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

Red Hat

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 st 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:

```
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/
}
```

JDB, cont’d.

- After starting and attaching, you can set you breakpoint and continue.

```
[csutherl@localhost tomcat]$ jdb-attach
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
Initializing jdb ...
> stop at org.apache.catalina.servlets.DefaultServlet:497
Set breakpoint org.apache.catalina.servlets.DefaultServlet:497
> cont
Nothing suspended.
>
Breakpoint hit: "thread=http-nio-8080-exec-2", org.apache.catalina.servlets.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
496            // Serve the requested resource, including the data content
497 =>         serveResource(request, response, true, fileEncoding);
498
499        }
500
501
502        /**
http-nio-8080-exec-2[1]
```

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
  - Prometheus.io and 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 occuring, 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:8080/badapp/loopfor1min.jsp
```

High CPU, an Example... cont’d.

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.

```
[csutherl@localhost apacheconna-demo]$ ./high_cpu_linux_jstack.sh $(jps | grep Bootstrap)
thread dump # 1
Sleeping...
thread dump # 2
Sleeping...
thread dump # 3
Sleeping...
thread dump # 4
Sleeping...
thread dump # 5
Sleeping...
thread dump # 6
```

High CPU, an Example... cont’d.

The jstack script looks like this:

```
# 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
```

High CPU, an Example... cont'd.

3. After capturing the data, check the CPU usage first to identify large

```
[csutherl@localhost highcpu]$ grep PID -A3 high-cpu.out
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
29717 csutherl  20   0   13.3g 261428  30216 R  93.8   0.8   1:02.60 http-nio-8080-e
29679 csutherl  20   0   13.3g 261428  30216 S   0.0   0.8   0:00.00 java
29680 csutherl  20   0   13.3g 261428  30216 S   0.0   0.8   0:01.21 java
--
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
29717 csutherl  20   0   13.3g 248444  30216 R  99.9   0.8   1:22.86 http-nio-8080-e
29679 csutherl  20   0   13.3g 248444  30216 S   0.0   0.8   0:00.00 java
29680 csutherl  20   0   13.3g 248444  30216 S   0.0   0.8   0:01.21 java
--
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
29717 csutherl  20   0   13.3g 248444  30216 R  99.9   0.8   1:43.11 http-nio-8080-e
29679 csutherl  20   0   13.3g 248444  30216 S   0.0   0.8   0:00.00 java
29680 csutherl  20   0   13.3g 248444  30216 S   0.0   0.8   0:01.21 java
--
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
29717 csutherl  20   0   13.3g 248444  30216 R  99.9   0.8   2:03.36 http-nio-8080-e
29679 csutherl  20   0   13.3g 248444  30216 S   0.0   0.8   0:00.00 java
29680 csutherl  20   0   13.3g 248444  30216 S   0.0   0.8   0:01.21 java
```

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:23
        at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:166)
```

High CPU, an Example... cont’d.

5. Now that you know where the hang is, find it in the code and see why :)

```
[csutherl@localhost build]$ awk 'NR == 116, NR == 120 { print NR, $0 }' work/Catalina/localhost/badapp/org/apache/jsp/loopfor1min_jsp.java
116
117 long finish = System.currentTimeMillis() + 60*1000;
118 while (System.currentTimeMillis() < finish) {
119     // Do nothing...
120 }
```

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.

```
[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 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.

```
[csutherl@localhost build]$ time curl http://localhost:8080/badapp/oome.jsp
<!doctype html><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,A
rial,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-col
or: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;b
ackground-color:#525D76;border:none;}</style></head><body><h1>HTTP Status 500 – Internal Server Error</h1><hr class=
"line" /><p><b>Type</b> Exception Report</p><p><b>Message</b> An exception occurred processing [oome.jsp] at line [7
]</p><p><b>Description</b> The server encountered an unexpected condition that prevented it from fulfilling the requ
est.</p><p><b>Exception</b></p><pre>org.apache.jasper.JasperException: An exception occurred processing [oome.jsp] a
t line [7]

4: ArrayList list = new ArrayList&lt;String&gt;();
5:
6: for (int s = 0; s &lt; 1000000000; 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 :)

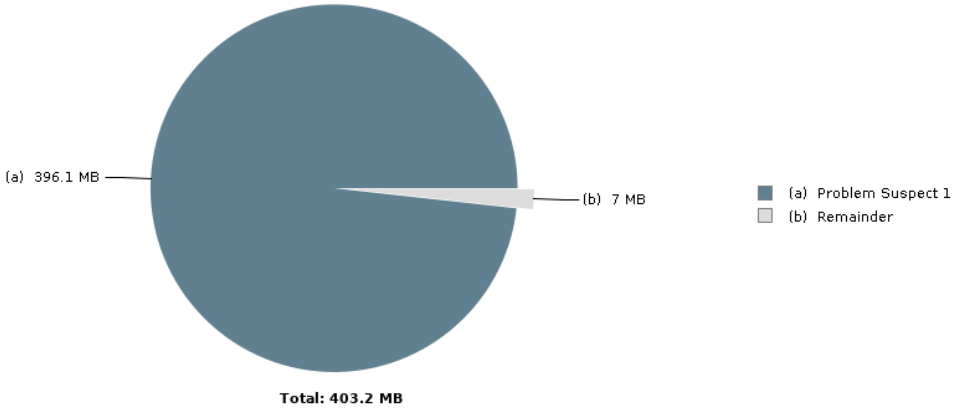
Leak Suspects

Leak Suspects

System Overview

Leaks

Overview



Problem Suspect 1

The thread **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 **"java.lang.Object[]"** loaded by **"<system class loader>"**.

