IBM Software Group

Deep Dive on Java Virtual Machine (JVM) Javacores and Javadumps

Kevin Grigorenko (kevin.grigorenko@us.ibm.com)
IBM WebSphere Application Server SWAT
9 February 2010

WebSphere® Support Technical Exchange

Agenda

- Javacores: What, why, and how
- Tips and Tricks
- Hung Thread Detection
- Tooling
- Questions

NOTE: The context of this presentation is problem determination within WebSphere Application Server (WAS); however, most of the information provided herein is applicable to any Java™ process. This presentation will focus primarily on IBM® Javacores which are more detailed than Sun javacores, although the latter will also be covered.

What is a javacore?

- A javacore is a Java diagnostic feature providing a quick, human readable snapshot of a running JVM – a sort of mini-core dump. It is available on both IBM and Sun JVMs.
  - A JVM is a Java Virtual Machine which is the underlying operating system process.
- On the IBM JVM, the javacore is output to a javacore....txt file.
- On the Sun JVM, the javacore is appended to stdout / native_stdout.txt.
- A javacore is also called a “thread dump” and a “javadump.” The official documentation has started referring to javadumps instead of javacores, even though the file produced is still javacore*.
- A javacore usually takes a few hundred milliseconds to create and each IBM javacore is usually ~2MB in size.
- A javacore is very distinct from an operating system core file. A system core is a dump of the entire address space.

IBM JVM Javacore Text File

- 2 columns of output. First column is the "type" of the row, and the second column is the data. For example, at the top you will find the date/time of the javacore and what caused it (in this case, "user" is a caught signal from user space [e.g. kill -QUIT])

```
1TISIGINFO     Dump Event "user" (00004000) received
1TIDATETIME    Date:              2010/01/09 at 12:44:19
1TIFILENAME    Javacore filename:    ...javacore.20100109.184419.3476.0001.txt
```

What's in an IBM Javacore?

- There are significant differences between IBM 1.4 and 5.0+ JVMs.
- An IBM Javacore is so valuable because of the depth and breadth of information it provides with such lower overhead:
  - The signal that produced the javacore (e.g. crash, OOM, manual)
    - GPINFO can also be used to understand which component crashed.
  - The date, time, java version, full command path and arguments, classpath.
  - All the threads in the JVM, including thread state, priority, thread ID, name.
  - Information on classloaders, classes, and shared classes.
  - Information on native and Java locks and monitors.
  - Information on storage.
  - The free and used amounts of the Java heap.
  - The last few garbage collection cycles.

Sun JVM Javacore (stdout)

- Example:

```
2009-11-16 19:35:14
Full thread dump Java HotSpot(TM) Client VM (14.2-b01 mixed mode, sharing):...
"Socrates priority5" prio=6 tid=0x02af8000 nid=0x8e0 waiting for monitor entry [0x02e2f000]
   java.lang.Thread.State: BLOCKED (on object monitor)
        at Philosopher.run(Philosopher.java:25)
        - waiting to lock <0x229c07f0> (a Chopstick)
        - locked <0x229c07e0> (a Chopstick)...

JNI global references: 606
Found one Java-level deadlock:
…
Heap
 def new generation   total 960K, used 257K [0x22990000, 0x22a90000, 0x22e70000)
  eden space 896K,  28% used [0x22990000, 0x229d0578, 0x22a70000)
  from space 64K,   0% used [0x22a70000, 0x22a70000, 0x22a80000)
  to   space 64K,   0% used [0x22a80000, 0x22a80000, 0x22a90000)
 tenured generation   total 4096K, used 0K [0x22e70000, 0x23270000, 0x26990000)
   the space 4096K,   0% used [0x22e70000, 0x22e70000, 0x22e70200, 0x23270000)
 compacting perm gen  total 12288K, used 75K [0x26990000, 0x27590000, 0x2a990000)
   the space 12288K,   0% used [0x26990000, 0x269a2fa0, 0x269a3000, 0x27590000)
    ro space 8192K,  63% used [0x2a990000, 0x2aea9920, 0x2aea9a00, 0x2b190000)
    rw space 12288K,  53% used [0x2b190000, 0x2b804dd0, 0x2b804e00, 0x2bd90000)
```

How is a javacore created?

