux-x86_64-normal-server-release/images/jdk/lib/zip
3CLTEXTLIB              /home/me/openj9-openjdk-jdk9/build/linux-x86_64-normal-server-release/images/jdk/lib/nio
1CLTEXTCLLOD    ClassLoader loaded classes
2CLTEXTCLLOAD       Loader *System*(0x00000000FFE1D258)
3CLTEXTCLASS            [Ljava/lang/Thread$State;(0x0000000001056400)
...
2CLTEXTCLLOAD       Loader jdk/internal/loader/ClassLoaders$PlatformClassLoader(0x00000000FFE1D4F0)
2CLTEXTCLLOAD       Loader java/lang/InternalAnonymousClassLoader(0x00000000FFE1DFD0)
3CLTEXTCLASS            jdk/internal/loader/BuiltinClassLoader$$Lambda$2/00000000F03876A0(0x0000000001030F00)
3CLTEXTCLASS            jdk/internal/loader/BuiltinClassLoader$$Lambda$1/00000000F00D2460(0x0000000001018A00)
```

```
2CLTEXTCLLOAD       Loader jdk/internal/loader/ClassLoaders$AppClassLoader(0x00000000FFE1DAD0)
```

Scenarios

General Protection Fault

In this scenario, a Java application has crashed due to a General Protection Fault (GPF), automatically generating a Java dump file.

The first section of the file (TITLE) tells you that the GPF triggered the Java dump.

```
0SECTION       TITLE subcomponent dump routine
NULL           ===============================
1TICHARSET     UTF-8
1TISIGINFO     Dump Event "gpf" (00002000) received
1TIDATETIMEUTC Date: 2021/04/23 at 18:02:44:017 (UTC)
1TIDATETIME    Date: 2021/04/23 at 14:02:44:017
1TITIMEZONE    Timezone: UTC-4 (EDT)
1TINANOTIME    System nanotime: 379202644260787
1TIFILENAME    Javacore filename:    /home/test/JNICrasher/javacore.20210423.140244.29399.0002.txt
1TIREQFLAGS    Request Flags: 0x81 (exclusive+preempt)
1TIPREPSTATE   Prep State: 0x100 (trace_disabled)
1TIPREPINFO    Exclusive VM access not taken: data may not be consistent across javacore sections
```

To troubleshoot this problem, you need to know which thread caused the GPF to occur. The thread that was running at the time of the crash is reported as th

```
NULL           ------------------------------------------------------------------------
0SECTION       THREADS subcomponent dump routine
NULL           ================================
NULL
1XMPOOLINFO    JVM Thread pool info:
2XMPOOLTOTAL       Current total number of pooled threads: 16
2XMPOOLLIVE        Current total number of live threads: 15
2XMPOOLDAEMON      Current total number of live daemon threads: 14
NULL
1XMCURTHDINFO  Current thread
3XMTHREADINFO      "main" J9VMThread:0xB6B60E00, omrthread_t:0xB6B049D8, java/lang/Thread:0xB55444D0, state:R, prio=5
3XMJAVALTHREAD            (java/lang/Thread getId:0x1, isDaemon:false)
3XMTHREADINFO1            (native thread ID:0x72D8, native priority:0x5, native policy:UNKNOWN, vmstate:R, vm thread flags:0x00000000)
3XMTHREADINFO2            (native stack address range from:0xB6CE3000, to:0xB74E4000, size:0x801000)
3XMCPUTIME               CPU usage total: 0.319865924 secs, current category="Application"
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=778008 (0xBDF18)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at JNICrasher.doSomethingThatCrashes(Native Method)
4XESTACKTRACE                at JNICrasher.main(JNICrasher.java:7)
3XMTHREADINFO3           Native callstack:
4XENATIVESTACK               (0xB6C6F663 [libj9prt29.so+0x3b663])
4XENATIVESTACK               (0xB6C52F6E [libj9prt29.so+0x1ef6e])
4XENATIVESTACK               (0xB6C6F1CE [libj9prt29.so+0x3b1ce])
4XENATIVESTACK               (0xB6C6F2C6 [libj9prt29.so+0x3b2c6])
4XENATIVESTACK               (0xB6C6ED93 [libj9prt29.so+0x3ad93])
4XENATIVESTACK               (0xB6C52F6E [libj9prt29.so+0x1ef6e])
4XENATIVESTACK               (0xB6C6ED07 [libj9prt29.so+0x3ad07])
4XENATIVESTACK               (0xB6C6AA3D [libj9prt29.so+0x36a3d])
4XENATIVESTACK               (0xB6C6C3A4 [libj9prt29.so+0x383a4])
4XENATIVESTACK               (0xB667FA19 [libj9dmp29.so+0xfa19])
4XENATIVESTACK               (0xB6C52F6E [libj9prt29.so+0x1ef6e])
4XENATIVESTACK               (0xB66878CF [libj9dmp29.so+0x178cf])
```