The stacktrace of this Thread is available. See stacktrace.

Keywords
java.lang.Object[]
Details »

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[].

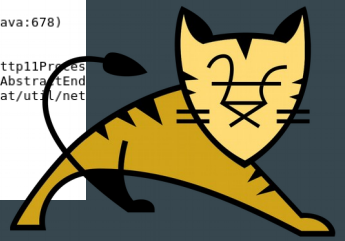
Accumulated Objects by Class in Dominator Tree

Label	Number of Objects	Used Heap Size	Retained Heap Size
java.lang.Integer First 10 of 20,767,597 objects	20,767,597	332,281,552	332,281,552

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!

```
http-nio-8080-exec-1
  at java.lang.OutOfMemoryError.<init>()V (OutOfMemoryError.java:48)
  at java.util.Arrays.copyOf([Ljava/lang/Object;I)[Ljava/lang/Object; (Arrays.java:3689)
  at java.util.ArrayList.grow(I)[Ljava/lang/Object; (ArrayList.java:237)
  at java.util.ArrayList.grow()[Ljava/lang/Object; (ArrayList.java:242)
  at java.util.ArrayList.add(Ljava/lang/Object;[Ljava/lang/Object;I)V (ArrayList.java:485)
  at java.util.ArrayList.add(Ljava/lang/Object;)Z (ArrayList.java:498)
  at org.apache.jsp.oome_jsp._jspService(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;)V (oome_jsp.java:124)
  at org.apache.jasper.runtime.HttpJspBase.service(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;)V (HttpJspBase.java:70)
  at javax.servlet.http.HttpServlet.service(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V (HttpServlet.java:741)
  at org.apache.jasper.servlet.JspServletWrapper.service(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;Z)V (JspServletWrapper.java:476)
  at org.apache.jasper.servlet.JspServlet.serviceJspFile(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;Ljava/lang/String;Z)V (JspServlet.java:385)
  at org.apache.jasper.servlet.JspServlet.service(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServletResponse;)V (JspServlet.java:329)
  at javax.servlet.http.HttpServlet.service(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V (HttpServlet.java:741)
  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V (ApplicationFilterChain.java:231)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V (ApplicationFilterChain.java:166)
  at org.apache.tomcat.websocket.server.WsFilter.doFilter(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;Ljavax/servlet/FilterChain;)V (WsFilter.java:53)
  at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V (ApplicationFilterChain.java:193)
  at org.apache.catalina.core.ApplicationFilterChain.doFilter(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V (ApplicationFilterChain.java:166)
  at org.apache.catalina.core.StandardWrapperValve.invoke(Lorg/apache/catalina/connector/Request;Lorg/apache/catalina/connector/Response;)V (StandardWrapperValve.java:202)
  at org.apache.catalina.core.StandardContextValve.invoke(Lorg/apache/catalina/connector/Request;Lorg/apache/catalina/connector/Response;)V (StandardContextValve.java:96)
  at org.apache.catalina.authenticator.AuthenticatorBase.invoke(Lorg/apache/catalina/connector/Request;Lorg/apache/catalina/connector/Response;)V (AuthenticatorBase.java:526)
  at org.apache.catalina.core.StandardHostValve.invoke(Lorg/apache/catalina/connector/Request;Lorg/apache/catalina/connector/Response;)V (StandardHostValve.java:139)
  at org.apache.catalina.valves.ErrorReportValve.invoke(Lorg/apache/catalina/connector/Request;Lorg/apache/catalina/connector/Response;)V (ErrorReportValve.java:92)
  at org.apache.catalina.valves.AbstractAccessLogValve.invoke(Lorg/apache/catalina/connector/Request;Lorg/apache/catalina/connector/Response;)V (AbstractAccessLogValve.java:678)
  at org.apache.catalina.core.StandardEngineValve.invoke(Lorg/apache/catalina/connector/Request;Lorg/apache/catalina/connector/Response;)V (StandardEngineValve.java:74)
  at org.apache.catalina.connector.CoyoteAdapter.service(Lorg/apache/coyote/Request;Lorg/apache/coyote/Response;)V (CoyoteAdapter.java:343)
  at org.apache.coyote.http11.Http11Processor.service(Lorg/apache/tomcat/util/net/SocketWrapperBase;)Lorg/apache/tomcat/util/net/AbstractEndpoint$Handler$SocketState; (Http11Proces
  at org.apache.coyote.AbstractProcessorLight.process(Lorg/apache/tomcat/util/net/SocketWrapperBase;Lorg/apache/tomcat/util/net/SocketEvent;)Lorg/apache/tomcat/util/net/AbstractEnd
  at org.apache.coyote.AbstractProtocol$ConnectionHandler.process(Lorg/apache/tomcat/util/net/SocketWrapperBase;Lorg/apache/tomcat/util/net/SocketEvent;)Lorg/apache/tomcat/util/net
  at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun()V (NioEndpoint.java:1589)
  at org.apache.tomcat.util.net.SocketProcessorBase.run()V (SocketProcessorBase.java:49)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(Ljava/util/concurrent/ThreadPoolExecutor$Worker;)V (ThreadPoolExecutor.java:1128)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run()V (ThreadPoolExecutor.java:628)
  at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run()V (TaskThread.java:61)
  at java.lang.Thread.run()V (Thread.java:834)
```