- POSIX environments (AIX, Linux®, Solaris, HP/UX, z/OS, i5/OS, etc.):
  - kill -3 $PID
  - kill -QUIT `cat server1.pid`
  - http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/topic/com.ibm.java.doc.diagnostics.50/diag/tools/javadump_trigger.html
- Windows®: Unsupported Utility: SendSignal.exe %PID%
  - FOR /F "delims=" %i IN ('type server1.pid') DO @set PID=%i
    SendSignal.exe %PID%
  - http://www.latenighthacking.com/projects/2003/sendSignal/
- Run “startServer.bat -script”, and run from the command line and use Ctrl+BREAK on Windows, CTRL+\ on POSIX, or CTRL+V on z/OS.
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/rxml_startserver.html
- wsadmin: AdminControl.invoke(AdminControl.completeObjectName("type=JVM,process=server1,*"), "dumpThreads")
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.doc/info/ae/ae/rxml_commandline.html
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.javadoc.doc/public_html/mbeandocs/JVM.html

How is a javacore created? (continued)

- On IBM JVMs only:
  - Using Java code (for example, from a JSP): com.ibm.jvm.Dump.JavaDump()
  - The data in a javacore can be extracted from a jextracted system dump
    - http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/topic/com.ibm.java.doc.diagnostics.50/diag/tools/dump_viewer_jdmpview/jextract.html
    - http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/topic/com.ibm.java.doc.diagnostics.50/diag/tools/dump_viewer_jdmpview/dumpviewer_jdmpview.html
    - http://www.ibm.com/developerworks/java/jdk/tools/dumpanalyzer/
  - Using -Xdump dump agents with other signals
    - http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/topic/com.ibm.java.doc.diagnostics.50/diag/tools/dumpagents_syntax.html
  - Using -Xtrace:trigger
    - To get a javacore the first time method MyClass.foo is invoked: -Xtrace:trigger=method{com/test/MyClass.foo,javadump,,1}
    - http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/topic/com.ibm.java.doc.diagnostics.50/diag/tools/trace_options_trigger.html
  - The JVM attempts a javacore when it ends abnormally.
  - z/OS: MODIFY $CONTROLLER,JAVACORE or extract from SVCDUMP
  - To operator console: MODIFY $CONTROLLER,STACKTRACE
    - http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/rxml_mvsmodify.html

How is a javacore created? (continued)

- On Sun JVMs only:
  - Using Jconsole
    - http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html
  - Using JMX bean java.lang.management.ThreadMXBean
    - http://java.sun.com/j2se/1.5.0/docs/api/java/lang/management/ThreadMXBean.html
  - Using VisualVM
    - http://java.sun.com/javase/6/docs/technotes/guides/visualvm/index.html
  - jstack -m $PID
    - http://java.sun.com/javase/6/webnotes/trouble/TSG-VM/html/tooldescr.html#gblfh
  - If the Java process is started with the -XX:+PrintClassHistogram command-line option, then the Ctrl-Break handler will produce a heap histogram.
    - http://java.sun.com/javase/6/webnotes/trouble/TSG-VM/html/hangloop.html
  - More info:
    - http://java.sun.com/javase/6/webnotes/trouble/TSG-VM/html/tooldescr.html#gbmpn
    - http://java.sun.com/developer/technicalArticles/Programming/Stacktrace/

Where does a javacore go?

- Easiest way to tell is to check native_stderr.log:
  - JDK 5: JVMDUMP010I Java Dump written to /usr/WebS...
  - JDK 1.4: JVMDG304: Java core file written to /usr/WebS...
- A javacore is usually no more than 3 megabytes in size and there needs to be sufficient hard drive space available.
- Written to (in the following order, based on availability):
  - IBM_JAVACOREDIR environment variable
  - Current working directory (WAS profile directory)
  - TMPDIR environment variable
  - /tmp on POSIX OSes, or X:\tmp on Windows (where X is the current working drive)
  - Windows only: Appended to STDERR

What's in a javacore's name?