```
4XENATIVESTACK               (0xB6688083 [libj9dmp29.so+0x18083])
4XENATIVESTACK               (0xB6C52F6E [libj9prt29.so+0x1ef6e])
4XENATIVESTACK               (0xB6680C0D [libj9dmp29.so+0x10c0d])
4XENATIVESTACK               (0xB667F9D7 [libj9dmp29.so+0xf9d7])
4XENATIVESTACK               (0xB6C52F6E [libj9prt29.so+0x1ef6e])
4XENATIVESTACK               (0xB668B02F [libj9dmp29.so+0x1b02f])
4XENATIVESTACK               (0xB668B4D3 [libj9dmp29.so+0x1b4d3])
4XENATIVESTACK               (0xB66740F1 [libj9dmp29.so+0x40f1])
4XENATIVESTACK               (0xB66726FA [libj9dmp29.so+0x26fa])
4XENATIVESTACK               (0xB6C52F6E [libj9prt29.so+0x1ef6e])
4XENATIVESTACK               (0xB66726A9 [libj9dmp29.so+0x26a9])
4XENATIVESTACK               (0xB6676AE4 [libj9dmp29.so+0x6ae4])
4XENATIVESTACK               (0xB668D75A [libj9dmp29.so+0x1d75a])
4XENATIVESTACK               (0xB6A28DD4 [libj9vm29.so+0x81dd4])
4XENATIVESTACK               (0xB6C52F6E [libj9prt29.so+0x1ef6e])
4XENATIVESTACK               (0xB6A289EE [libj9vm29.so+0x819ee])
4XENATIVESTACK               (0xB6A29A40 [libj9vm29.so+0x82a40])
4XENATIVESTACK               (0xB6C52B6A [libj9prt29.so+0x1eb6a])
4XENATIVESTACK               __kernel_rt_sigreturn+0x0 (0xB7747410)
4XENATIVESTACK               (0xB75330B6 [libffi29.so+0x50b6])
4XENATIVESTACK               ffi_raw_call+0xad (0xB7531C53 [libffi29.so+0x3c53])
4XENATIVESTACK               (0xB69BE4AB [libj9vm29.so+0x174ab])
4XENATIVESTACK               (0xB6A665BC [libj9vm29.so+0xbf5bc])
4XENATIVESTACK               (0xB6A15552 [libj9vm29.so+0x6e552])
4XENATIVESTACK               (0xB6A30894 [libj9vm29.so+0x89894])
4XENATIVESTACK               (0xB6A6F169 [libj9vm29.so+0xc8169])
4XENATIVESTACK               (0xB6C52F6E [libj9prt29.so+0x1ef6e])
4XENATIVESTACK               (0xB6A6F1FA [libj9vm29.so+0xc81fa])
4XENATIVESTACK               (0xB6A30994 [libj9vm29.so+0x89994])
4XENATIVESTACK               (0xB6A2CE4C [libj9vm29.so+0x85e4c])
4XENATIVESTACK               (0xB770487D [libjli.so+0x787d])
4XENATIVESTACK               (0xB7719F72 [libpthread.so.0+0x6f72])
4XENATIVESTACK               clone+0x5e (0xB763543E [libc.so.6+0xee43e])
```

The extract tells you that the current thread was `java/lang/Thread`, and information is provided about the Java call stack and native call stack (3XMTHRE/ causes a crash. The Java call stack shows the call to the JNI native method (`JNIcrasher`), and the native call stack shows the point of failure. In this examp usually produced alongside the Java dump. Open the system dump with the Dump viewer and use the `info thread` command to print the Java and native

The next time you run the application, you can use the -XX:+ShowNativeStackSymbols=all command line option to display the corresponding function name

```
4XENATIVESTACK               protectedBacktrace+0x12 (0x00007F3F9213E312 [libj9prt29.so+0x25312])
4XENATIVESTACK               omrsig_protect+0x1e3 (0x00007F3F92142AD3 [libj9prt29.so+0x29ad3])
4XENATIVESTACK               omrintrospect_backtrace_thread_raw+0xbf (0x00007F3F9213E80F [libj9prt29.so+0x2580f])
4XENATIVESTACK               omrsig_protect+0x1e3 (0x00007F3F92142AD3 [libj9prt29.so+0x29ad3])
4XENATIVESTACK               omrintrospect_backtrace_thread+0x70 (0x00007F3F9213E1D0 [libj9prt29.so+0x251d0])
4XENATIVESTACK               setup_native_thread+0x1d2 (0x00007F3F9213F652 [libj9prt29.so+0x26652])
4XENATIVESTACK               omrintrospect_threads_startDo_with_signal+0x474 (0x00007F3F921403F4 [libj9prt29.so+0x273f4])
4XENATIVESTACK               omrsig_protect+0x1e3 (0x00007F3F92142AD3 [libj9prt29.so+0x29ad3])
```

Java OutOfMemoryError

In this scenario, the Java heap runs out of memory, causing an `OutOfMemoryError`, which automatically generates a Java dump file.

The first section of the file (TITLE) tells you that a `systhrow` event triggered the Java dump as a result of an OOM (`java/lang/OutOfMemoryError`) for Ja

```
0SECTION       TITLE subcomponent dump routine
NULL           ===============================
1TICHARSET     UTF-8
```

