Tips for Debugging Tomcat and Web Applications

Coty Sutherland
Coty Sutherland

Software Engineer, Red Hat
JBoss Web Server Project

Supported Tomcat, httpd, and JBoss EAP/JBossWeb for ~3 years

ASF Tomcat committer since late 2016

ASF Member

Fedora tomcat and tomcat-native package co-maintainer since 2015
Agenda

My examples and notes are from a Fedora 30 machine, so there will be Linux-specific tools in use. There are Windows equivalents available.

- Some Helpful Debugging Tools
- General Debugging
- Tomcat is using all my CPU!
- Heap Analysis with Eclipse MAT
- How to get Help with Debugging Tomcat
- Questions?
Helpful Tools for Debugging

● General Debugging:
  ○ Tomcat Log Files
  ○ Integrated Development Environment (IDE)
  ○ The Java Debugger (JDB) (not super great, but useful)
  ○ Java Management Extensions (JMX)

● For capturing thread dumps:
  ○ jstack
  ○ `kill -3`

● For analyzing thread dumps:
  ○ Text Editor (like Gedit or ViM)
  ○ Samurai
  ○ Thread Dump Analyzer (TDA)

● For capturing/analyzing heap dumps:
  ○ Eclipse Memory Analyzer (MAT)
General Debugging
Tomcat Log Files

Output found in $CATALINA_HOME/logs
- catalina.out and catalina.$(date).log - container log, most tomcat core logging
- localhost.$(date).log - Host log (default name), most internal errors logged here
- localhost_access_log.$(date).log - access log equivalent to httpd’s access_log. Valve defined in the server.xml.
- manager.$(date).log and host-manager.$(date).log

Configuration
- $CATALINA_HOME/conf/logging.properties
Integrated Development Environment (IDE)

- **IDE Examples:**
  - IntelliJ IDEA (my current favorite)
  - Eclipse
  - Visual Studio Code aka VS Code

- Sort of a pain to configure, but tomcat ships with some helpful config files nowadays (e.g. res/ide-support/tomcat.iml for IntelliJ)

- Run tomcat in debug mode from an IDE and break, examine wherever you’d like in the IDE’s GUI.
  - stop at org.apache.catalina.servlets.DefaultServlet:497
  - curl localhost:8080/badapp/

- Note that when the breakpoint is hit, you can see the thread stack too...I’ll show this again in a bit from a heap dump
The Java Debugging (JDB)

- To use JDB you have to start tomcat in debug mode and then attach to it with JDB.
- I have some handy functions defined in my .bashrc for me to do this quickly, when needed:

```bash
function start-debug() {
  if ! [ -e output/build/bin/setenv.sh ]; then
    echo "export JPDA_SUSPEND="y"
" > output/build/bin/setenv.sh
  fi
  output/build/bin/catalina.sh jpda start;
}
function jdb-attach() {
  jdb -attach 8000 -sourcepath java/
}
```
After starting and attaching, you can set your breakpoint and continue.

```java
[csummer@localhost tomcat]$ jdb-attach
Set uncaught java.lang.Throwables
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
> stop at org.apache.catalina.svlets.DefaultServlet:497
Set breakpoint org.apache.catalina.svlets.DefaultServlet:497
> cont
Nothing suspended.
> Breakpoint hit: "thread=http-nio-8080-exec-2", org.apache.catalina.svlets.DefaultServlet.doGet(), line=497 bci=0
497     serveResource(request, response, true, fileEncoding);
http-nio-8080-exec-2[1] list
493         HttpServletResponse response)
494             throws IOException, ServletException {
495             // Serve the requested resource, including the data content
497     => serveResource(request, response, true, fileEncoding);
498
499         }
500
501         /**
```
Java Management Extensions (JMX)

- JMX is a powerful way to see everything about Tomcat’s JVM in real time
- Local access directly via attaching to the process
- Remote access over a specified (pre-configured) port
- **JMXProxyServlet** which is accessible through the manager webapp
- JConsole is useful for quick access
- There are some helpful frameworks for collecting data via JMX for later debugging:
  - [Jolokia.org](http://Jolokia.org)
  - [Prometheus.io](http://Prometheus.io) and [Prometheus JMX Exporter](http://Prometheus JMX Exporter)
Debugging CPU Issues
Help, Tomcat is using all my CPU!

- Pretty common issue raised in support, “Why is Tomcat using so much CPU time?”
- Generally the problem is in an application (or library) :)
- Some common causes include:
  - Application or library code misbehaving (excessive looping)
  - Excessive Garbage Collection (likely due to an undersized heap)
  - Concurrent access to non thread-safe objects (HashMap, TreeMap, etc)
High CPU, an Example...