- An example javacore file name is: javacore.20090909.154950.5544.0001.txt
- ON JDK 5+, the basic form is: javacore.%Y%m%d.%H%M%S.%pid.%seq.txt
  - %Y = year, %m = month, %d = day
  - %H = hours, %M = minutes, %S = seconds
  - %pid = PID of the JVM
  - %seq = The Nth output for this PID since the JVM has started.
  - Note that %seq is not available on JDK 1.4
- On AIX and JDK 1.4: javacore%PID.%TIME.txt
  - %TIME is the number of seconds since 1/1/1970

How do I turn off Javacores?

- Why would you want to?
  - -Xdump:java:none
  - DISABLE_JAVADUMP environment variable
- Sometimes you may want to take a javadump on a different signal other than SIGQUIT. For SIGABRT (6), use -Xdump to remove other options on abort, and then use:
  - -Xdump:java:events=abort
    - Use JAVA_DUMP_OPTS environment variable on JDK 1.4

Javacores and MustGathers

- As discussed, javacores are lightweight, small in size, incredibly valuable, and have a small overhead to create. It is advisable to get one or multiple javacores separated by 60 seconds any time a problem occurs.
- WAS MustGathers: http://www-01.ibm.com/support/docview.wss?uid=swg21145599
  - Javacores are requested for hangs, performance problems and High CPU problems.
  - They are also requested with crashes, memory leaks, OutOfMemoryErrors, and classloader issues.
- Javacores are also valuable in many other situations. For example, if a customer isn't sure if a JVMTI agent is attached, but we see high exclusive access times, the javacore will prove if there is an agent attached.

Javacores and MustGathers (continued)

- The common thing to do is to take 3 javacores, 1 minute apart on any JVMs which may be related to the problem. The reason this is done is to better understand the activity within a JVM.
  - This is an art more than a science. For example, for the high CPU MustGather, a javacore is taken before and after the TPROF, but not during because there is a lot of native activity during a javacore so that's all a TPROF would show. The TPROF is then used to map to the native thread ID (DBX is no longer needed in Java 5), but as mentioned, there is not a direct correlation. This is one example where it is more of an art than a science. Nevertheless it is incredibly valuable information.

Tips and Tricks

- When an OutOfMemoryError occurs, a javacore will also be produced. Search for “Current Thread” to find the exact stack that caused the OOM.
- Use javacores to understand what parameters the JVM is running with, such as -Xmx, -Xgcpolicy, etc.
  - In JDK 1.4 and recent releases of 5, you can also see environment variables
- Javacores include the JVM version.
- Check how much space is free versus allocated (both in hex):
  - 1STHEAPFREE   Bytes of Heap Space Free: 3f7c38
  - 1STHEAPALLOC   Bytes of Heap Space Allocated: 3200000
- Note that the heap shrinks and grows, so you may catch the JVM right when bytes of heap space free is at or near 0, but bytes of heap space allocated is not at -Xmx. This means the JVM is about to expand the heap.
  - Furthermore, with -Xgcpolicy:gencon, sometimes scavengers fall behind and a full GC is only run at or near -Xmx, so you may not be out of heap. Check verbosegc.

Tips and Tricks

- Of course, the most useful part of a javacore is the thread details section. This lists all Java thread stacks. In some release of the JDK, native thread stacks are shown as well.
  - Search for “Current Thread” to see what was running at the time of the javacore. Often times this may be the “Signal Dispatcher” thread which took control of the JVM after handling a signal and then dispatched the Javacore creation; therefore, it was not the thread running right before the signal.
- Example
  - 3XMTHREADINFO "WebContainer : 1" (TID:0x16A72E00, sys_thread_t:0x14700D40, state:CW, native ID:0x000011F0) prio=5 4XESTACKTRACE at com/ibm/InfiniteLoop.service(InfiniteLoop.java:35)
- There are a lot of threads, but usually only worry about WebContainer, EJB, MDBs, or wherever your application does work.
- Various thread identifiers. Some represent OS native threads, some internal data structures, etc.