```
1TISIGINFO     Dump Event "systhrow" (00040000) Detail "java/lang/OutOfMemoryError" "Java heap space" received
1TIDATETIMEUTC Date: 2021/04/23 at 18:02:44:017 (UTC)
1TIDATETIME    Date: 2021/04/23 at 14:02:44:017
1TITIMEZONE    Timezone: UTC-4 (EDT)
1TINANOTIME    System nanotime: 379202644260787
1TIFILENAME    Javacore filename:    /home/cheesemp/test/javacore.20210423.140244.18885.0003.txt
1TIREQFLAGS    Request Flags: 0x81 (exclusive+preempt)
1TIPREPSTATE   Prep State: 0x104 (exclusive_vm_access+trace_disabled)
```

The MEMINFO section records how much memory is allocated to the Java heap (`1STHEAPTYPE Object Memory`), how much is in use, and how much is fre

If you don't know what size the Java heap was set to, you might find that information in the ENVINFO section, which records the command-line options that `2CIUSERARG`. The Java heap size is set by the `-Xmx` option. If the size has not been set on the command line by `-Xmx`, the default value applies, which you c

In this scenario, the solution to the problem is not an adjustment to the Java heap size. Here is the MEMINFO section:

```
0SECTION       MEMINFO subcomponent dump routine
NULL           ================================
NULL
1STHEAPTYPE    Object Memory
NULL           id         start      end        size       space/region
1STHEAPSPACE   0xB6B49D20     --         --         --     Generational
1STHEAPREGION  0xB6B4A078 0x95750000 0xB5470000 0x1FD20000 Generational/Tenured Region
1STHEAPREGION  0xB6B49F10 0xB5470000 0xB54C0000 0x00050000 Generational/Nursery Region
1STHEAPREGION  0xB6B49DA8 0xB54C0000 0xB5750000 0x00290000 Generational/Nursery Region
NULL
1STHEAPTOTAL   Total memory:         536870912 (0x20000000)
1STHEAPINUSE   Total memory in use:  302603160 (0x12095B98)
1STHEAPFREE    Total memory free:    234267752 (0x0DF6A468)
```

The output shows that only 56% of the Java heap is in use, so this suggests that the application is trying to do something suboptimal. To investigate further, **thread** in the THREADS section. Here is an extract from the output:

```
0SECTION       THREADS subcomponent dump routine
NULL           ================================
NULL
1XMPOOLINFO    JVM Thread pool info:
2XMPOOLTOTAL       Current total number of pooled threads: 16
2XMPOOLLIVE        Current total number of live threads: 16
2XMPOOLDAEMON      Current total number of live daemon threads: 15
NULL
1XMCURTHDINFO  Current thread
3XMTHREADINFO      "main" J9VMThread:0xB6B60C00, omrthread_t:0xB6B049D8, java/lang/Thread:0x95764520, state:R, prio=5
3XMJAVALTHREAD            (java/lang/Thread getId:0x1, isDaemon:false)
3XMTHREADINFO1            (native thread ID:0x49C6, native priority:0x5, native policy:UNKNOWN, vmstate:R, vm thread flags:0x00001020)
3XMTHREADINFO2            (native stack address range from:0xB6CB5000, to:0xB74B6000, size:0x801000)
3XMCPUTIME               CPU usage total: 8.537823831 secs, current category="Application"
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=0 (0x0)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at java/lang/StringBuffer.ensureCapacityImpl(StringBuffer.java:696)
4XESTACKTRACE                at java/lang/StringBuffer.append(StringBuffer.java:486(Compiled Code))
5XESTACKTRACE                   (entered lock: java/lang/StringBuffer@0x957645B8, entry count: 1)
4XESTACKTRACE                at java/lang/StringBuffer.append(StringBuffer.java:428(Compiled Code))
4XESTACKTRACE                at HeapBreaker.main(HeapBreaker.java:34(Compiled Code))
3XMTHREADINFO3           Native callstack:
4XENATIVESTACK               (0xB6C535B3 [libj9prt29.so+0x3b5b3])
4XENATIVESTACK               (0xB6C36F3E [libj9prt29.so+0x1ef3e])
4XENATIVESTACK               (0xB6C5311E [libj9prt29.so+0x3b11e])
```

```
4XENATIVESTACK               (0xB6C53216 [libj9prt29.so+0x3b216])
4XENATIVESTACK               (0xB6C52CE3 [libj9prt29.so+0x3ace3])
4XENATIVESTACK               (0xB6C36F3E [libj9prt29.so+0x1ef3e])
4XENATIVESTACK               (0xB6C52C57 [libj9prt29.so+0x3ac57])
4XENATIVESTACK               (0xB6C4E9CD [libj9prt29.so+0x369cd])
4XENATIVESTACK               (0xB6C502FA [libj9prt29.so+0x382fa])
```

To simulate a Java `OutOfMemoryError`, this example application repeatedly appends characters to a `StringBuffer` object in an infinite loop. The Java ca `java/lang/StringBuffer.ensureCapacityImpl()` throws the `OutOfMemoryError`.

StringBuffer objects are wrappers for character arrays (`char[]`) and when the capacity of the underlying array is reached, the contents are automatically cop scenario, the array takes up all the remaining space in the Java heap.

The MEMINFO section of the Java dump file can also tell you when an unexpectedly large allocation request causes an OOM. Look for the GC History (1STGC triggered a global GC. When the GC could not free up sufficient space in the heap to satisfy the request, the allocation failure generated the OOM.