In this scenario, we’ve identified that a request to a certain webapp does not complete/is hanging. To determine why the hang is occurring, follow the steps below:

1. Wait for the issue to occur (or reproduce the problem). If you don’t know the problematic app/request, one way to narrow it down is to use the **AccessLogValve** with Time Taken (%D or %T) and looking through the logging to find longer than usual request times.

```
[csutherl@localhost apacheconna-demo]$ curl http://localhost:8880/badapp/loopfor1min.jsp
```
2. When the problem is occurring, use one of the thread dump capture tools mentioned before to capture thread dumps, and also capture CPU data at the same interval. We are using a script that executes `jstack` and `top` in a loop over a 20 second period.
High CPU, an Example... cont’d.

The jstack script looks like this:

```bash
# Number of times to collect data.
LOOP=6
# Interval in seconds between data points.
INTERVAL=20

for ((i=1; i <= $LOOP; i++))
  do
    _now=$(date)
    echo "${_now}" >>high-cpu.out
    top -b -n 1 -H -p $1 >>high-cpu.out
    echo "${_now}" >>high-cpu-tdump.out
    jstack -l $1 >>high-cpu-tdump.out
    echo "thread dump " $i
    if [ $i -lt $LOOP ]; then
      echo "Sleeping..."
      sleep $INTERVAL
    fi
  done
```
3. After capturing the data, check the CPU usage first to identify large consumers.

```
[cmailer@localhost highcpu] grep PID -A3 high-cpu.out

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>29717</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>261428</td>
<td>30216</td>
<td>R</td>
<td>93.8</td>
<td>0.8</td>
<td>1:02.60 http-nio-8080-e</td>
</tr>
<tr>
<td>29679</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>261428</td>
<td>30216</td>
<td>S</td>
<td>0.0</td>
<td>0.8</td>
<td>0:00:00 java</td>
</tr>
<tr>
<td>29680</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>261428</td>
<td>30216</td>
<td>S</td>
<td>0.0</td>
<td>0.8</td>
<td>0:01.21 java</td>
</tr>
</tbody>
</table>

--

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>29717</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>248444</td>
<td>30216</td>
<td>R</td>
<td>99.9</td>
<td>0.8</td>
<td>1:22.86 http-nio-8080-e</td>
</tr>
<tr>
<td>29679</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>248444</td>
<td>30216</td>
<td>S</td>
<td>0.0</td>
<td>0.8</td>
<td>0:00:00 java</td>
</tr>
<tr>
<td>29680</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>248444</td>
<td>30216</td>
<td>S</td>
<td>0.0</td>
<td>0.8</td>
<td>0:01.21 java</td>
</tr>
</tbody>
</table>

--

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>29717</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>248444</td>
<td>30216</td>
<td>R</td>
<td>99.9</td>
<td>0.8</td>
<td>1:43.11 http-nio-8080-e</td>
</tr>
<tr>
<td>29679</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>248444</td>
<td>30216</td>
<td>S</td>
<td>0.0</td>
<td>0.8</td>
<td>0:00:00 java</td>
</tr>
<tr>
<td>29680</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>248444</td>
<td>30216</td>
<td>S</td>
<td>0.0</td>
<td>0.8</td>
<td>0:01.21 java</td>
</tr>
</tbody>
</table>

--

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>29717</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>248444</td>
<td>30216</td>
<td>R</td>
<td>99.9</td>
<td>0.8</td>
<td>2:03.36 http-nio-8080-e</td>
</tr>
<tr>
<td>29679</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>248444</td>
<td>30216</td>
<td>S</td>
<td>0.0</td>
<td>0.8</td>
<td>0:00:00 java</td>
</tr>
<tr>
<td>29680</td>
<td>cmailer</td>
<td>20</td>
<td>0</td>
<td>13.3g</td>
<td>248444</td>
<td>30216</td>
<td>S</td>
<td>0.0</td>
<td>0.8</td>
<td>0:01.21 java</td>
</tr>
</tbody>
</table>
```
High CPU, an Example... cont’d.

4. Now that you know the offending pid/tid (in our example we have one thread that’s consuming CPU) you can find the thread in the thread dump outputs (after converting the decimal value to hex) to see what it's doing.

```
[csutherl@localhost highcpu]$ grep -m 1 0x\$(printf "%x\n" 29717) high-cpu-tdump.out -A10
"http-nio-8080-exec-7" #26 daemon prio=5 os_prio=0 cpu=62733.80ms elapsed=3493.76s tid=0x00007f46108c3800 nid=0x7415
    runnable [0x00007f45b654b000]
    java.lang.Thread.State: RUNNABLE
    at org.apache.jsp.index_jsp._jspService(index_jsp.java:118)
    at org.apache.jasper.runtime.HttpJspBase.service(HttpJspBase.java:70)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
    at org.apache.jasper.servlet.JspServletWrapper.service(JspServletWrapper.java:476)
    at org.apache.jasper.servlet.JspServlet.serviceJspFile(JspServlet.java:385)
    at org.apache.jasper.servlet.JspServlet.service(JspServlet.java:329)
    at javax.servlet.http.HttpServlet.service(HttpServlet.java:741)
    at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:29)
    at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:156)
```
High CPU, an Example... cont’d.