Tips and Tricks

Thread State is important:

R - Runnable - the thread is able to run when given the chance.

CW - Condition Wait - the thread is waiting. For example, because:

A sleep() call is made

The thread has been blocked for I/O

A wait() method is called to wait on a monitor being notified

The thread is synchronizing with another thread with a join() call

S – Suspended – the thread has been suspended by another thread.

Z – Zombie – the thread has been killed.

P – Parked – the thread has been parked by the new concurrency API (java.util.concurrent).

B – Blocked – the thread is waiting to obtain a lock that something else currently owns.

http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/topic/com.ibm.java.doc.diagnostics.50/diag/tools/javadump_tags_threads.html

Tips and Tricks

- The thread name in double quotes can tell a lot.
  - The identifier after the colon is, for many thread groups, a monotonically increasing number. A high number suggests high destroy/create rates, or a thread leak.
- Common tops of stacks in include:
  - Object.wait: Waiting for a notify from another thread (notify leaks can occur)
  - Thread.sleep: Operating system call to sleep for X ms
  - AsyncLibrary.aio_getioev2: Async I/O waiting on new work
  - SocketInputStream.socketRead0: Socket established, processing or waiting for data
- If an error occurs while writing some or all of the javacore, <ERROR> will often appear. Also, if any address is unknown or malformed, it may appear as 0xDEADBEEF.

Tips and Tricks

- On Java 5, you may see threads in a Javacore which are in Conditional Wait (CW) state that you would expect to be Runnable (R). This is “by design” for the new IBM 5 JVM. If the top of a thread stack is neither in Object.wait, nor Thread.sleep, nor Thread.join, nor a native method, then the JVM will put the thread into CW state (Conditional Wait) in preparation for the javacore. This is what "Thread public flags mutex lock" means -- it is an internal JVM lock per thread used for thread quiesce. After the javacore, the thread will go back to Runnable. Therefore, in effect, any threads without the aforementioned methods at the top of the stack are running/runnable at the time of the javacore.
  - http://www-01.ibm.com/support/docview.wss?uid=swg21413580
- If you are using -Xgcpolicy:gencon, the heap used/free reported in the Javacore is the cumulative heap used/free in both nursery and tenured spaces. Therefore, a javacore might show 300MB free, but if tenured is only 10MB free and the rest in nursery, you're effectively out of memory. Consult verbosegc.
- Unfortunately, there's no easy way (without a system core dump) to map thread IDs in a javacore to System*.log or trace*.log: http://www-01.ibm.com/support/docview.wss?uid=swg21418557

Tips and Tricks

- Javadumps triggered by a user signal do not usually show a current thread because the signal is handled on a native thread, and the Java threads will be suspended while the Javadump is produced.
- The JVM has built-in deadlock detection. Simply search in the javacore file for “Deadlock detected !!!”

```
1LKDEADLOCK    Deadlock detected !!!
NULL           ---------------------
NULL
2LKDEADLOCKTHR  Thread "Socrates priority5" (0x41602300)
3LKDEADLOCKWTR    is waiting for:
4LKDEADLOCKMON      sys_mon_t:0x0003A6D8 infl_mon_t: 0x0003A718:
4LKDEADLOCKOBJ      Chopstick@004E8FF8/004E9004:
3LKDEADLOCKOWN    which is owned by:
2LKDEADLOCKTHR  Thread "Aristotle priority5" (0x41DF5A00)
3LKDEADLOCKWTR    which is waiting for:
4LKDEADLOCKMON      sys_mon_t:0x0003A628 infl_mon_t: 0x0003A668:
4LKDEADLOCKOBJ      Chopstick@004E9010/004E901C:
3LKDEADLOCKOWN    which is owned by:
2LKDEADLOCKTHR  Thread "Plato priority9" (0x41602700)
3LKDEADLOCKWTR    which is waiting for:
4LKDEADLOCKMON      sys_mon_t:0x0003A680 infl_mon_t: 0x0003A6C0:
4LKDEADLOCKOBJ      Chopstick@004E8FE0/004E8FEC:
3LKDEADLOCKOWN    which is owned by:
2LKDEADLOCKTHR  Thread "Socrates priority5" (0x41602300)
```