```
1STGCHTYPE     GC History
3STHSTTYPE     14:29:29:580239000 GMT j9mm.101 -   J9AllocateIndexableObject() returning NULL! 0 bytes requested for object of class B6BBC300 from memory
3STHSTTYPE     14:29:29:579916000 GMT j9mm.134 -   Allocation failure end: newspace=2686912/3014656 oldspace=231597224/533856256 loa=5338112/5338112
3STHSTTYPE     14:29:29:579905000 GMT j9mm.470 -   Allocation failure cycle end: newspace=2686912/3014656 oldspace=231597224/533856256 loa=5338112/533811
3STHSTTYPE     14:29:29:579859000 GMT j9mm.475 -   GlobalGC end: workstackoverflow=0 overflowcount=0 memory=234284136/536870912
3STHSTTYPE     14:29:29:579807000 GMT j9mm.90 -   GlobalGC collect complete
3STHSTTYPE     14:29:29:579776000 GMT j9mm.137 -   Compact end: bytesmoved=301989896
3STHSTTYPE     14:29:29:313899000 GMT j9mm.136 -   Compact start: reason=compact to meet allocation
3STHSTTYPE     14:29:29:313555000 GMT j9mm.57 -   Sweep end
3STHSTTYPE     14:29:29:310772000 GMT j9mm.56 -   Sweep start
3STHSTTYPE     14:29:29:310765000 GMT j9mm.94 -   Class unloading end: classloadersunloaded=0 classesunloaded=0
3STHSTTYPE     14:29:29:310753000 GMT j9mm.60 -   Class unloading start
3STHSTTYPE     14:29:29:310750000 GMT j9mm.55 -   Mark end
3STHSTTYPE     14:29:29:306013000 GMT j9mm.54 -   Mark start
3STHSTTYPE     14:29:29:305957000 GMT j9mm.474 -   GlobalGC start: globalcount=9
3STHSTTYPE     14:29:29:305888000 GMT j9mm.475 -   GlobalGC end: workstackoverflow=0 overflowcount=0 memory=234284136/536870912
3STHSTTYPE     14:29:29:305837000 GMT j9mm.90 -   GlobalGC collect complete
3STHSTTYPE     14:29:29:305808000 GMT j9mm.137 -   Compact end: bytesmoved=189784
3STHSTTYPE     14:29:29:298042000 GMT j9mm.136 -   Compact start: reason=compact to meet allocation
3STHSTTYPE     14:29:29:297695000 GMT j9mm.57 -   Sweep end
3STHSTTYPE     14:29:29:291696000 GMT j9mm.56 -   Sweep start
3STHSTTYPE     14:29:29:291692000 GMT j9mm.55 -   Mark end
3STHSTTYPE     14:29:29:284994000 GMT j9mm.54 -   Mark start
3STHSTTYPE     14:29:29:284941000 GMT j9mm.474 -   GlobalGC start: globalcount=8
3STHSTTYPE     14:29:29:284916000 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.016 meanexclusiveaccessms=0.016 threads=0 lastthreadtid=0xB6B611
3STHSTTYPE     14:29:29:284914000 GMT j9mm.469 -   Allocation failure cycle start: newspace=2678784/3014656 oldspace=80601248/533856256 loa=5338112/533811
3STHSTTYPE     14:29:29:284893000 GMT j9mm.470 -   Allocation failure cycle end: newspace=2678784/3014656 oldspace=80601248/533856256 loa=5338112/5338112
3STHSTTYPE     14:29:29:284858000 GMT j9mm.560 -   LocalGC end: rememberedsetoverflow=0 causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=
3STHSTTYPE     14:29:29:284140000 GMT j9mm.140 -   Tilt ratio: 89
3STHSTTYPE     14:29:29:283160000 GMT j9mm.64 -   LocalGC start: globalcount=8 scavengecount=335 weakrefs=0 soft=0 phantom=0 finalizers=0
3STHSTTYPE     14:29:29:283123000 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.016 meanexclusiveaccessms=0.016 threads=0 lastthreadtid=0xB6B611
3STHSTTYPE     14:29:29:283120000 GMT j9mm.469 -   Allocation failure cycle start: newspace=753616/3014656 oldspace=80601248/533856256 loa=5338112/533811
3STHSTTYPE     14:29:29:283117000 GMT j9mm.133 -   Allocation failure start: newspace=753616/3014656 oldspace=80601248/533856256 loa=5338112/5338112 requ
3STHSTTYPE     14:29:29:269762000 GMT j9mm.134 -   Allocation failure end: newspace=2686928/3014656 oldspace=80601248/533856256 loa=5338112/5338112
3STHSTTYPE     14:29:29:269751000 GMT j9mm.470 -   Allocation failure cycle end: newspace=2686976/3014656 oldspace=80601248/533856256 loa=5338112/5338112
3STHSTTYPE     14:29:29:269718000 GMT j9mm.560 -   LocalGC end: rememberedsetoverflow=0 causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=
3STHSTTYPE     14:29:29:268981000 GMT j9mm.140 -   Tilt ratio: 89
3STHSTTYPE     14:29:29:268007000 GMT j9mm.64 -   LocalGC start: globalcount=8 scavengecount=334 weakrefs=0 soft=0 phantom=0 finalizers=0
3STHSTTYPE     14:29:29:267969000 GMT j9mm.135 -   Exclusive access: exclusiveaccessms=0.016 meanexclusiveaccessms=0.016 threads=0 lastthreadtid=0xB6B611
3STHSTTYPE     14:29:29:267966000 GMT j9mm.469 -   Allocation failure cycle start: newspace=0/3014656 oldspace=80601248/533856256 loa=5338112/5338112 requ
3STHSTTYPE     14:29:29:267963000 GMT j9mm.133 -   Allocation failure start: newspace=0/3014656 oldspace=80601248/533856256 loa=5338112/5338112 requested
3STHSTTYPE     14:29:29:249015000 GMT j9mm.134 -   Allocation failure end: newspace=2686928/3014656 oldspace=80601248/533856256 loa=5338112/5338112
```