5. Now that you know where the hang is, find it in the code and see why :)
High CPU, an Example... bonus!

CPU usage can also occur due to excessive garbage collection, which you can identify with the same data collection techniques mentioned previously. Here is a shot of excessive GC captured in CPU data (captured with `top -H` to display thread info):

```
[csutherl@localhost gcchurn]$ grep PID -A3 high-cpu.out -m 2

PID USER   PR NI  VIRT  RES SHR S %CPU %MEM TIME+ COMMAND
25233 csutherl 20   0  13.5g 128420 32860 S  53.3  0.4 0:02.62 http-nio-8080-e
25205 csutherl 20   0  13.5g 128420 32860 S  20.0  0.4 0:01.11 G1 Refine#0
25224 csutherl 20   0  13.5g 128420 32860 S  13.3  0.4 0:00.55 GC Thread#3
--

PID USER   PR NI  VIRT  RES SHR S %CPU %MEM TIME+ COMMAND
25233 csutherl 20   0  13.5g 132356 32860 S  56.2  0.4 0:12.55 http-nio-8080-e
25205 csutherl 20   0  13.5g 132356 32860 S  18.8  0.4 0:05.01 G1 Refine#0
25202 csutherl 20   0  13.5g 132356 32860 R  12.5  0.4 0:02.78 GC Thread#0
```
High CPU, an Example... bonus!

If you’re using an older version of top (that doesn’t display thread names), you may need to determine which thread is the problem by examining the thread dump as well.

```bash
[csutherl@localhost gcchurn]$ grep -m 2 0x$(printf '%x\n' 25205) high-cpu-tdump.out
"G1 Refine#0" os_prio=0 cpu=1208.88ms elapsed=13.70s tid=0x00007fa7f01e3000 nid=0x6275 runnable
"G1 Refine#0" os_prio=0 cpu=5047.75ms elapsed=34.00s tid=0x00007fa7f01e3000 nid=0x6275 runnable
```
Quick Look at Samurai and TDA
Debugging Memory Issues
Common Memory Problems

- One of the main problems when it comes to java memory are OutOfMemoryErrors (OOME). There are many different flavors of an OOME:
  - Heap Space
  - PermGen/MetaSpace (Java 8+)
  - “Unable to create new native thread”
  - “GC overhead limit exceeded”
  - Out of swap space
  - Native Memory Exhausted

- We will take a look at an example of a Heap Space OOME and how one could go about debugging one.
Heap Analysis with Eclipse MAT

- In order to capture a heap dump for review, you must first configure tomcat with `-XX:+HeapDumpOnOutOfMemoryError` and restart.
- In our example, we will create an OOME by invoking an application (badapp/oome.jsp) that causes the heap space to become exhausted.

```java
[csutherl@localhost build]$ time curl http://localhost:8080/badapp/oome.jsp
</doctype>
<html lang="en"><head><title>HTTP Status 500 – Internal Server Error</title><style type="text/css">h1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:22px;} h2 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:16px;} h3 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;} body {font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;} b {font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} p {font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size:12px;} a {color:black;} a.name {color:black;} .line {height:1px;background-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 500 – Internal Server Error</h1><hr class="line" />
</p><b>Type</b></p>

Exception Report</p><p><b>Message</b></p>

An exception occurred processing [oome.jsp] at line [7]</p><p><b>Description</b></p>

The server encountered an unexpected condition that prevented it from fulfilling the request.</p><p><b>Exception</b></p>


```
4: ArrayList list = new ArrayList&lt;String&gt;();
5: 6: for (int s = 0; s &lt; 100000000; s++) {
7:  list.add(s);
8:  }
9: %&gt;
```
Heap Analysis with Eclipse MAT, cont’d.

To analyze the heap dump, we can simply open it with MAT. Our heap dump in this example is only ~1G, but depending on memory available, etc you may want to parse the heap dump in the background with MAT’s ParseHeapDump.sh script first.

Eclipse asks if you’d like for it to run the “Leak Suspects” report, which is very helpful :)
Leak Suspects

System Overview

Leaks

Overview

Problem Suspect 1

The thread \texttt{org.apache.tomcat.util.threads.TaskThread @ 0xe2e78730 http-nio-8080-exec-1} keeps local variables with total size \(415,374,872\) (98.26\%) bytes.

The memory is accumulated in one instance of \texttt{java.lang.Object} loaded by \texttt{<system class loader>}. The stacktrace of this Thread is available. See stacktrace.

Keywords

\texttt{java.lang.Object}
Heap Analysis with Eclipse MAT, cont’d.

From the Leak Suspects Report we can see that 98.26% of the heap is being used by a thread named http-nio-8080-exec-1 and that the memory is being accumulated in one instance of java.lang.Object[].

Digging into the report a bit more, we can see that there are > 20 million Integer objects in the Object[].

![Accumulated Objects by Class in Dominator Tree](image-url)
Heap Analysis with Eclipse MAT, cont’d.

Now that we know what sort of objects are sucking up all the memory, you can dig even further into it to trace it to a thread to see where it comes from in the application!
Heap Analysis with Eclipse MAT, cont’d.

And the offending application code is...