  - Courtesy of http://www-01.ibm.com/support/docview.wss?uid=swg27011855

Tips and Tricks

- IBM JVM Deadlock Detection:
  - "Deadlock detected !!!"
- Sun JVM Deadlock Detection:
  - "Found ... Java-level deadlock"
- One interesting strategy to increase the value of javacores is to append a user ID or other identifier to the thread name during application processing. This gives you quick information on who or what was driving the execution of that thread and for what purpose. Be careful to append to the name instead of replacing the name (for example, the effective name should be "WebContainer : 123 – Bob Jones /order.jsp" instead of "Bob Jones /order.jsp")

Monitors

- Monitors, or semaphores, are used to serialize concurrent access to shared data in "critical sections"
- Javacores show two types of monitors: internal and application
- "Application" level monitors:

  -
    ```
    1LKMONPOOLDUMP Monitor Pool Dump (flat & inflated object-monitors):
    2LKMONINUSE      sys_mon_t:0x147A8ED8 infl_mon_t: 0x147A8F18:
    3LKMONOBJECT       java/lang/Object@011CB568/011CB574: <unowned>
    LKNOTIFYQ           Waiting to be notified:
    3LKWAITNOTIFY            "WebContainer : 2" (0x16A73200)
    ```
- Shows the object being synchronized on and the unique address of the object.
- Unowned may mean an internal lock, or waiting on a notify
- Owned can be followed to see what is blocking (use tooling)

Monitors

- If one thread attempts to acquire a lock which is currently held by another thread, then that lock becomes “inflated.” Otherwise, the JVM simply performs a compare-and-swap to acquire the lock (flat).
- "Three-tier" describes a spinlock strategy with 3 tiers of incrementally larger back-off between attempts to acquire the spinlock.
- The term “thin lock” refers to the lock word at the front of the object. In general, the “thin locking” is the thin veneer over classic and OS-level locking, including flat and inflated locks, the 3 tier spinlock strategy, and lock deflation.
  - http://www.research.ibm.com/people/d/dfb/talks/Bacon98ThinTalk.pdf

Native Memory

- Java 5 includes some information about native memory used by the JVM, including JIT code and data, object memory, and internal memory. Useful for JIT leaks.