```
3STHSTTYPE     14:29:29:249003000 GMT j9mm.470 -   Allocation failure cycle end: newspace=2686976/3014656 oldspace=80601248/533856256 loa=5338112/5338112
3STHSTTYPE     14:29:29:248971000 GMT j9mm.560 -   LocalGC end: rememberedsetoverflow=0 causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=
```

Although the Java code that was used in this scenario deliberately triggered an `OutOfMemoryError` in a pronounced way, similar allocation issues can and

The next step in diagnosing the problem is to open the system dump that gets generated automatically when an `OutOfMemoryError` occurs. Open the dum is seeing the same `String` duplicated over and over again, which might indicate that code is stuck in a loop.

Note: If you want to use MAT to analyze your system dump, you must install the Diagnostic Tool Framework for Java (DTFJ) plug-in in the Eclipse IDE. Selec

```
Help > Install New Software > Work with "IBM Diagnostic Tool Framework for Java" >
```

If, unlike the previous scenario, you receive an `OutOfMemoryError` and the MEMINFO section shows that there is very little space left on the Java heap, the might want to increase your Java heap size. For help with this task, see How to do heap sizing.

Native OutOfMemoryError

In this scenario, the VM runs out of native memory. Native memory is memory that is used by the VM for storing all virtualized resources and data that it nee additional limits imposed by the operating system, for example Unix `ulimits`.

When a `NativeOutOfMemoryError` occurs, a Java dump is generated by default. The first section of the file (TITLE) tells you that a systhrow event triggere

```
0SECTION       TITLE subcomponent dump routine
NULL           ===============================
1TICHARSET     UTF-8
1TISIGINFO     Dump Event "systhrow" (00040000) Detail "java/lang/OutOfMemoryError" "native memory exhausted" received
1TIDATETIMEUTC Date: 2021/04/23 at 18:02:44:017 (UTC)
1TIDATETIME    Date: 2021/04/23 at 14:02:44:017
1TITIMEZONE    Timezone: UTC-4 (EDT)
1TINANOTIME    System nanotime: 379202644260787
1TIFILENAME    Javacore filename:    /home/cheesemp/test/javacore.20210423.140244.19708.0003.txt
1TIREQFLAGS    Request Flags: 0x81 (exclusive+preempt)
1TIPREPSTATE   Prep State: 0x104 (exclusive_vm_access+trace_disabled)
```

Sometimes, the current thread is responsible for causing the `NativeOutOfMemoryError`. Information about the current thread can be found in the THREA

```
0SECTION       THREADS subcomponent dump routine
NULL           ================================
NULL
1XMPOOLINFO    JVM Thread pool info:
2XMPOOLTOTAL       Current total number of pooled threads: 16
2XMPOOLLIVE        Current total number of live threads: 16
2XMPOOLDAEMON      Current total number of live daemon threads: 15
NULL
1XMCURTHDINFO  Current thread
3XMTHREADINFO      "main" J9VMThread:0xB6C60C00, omrthread_t:0xB6C049D8, java/lang/Thread:0xB55E3C10, state:R, prio=5
3XMJAVALTHREAD            (java/lang/Thread getId:0x1, isDaemon:false)
3XMTHREADINFO1            (native thread ID:0x4CFD, native priority:0x5, native policy:UNKNOWN, vmstate:R, vm thread flags:0x00001020)
3XMTHREADINFO2            (native stack address range from:0xB6D4E000, to:0xB754F000, size:0x801000)
3XMCPUTIME               CPU usage total: 3.654896026 secs, current category="Application"
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=0 (0x0)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at sun/misc/Unsafe.allocateDBBMemory(Native Method)
4XESTACKTRACE                at java/nio/DirectByteBuffer.<init>(DirectByteBuffer.java:127(Compiled Code))
4XESTACKTRACE                at java/nio/ByteBuffer.allocateDirect(ByteBuffer.java:311)
4XESTACKTRACE                at NativeHeapBreaker.main(NativeHeapBreaker.java:9)
3XMTHREADINFO3           Native callstack:
4XENATIVESTACK               (0xB6A9F5B3 [libj9prt29.so+0x3b5b3])
...
```