```
[csutherl@localhost build]$ awk 'NR == 122, NR == 126 { print NR, $0 }' work/Catalina/localhost/badapp/org/apache/jsp/oome_jsp.java
122
123 for (int s = 0; s < 100000000; s++) {
124    list.add(s);
125 }
126```
Heap Analysis, a bit deeper...

You can also dig further into the large object by using MAT’s Thread Overview and Stacks feature. When using that, you can dig all the way down to see which request cause the issue, and all sorts of other aspects of the problematic object(s).
java_pid9122.hprof

Overview  thread overview [selection of 'TaskThread@0xe2e78730 http-nio-8080-exec-1']

Object / Stack Frame

at java.util.Arrays.copyOfLLL(java.lang.Object[],java.lang.Object[]):3689
at java.util.ArrayList.get():237
at java.util.ArrayList.get():242
at java.util.ArrayList.add():3689
at java.util.ArrayList.add():485
at java.util.ArrayList.add():498

<clinit> org.apache.jsp.oomm.jsp _jspInit 0xe2e789c8 24
<clinit> org.apache.catalina.connector.RequestFacade 0xe2e789e0 16
   <class> class org.apache.catalina.connector.RequestFacade 0xe161e620 8
   request org.apache.catalina.connector.Request 0xe2e78e50 168 29856
      <class> class org.apache.catalina.connector.Request 0xe0defdc48 32 4328
      asyncSupported java.lang.Boolean 0xe0dd7fd0 false 16 16
      connector org.apache.catalina.connector.Connector 0xe0dfa39f8 112 344
      session org.apache.catalina.session.StandardSession 0xe2b4d768 88 328
      coyoteRequest org.apache.coyote.Request 0xe2be0900 176 2888
         <class> class org.apache.coyote.Request 0xe12d518 16 80
         response org.apache.coyote.Response 0xe2be3d58 112 784
         inputBuffer org.apache.coyote.http1.1.HttpInputBuffer 0xe2be0938 88 448
         <class> class org.apache.coyote.http1.1.HttpInputBuffer 0xe1623170 16 215
         wrapper org.apache.tomcat.util.net.NoEndpoint$NoSocketWrapper 0xe2be4b08 160 528
         request org.apache.coyote.Request 0xe2be9090 176 2888
            <class> class org.apache.coyote.Request 0xe102d338 16 80
            response org.apache.coyote.Response 0xe2be2d58 112 784
            inputBuffer org.apache.coyote.http1.1.HttpInputBuffer 0xe2be0938 88 448
            serverNameMB org.apache.tomcat.util.buf.MessageBytes 0xe2be9140 48 144
            schemeMB org.apache.tomcat.util.buf.MessageBytes 0xe2be9200 48 144
            methodMB org.apache.tomcat.util.buf.MessageBytes 0xe2be9290 48 144
            uriMB org.apache.tomcat.util.buf.MessageBytes 0xe1626450 32 48
         <class> class org.apache.coyote.http1.1.HttpInputBuffer 0xe1626430 48 48

byteObj org.apache.tomcat.util.buf.ByteChunk 0xe2bed390 48 48
Heap Analysis, cont’d.

Some things to remember when analyzing heap dumps…

- Problems will not always be as obvious as this one; bad acting applications aren’t always the cause of an OOME.
- Sometimes the heap is just too small.
How to get Help with Debugging Tomcat

● Be prepared to provide as much information as you can. Commons questions that we ask users are:
  ○ Java version
  ○ Tomcat version
  ○ OS details
  ○ Does a particular event/resource trigger the problem?
  ○ How long does the problem last?
  ○ Did the problem start recently (after an update)?

● After you have the information, reach out to the community:
  ○ Mailing list: tomcat-users
  ○ IRC: Freenode #tomcat
Questions?
THANK YOU!

Coty Sutherland

github.com/csutherl
linkedin.com/in/cotysutherland
twitter.com/cotysutherland