  -
    ```
    1STSEGTYPE     Internal Memory
    NULL           segment  start    alloc    end      type     bytes
    1STSEGMENT     170AC428 171D3920 171D4368 171E3920
    01000040 10000
    ```
- Great article on native memory includes a script to summarize:
  - http://www.ibm.com/developerworks/aix/library/j-nativememory-aix/index.html
- perl get_memory_use.pl javacore.txt

  -
    ```
    Segment Usage          Reserved Bytes
    Class Memory          73434980
    Internal Memory       23450756
    JIT Code Cache         2621440
    JIT Data Cache        1572864
    Object Memory         52428800
    ```

Class loaders and loaded classes

- The bottom section of a javacore shows all classloaders, their relationships, and all classes and in which classloaders they were loaded.
  - This can be useful when a class is being loaded multiple times, or in the wrong classloader.
  - Use the Classloader Viewer in WAS for further research.
  - See also
    - http://www.ibm.com/developerworks/java/library/j-dclp1/
-
```
2CLTEXTCLLOADER          p---st-- Loader *System*(0x0053EEF0)
3CLNMBRLOADEDLIB         Number of loaded libraries 5
3CLNMBRLOADEDCL          Number of loaded classes 2618
```
- Can tell you how many classes are loaded at peak, for example to tune -Xk in JDK 1.4

Garbage Collection Flight Recorder

- If verbose garbage collection (-verbose:gc) is enabled (which it should be!), then the javadump will contain the last few GC events. For example:
  - 1STGCHTYPE GC History
  - 3STHSTTYPE 18:54:45:862164291 GMT j9mm.134 - Allocation failure end: newspace=16603920/20641792 oldspace=11846880/39321600 loa=1966080/1966080
  - 3STHSTTYPE 18:54:45:862157865 GMT j9mm.139 - Reference count end: weak=10452 soft=4790 phantom=91 threshold=20 maxThreshold=32
  - 3STHSTTYPE 18:54:45:861154512 GMT j9mm.65 - LocalGC end: rememberedsetoverflow=0 causedrememberedsetoverflow=0 scancacheoverflow=0 failedflipcount=0 failedflipbytes=0 failedtenurecount=0 failedtenurebytes=0 flipcount=73143 flipbytes=3748876 newspace=16604448/20641792 oldspace=11846880/39321600 loa=1966080/1966080 tenureage=5
  - 3STHSTTYPE 18:54:45:861146968 GMT j9mm.140 - Tilt ratio: 78
  - 3STHSTTYPE 18:54:45:822633137 GMT j9mm.64 - LocalGC start: globalcount=2 scavengecount=33 weakrefs=11277 soft=4830 phantom=122 finalizers=701

Shared Class Cache

- If -Xshareclasses is enabled, a section in the javacore will provide more information:

```
0SECTION      SHARED CLASSES subcomponent dump routine
NULL          ========================================
NULL
1SCLTEXTCSUM   Cache Summary
NULL          ------------------
NULL
…
2SCLTEXTCSZ       Cache size               = 52428428
2SCLTEXTFRB       Free bytes               = 2910520
…
2SCLTEXTCPF       Cache is 94% full
…
1SCLTEXTCNTD      Cache Name               Memory type          Cache path
NULL
2SCLTEXTCMDT      myclassshare             Memory mapped file      /javasharedresources/C240D2A32P_myclassshare_G07
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
```

WAS Hung Thread Detection

- WAS has a feature called hung thread detection which monitors managed threads, such as the WebContainer thread pool. Any native threads, or threads spawned by an application are not monitored.
- Every 180 seconds (com.ibm.websphere.threadmonitor.interval), WAS checks if any managed thread has been "hung" for more than 600 seconds (com.ibm.websphere.threadmonitor.threshold)
- Since 6.1.0.19, com.ibm.websphere.threadmonitor.dump.java=true may be used to automatically take a javacore on the WSVR0605W message.
- The message can also be subscribed to using JMX.
- See:
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/ttrb_confighangdet.html
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.nd.multiplatform.doc/info/ae/ae/ctrb_hangdetection.html

WAS Hung Thread Detection

- WAS 7+: Same message as in earlier versions but also includes the stack trace of the suspect thread, e.g.:
-
```
[11/16/09 12:41:03:296 PST] 00000020 ThreadMonitor W   WSVR0605W: Thread "WebContainer : 0" (00000021) has been active for 655546 milliseconds and may be hung. There is/are 1 thread(s) in total in the server that may be hung.
    at java.lang.Thread.sleep(Native Method)
    at java.lang.Thread.sleep(Thread.java:851)
    at com.ibm.Sleep.doSleep(Sleep.java:55)
    at com.ibm.Sleep.service(Sleep.java:35)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:831)
    ...
```

WAS Hung Thread Detection

- The way it works is that WAS has its own implementation of Thread Pools. Every X seconds (X=com.ibm.websphere.threadmonitor.interval), a WAS Alarm thread wakes up and iterates over all managed thread pools. A managed thread pool is a thread pool created by WAS, for example the WebContainer thread pool, EJB thread pool, etc. This does not include any spawned threads nor any other threads not part of a managed thread pool. Then each thread pool iterates over all of its Worker threads and subtracts the "start time" of the thread from the current time, and passes it to a monitor. The “start time” is a deceiving term.
- How does the thread pool determines the "start time?" The start time, from the hung thread detection service point of view is really the start of the thread's last dispatch, not the absolute start. Of course, it is initially set to System.currentTimeMillis() when the Worker is dispatched in the thread pool and its run method is called (e.g. a servlet starts). However, the start time is also set or reset (thus as I mention it's not really a "start" time) with various calls, such as in the Async IO between the time when the thread is disptached and when it's waiting on data from the socket, as well as from many other places. In effect, all of these multitude of "other places" impact the view of what is "hung."

WAS Hung Thread Detection

- Therefore, the amount of time the thread has been active is approximate and is based on each containers' ability to accurately reflect a thread's waiting or running state; however, in general, it is the amount of milliseconds that a thread has been dispatched and doing non-blocking “work” (i.e. started or reset to "non waiting" by a container) within a WAS managed thread pool.
- Note that since the "checker" thread fires only every X seconds, there is a potential gap there as you can tell on the detection policy -- i.e. the effective "time resolution" of the hung thread detection. The higher the detection interval, the less accurate.
- In Summary, WAS Hung Thread Detection is a heuristic, and that's why the message says “may be hung.”

z/OS

- A javacore.txt file can be extracted from an SVCDUMP using svcdump.jar
  - https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=diagjava
- Java 5 and greater
  - java -classpath svcdump.jar com.ibm.zebedee.dtfj.j9.PrintJavaCore SVCDUMP
  - This will print to standard out a javacore file for each ASID, appended together.
- The old format for just a list of thread stacks:
  - java -classpath svcdump.jar com.ibm.zebedee.dtfj.PrintThreads SVCDUMP

z/OS

- When you send the modify command to the control region, the javacores are created in the servant regions.
- For many JDK versions, this was an undocumented feature. It is fully documented and supported in WAS 7:
  - http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.zseries.doc/info/zseries/ae/rxml_mvsmodify.html
- In WAS 7, you can also control exactly which jobs create javacores (including the control region)
- The equivalent to native_stdout is SYSOUT and will show where the javacore file was placed on HFS/ZFS
- The location of the javacores can be specified with the _CEE_DMPTARG environment variable:
  - http://publib.boulder.ibm.com/infocenter/javasdk/v5r0/topic/com.ibm.java.doc.diagnostics.50/diag/appendixes/env_var/env_jvm.html

IBM Thread and Monitor Dump Analyzer

- TMDA supports both IBM and Sun/HP JVMs.
- TMDA provides 3 primary values:
  - Summary of the javacore, including any warnings, thread histogram by type and state, memory segment information, etc.
  - User interface for navigating thread stacks and monitors
  - User interface for comparing multiple javacores and their respective thread stacks and monitors
- Available in IBM Support Assistant: http://www.ibm.com/software/support/isa/
- An in-depth user guide and screenshots of common screens may be found either in the embedded TMDA Help system with ISA or at the following WSTE link: http://www-01.ibm.com/support/docview.wss?uid=swg27011855

Summary

- Javacores/Javadumps are a lightweight, mini-core dump of a Java Virtual Machine process.
- If you're having a problem, a javacore will almost always contribute to root cause analysis. You should become very familiar with the various ways of taking, finding, and analyzing javacores.

Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at: http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at: http://www.ibm.com/developerworks/websphere/community/
- Join the Global WebSphere Community: http://www.websphereusergroup.org
- Access key product show-me demos and tutorials by visiting IBM Education Assistant: http://www.ibm.com/software/info/education/assistant
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically: http://www.ibm.com/software/websphere/support/d2w.html
- Sign up to receive weekly technical My Notifications emails: http://www.ibm.com/software/support/einfo.html

We Want to Hear From You!

Tell us about what you want to learn

Suggestions for future topics
Improvements and comments about our webcasts
We want to hear everything you have to say!

Please send your suggestions and comments to:
wsehelp@us.ibm.com

Questions and Answers