```
4XENATIVESTACK               (0xB582CC9C [libjclse7b_29.so+0x40c9c])
4XENATIVESTACK               Java_sun_misc_Unsafe_allocateDBBMemory+0x88 (0xB5827F6B [libjclse7b_29.so+0x3bf6b])
4XENATIVESTACK               (0x94A2084A [<unknown>+0x0])
4XENATIVESTACK               (0xB6B2538B [libj9vm29.so+0x6c38b])
4XENATIVESTACK               (0xB6B4074C [libj9vm29.so+0x8774c])
4XENATIVESTACK               (0xB6B7F299 [libj9vm29.so+0xc6299])
4XENATIVESTACK               (0xB6A82F3E [libj9prt29.so+0x1ef3e])
4XENATIVESTACK               (0xB6B7F32A [libj9vm29.so+0xc632a])
4XENATIVESTACK               (0xB6B4084C [libj9vm29.so+0x8784c])
4XENATIVESTACK               (0xB6B3CD0C [libj9vm29.so+0x83d0c])
4XENATIVESTACK               (0xB776F87D [libjli.so+0x787d])
4XENATIVESTACK               (0xB7784F72 [libpthread.so.0+0x6f72])
4XENATIVESTACK               clone+0x5e (0xB76A043E [libc.so.6+0xee43e])
```

For clarity in the `Native callstack` output, `...` indicates that some lines are removed.

The Java call stack shows the transition from Java to native code (`sun/misc/Unsafe.allocateDBBMemory(Native Method)`), indicating a request for storage is the likely culprit for this `NativeOutOfMemoryError`.

The next step is to investigate the NATIVEMEMINFO section of the Java dump file, which reports the amount of memory used by the JRE process, broken do

```
0SECTION       NATIVEMEMINFO subcomponent dump routine
NULL           ================================
0MEMUSER
1MEMUSER       JRE: 3,166,386,688 bytes / 4388 allocations
1MEMUSER       |
2MEMUSER       +--VM: 563,176,824 bytes / 1518 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Classes: 3,104,416 bytes / 120 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Memory Manager (GC): 548,181,888 bytes / 398 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Java Heap: 536,932,352 bytes / 1 allocation
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Other: 11,249,536 bytes / 397 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Threads: 10,817,120 bytes / 147 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Java Stack: 115,584 bytes / 16 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Native Stack: 10,616,832 bytes / 17 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Other: 84,704 bytes / 114 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Trace: 163,688 bytes / 268 allocations
2MEMUSER       |  |
3MEMUSER       |  +--JVMTI: 17,320 bytes / 13 allocations
2MEMUSER       |  |
3MEMUSER       |  +--JNI: 23,296 bytes / 55 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Port Library: 8,576 bytes / 74 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Other: 860,520 bytes / 443 allocations
1MEMUSER       |
2MEMUSER       +--JIT: 3,744,728 bytes / 122 allocations
2MEMUSER       |  |
3MEMUSER       |  +--JIT Code Cache: 2,097,152 bytes / 1 allocation
```

```
2MEMUSER       |  |
3MEMUSER       |  +--JIT Data Cache: 524,336 bytes / 1 allocation
2MEMUSER       |  |
3MEMUSER       |  +--Other: 1,123,240 bytes / 120 allocations
1MEMUSER       |
2MEMUSER       +--Class Libraries: 2,599,463,024 bytes / 2732 allocations
2MEMUSER       |  |
3MEMUSER       |  +--Harmony Class Libraries: 1,024 bytes / 1 allocation
2MEMUSER       |  |
3MEMUSER       |  +--VM Class Libraries: 2,599,462,000 bytes / 2731 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--sun.misc.Unsafe: 2,598,510,480 bytes / 2484 allocations
4MEMUSER       |  |  |  |
5MEMUSER       |  |  |  +--Direct Byte Buffers: 2,598,510,480 bytes / 2484 allocations
3MEMUSER       |  |  |
4MEMUSER       |  |  +--Other: 951,520 bytes / 247 allocations
1MEMUSER       |
2MEMUSER       +--Unknown: 2,112 bytes / 16 allocations
NULL
```

In the `VM Class Libraries` section, the amount of memory allocated for `Direct Byte Buffers` is shown. Because the `NativeOutOfMemoryError` v process might have run out of memory because of the `ulimit` setting. Increasing the value for `ulimit` might avoid the error, which you can do temporarily b

The theoretical maximum size for a 32-bit process is the size of the 32-bit address space, which is 4 GB. On most operating systems, a portion of the address with a 32-bit VM is quite common.

The same 4 GB limit is also important if you are using a 64-bit VM with compressed references. In compressed references mode, all references to objects, cl operating system might place other allocations within this 4 GB of address space, and if this area becomes sufficiently full or fragmented, the VM throws a na contain more information about what the thread was doing at the VM level when the `NativeOutOfMemoryError` error occurred.

You can usually avoid this type of problem by using the `-Xmcrs` option to reserve a contiguous area of memory within the lowest 4 GB of memory at VM start

Another common cause of a `NativeOutOfMemoryError` is when an application loads duplicate classes. Classes are allocated outside of the Java heap in n Analyzer tool (MAT) can tell you if you have duplicate classes by using the *Class Loader Explorer* feature. Because a system dump is automatically generated

Deadlock

Deadlocks occur when two threads attempt to synchronize on an object and lock an instance of a class. When this happens, your application stops respondir VM.

The VM can detect the most common types of deadlock scenario involving Java monitors. If this type of deadlock is detected, information is provided in the l

Here is the output from the code that was used to cause a common deadlock scenario:

```
NULL
1LKDEADLOCK    Deadlock detected !!!
NULL           ---------------------
NULL
2LKDEADLOCKTHR  Thread "Worker Thread 2" (0x94501D00)
3LKDEADLOCKWTR    is waiting for:
4LKDEADLOCKMON      sys_mon_t:0x08C2B344 infl_mon_t: 0x08C2B384:
4LKDEADLOCKOBJ      java/lang/Object@0xB5666698
3LKDEADLOCKOWN    which is owned by:
2LKDEADLOCKTHR  Thread "Worker Thread 3" (0x94507500)
3LKDEADLOCKWTR    which is waiting for:
4LKDEADLOCKMON      sys_mon_t:0x08C2B3A0 infl_mon_t: 0x08C2B3E0:
4LKDEADLOCKOBJ      java/lang/Object@0xB5666678
3LKDEADLOCKOWN    which is owned by:
2LKDEADLOCKTHR  Thread "Worker Thread 1" (0x92A3EC00)
3LKDEADLOCKWTR    which is waiting for:
```

```
4LKDEADLOCKMON      sys_mon_t:0x08C2B2E8 infl_mon_t: 0x08C2B328:
4LKDEADLOCKOBJ      java/lang/Object@0xB5666688
3LKDEADLOCKOWN    which is owned by:
2LKDEADLOCKTHR  Thread "Worker Thread 2" (0x94501D00)
```

This output tells you that `Worker Thread 2` is waiting for `Worker Thread 3`, which is waiting for `Worker Thread 1`. Because `Worker Thread 1` is als
each of these worker threads, you can trace the problem back to specific lines in your application code.

In this example, you can see from the following output that for all worker threads, the stack traces (4XESTACKTRACE/5XESTACKTRACE) indicate a problem i

```
3XMTHREADINFO      "Worker Thread 1" J9VMThread:0x92A3EC00, omrthread_t:0x92A3C2B0, java/lang/Thread:0xB5666778, state:B, prio=5
3XMJAVALTHREAD            (java/lang/Thread getId:0x13, isDaemon:false)
3XMTHREADINFO1            (native thread ID:0x52CF, native priority:0x5, native policy:UNKNOWN, vmstate:B, vm thread flags:0x00000201)
3XMTHREADINFO2            (native stack address range from:0x9297E000, to:0x929BF000, size:0x41000)
3XMCPUTIME               CPU usage total: 0.004365543 secs, current category="Application"
3XMTHREADBLOCK     Blocked on: java/lang/Object@0xB5666688 Owned by: "Worker Thread 2" (J9VMThread:0x94501D00, java/lang/Thread:0xB56668D0)
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=0 (0x0)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at WorkerThread.run(DeadLockTest.java:35)
5XESTACKTRACE                   (entered lock: java/lang/Object@0xB5666678, entry count: 1)
...
3XMTHREADINFO      "Worker Thread 2" J9VMThread:0x94501D00, omrthread_t:0x92A3C8F0, java/lang/Thread:0xB56668D0, state:B, prio=5
3XMJAVALTHREAD            (java/lang/Thread getId:0x14, isDaemon:false)
3XMTHREADINFO1            (native thread ID:0x52D0, native priority:0x5, native policy:UNKNOWN, vmstate:B, vm thread flags:0x00000201)
3XMTHREADINFO2            (native stack address range from:0x946BF000, to:0x94700000, size:0x41000)
3XMCPUTIME               CPU usage total: 0.004555580 secs, current category="Application"
3XMTHREADBLOCK     Blocked on: java/lang/Object@0xB5666698 Owned by: "Worker Thread 3" (J9VMThread:0x94507500, java/lang/Thread:0xB5666A18)
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=0 (0x0)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at WorkerThread.run(DeadLockTest.java:35)
5XESTACKTRACE                   (entered lock: java/lang/Object@0xB5666688, entry count: 1)
...
3XMTHREADINFO      "Worker Thread 3" J9VMThread:0x94507500, omrthread_t:0x92A3CC10, java/lang/Thread:0xB5666A18, state:B, prio=5
3XMJAVALTHREAD            (java/lang/Thread getId:0x15, isDaemon:false)
3XMTHREADINFO1            (native thread ID:0x52D1, native priority:0x5, native policy:UNKNOWN, vmstate:B, vm thread flags:0x00000201)
3XMTHREADINFO2            (native stack address range from:0x9467E000, to:0x946BF000, size:0x41000)
3XMCPUTIME               CPU usage total: 0.003657010 secs, current category="Application"
3XMTHREADBLOCK     Blocked on: java/lang/Object@0xB5666678 Owned by: "Worker Thread 1" (J9VMThread:0x92A3EC00, java/lang/Thread:0xB5666778)
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=0 (0x0)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at WorkerThread.run(DeadLockTest.java:35)
5XESTACKTRACE                   (entered lock: java/lang/Object@0xB5666698, entry count: 1)
```

Hang

An application can hang for a number of reasons but the most common cause is excessive global garbage collection (GC) activity, where your application is re
`-verbose:gc` option.

Deadlock situations can also manifest themselves as hangs. For more information on diagnosing this type of problem from a Java dump, see the deadlock sc

If you have eliminated verbose GC activity and deadlocks, another common hang scenario involves threads that compete and wait for Java object locks. This
threads are waiting for, but it doesn't release the lock for some reason.

The first place to look in the Java dump output is the **LOCKS** section. This section lists all the monitors and shows which threads have acquired a lock and wh

In this example scenario, the Java dump **LOCKS** section shows that `Worker Thread 0` (3LKMONOBJECT) has acquired a lock and there are 19 other worker

```
NULL           ------------------------------------------------------------------------
0SECTION       LOCKS subcomponent dump routine
```

```
NULL           ------------------------------
NULL
1LKPOOLINFO    Monitor pool info:
2LKPOOLTOTAL     Current total number of monitors: 1
NULL
1LKMONPOOLDUMP Monitor Pool Dump (flat & inflated object-monitors):
2LKMONINUSE      sys_mon_t:0x92711200 infl_mon_t: 0x92711240:
3LKMONOBJECT       java/lang/Object@0xB56658D8: Flat locked by "Worker Thread 0" (J9VMThread:0x92A3EC00), entry count 1
3LKWAITERQ            Waiting to enter:
3LKWAITER                "Worker Thread 1" (J9VMThread:0x92703F00)
3LKWAITER                "Worker Thread 2" (J9VMThread:0x92709C00)
3LKWAITER                "Worker Thread 3" (J9VMThread:0x92710A00)
3LKWAITER                "Worker Thread 4" (J9VMThread:0x92717F00)
3LKWAITER                "Worker Thread 5" (J9VMThread:0x9271DC00)
3LKWAITER                "Worker Thread 6" (J9VMThread:0x92723A00)
3LKWAITER                "Worker Thread 7" (J9VMThread:0x92729800)
3LKWAITER                "Worker Thread 8" (J9VMThread:0x92733700)
3LKWAITER                "Worker Thread 9" (J9VMThread:0x92739400)
3LKWAITER                "Worker Thread 10" (J9VMThread:0x92740200)
3LKWAITER                "Worker Thread 11" (J9VMThread:0x92748100)
3LKWAITER                "Worker Thread 12" (J9VMThread:0x9274DF00)
3LKWAITER                "Worker Thread 13" (J9VMThread:0x92754D00)
3LKWAITER                "Worker Thread 14" (J9VMThread:0x9275AA00)
3LKWAITER                "Worker Thread 15" (J9VMThread:0x92760800)
3LKWAITER                "Worker Thread 16" (J9VMThread:0x92766600)
3LKWAITER                "Worker Thread 17" (J9VMThread:0x9276C300)
3LKWAITER                "Worker Thread 18" (J9VMThread:0x92773100)
3LKWAITER                "Worker Thread 19" (J9VMThread:0x92778F00)
NULL
```

The next step is to determine why `Worker Thread 0` is not releasing the lock. The best place to start is the stack trace for this thread, which you can find b

The following extract shows the details for `"Worker Thread 0" (J9VMThread:0x92A3EC00)`:

```
NULL
3XMTHREADINFO      "Worker Thread 0" J9VMThread:0x92A3EC00, omrthread_t:0x92A3C280, java/lang/Thread:0xB56668B8, state:CW, prio=5
3XMJAVALTHREAD            (java/lang/Thread getId:0x13, isDaemon:false)
3XMTHREADINFO1            (native thread ID:0x511F, native priority:0x5, native policy:UNKNOWN, vmstate:CW, vm thread flags:0x00000401)
3XMTHREADINFO2            (native stack address range from:0x9297E000, to:0x929BF000, size:0x41000)
3XMCPUTIME               CPU usage total: 0.000211878 secs, current category="Application"
3XMHEAPALLOC             Heap bytes allocated since last GC cycle=0 (0x0)
3XMTHREADINFO3           Java callstack:
4XESTACKTRACE                at java/lang/Thread.sleep(Native Method)
4XESTACKTRACE                at java/lang/Thread.sleep(Thread.java:941)
4XESTACKTRACE                at WorkerThread.doWork(HangTest.java:37)
4XESTACKTRACE                at WorkerThread.run(HangTest.java:31)
5XESTACKTRACE                   (entered lock: java/lang/Object@0xB56658D8, entry count: 1)
```

In the last line of this output, you can see where the thread acquired the lock. Working up from this line, you can see that `WorkerThread.run` was called, w
preventing the thread from completing its work and releasing the lock. In this example, the `sleep` call was added to induce a hang, but in real-world scenari
another thread.

It is important to remember that each Java dump represents a single snapshot in time. You should generate at least three Java dumps separated by a short p

In this example, the threads do not move and the investigation needs to focus on the logic in `WorkerThread.doWork` to understand why `Worker Thread`

Another common scenario is where each Java dump shows a number of threads waiting for a lock owned by another thread, but the list of waiting threads ar
the same lock. In severe cases, the lock is held only for a small amount of time but there are lots of threads trying to obtain it. Because more time is spent ha
an application design problem. You can use a similar approach to the one used in this scenario to determine which lines of code are responsible for the conte

Note: Content originates from an Eclipse open source project and might contain information about Java levels and platforms that are not supported by the IE