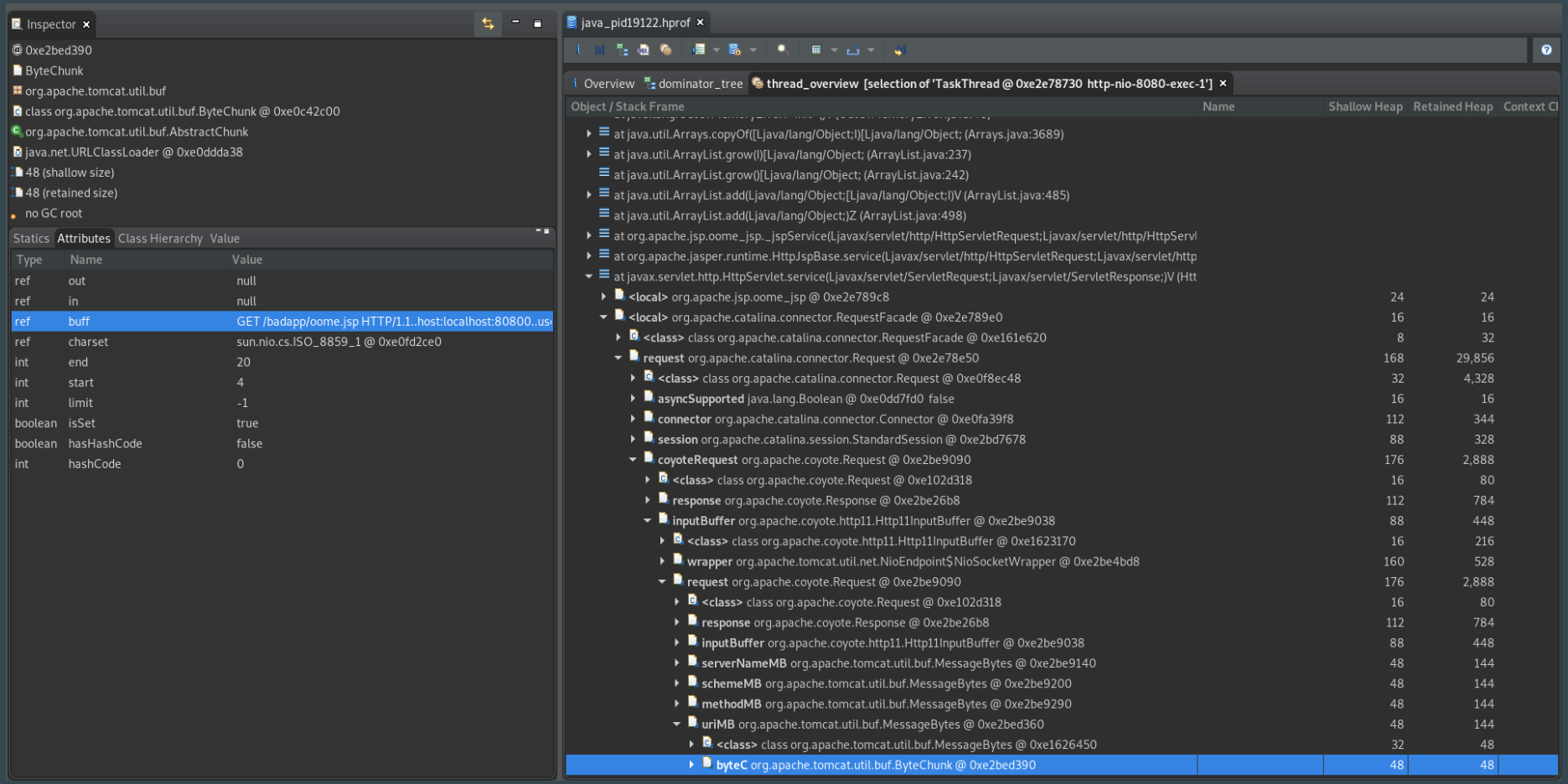
Heap Analysis with Eclipse MAT, cont’d.

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 < 1000000000; 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).


Inspector
@ 0xe2bed390
ByteChunk
org.apache.tomcat.util.buf
class org.apache.tomcat.util.buf.ByteChunk @ 0xe0c42c00
org.apache.tomcat.util.buf.AbstractChunk
java.net.URLClassLoader @ 0xe0ddda38
48 (shallow size)
48 (retained size)
no GC root
Statics Attributes Class Hierarchy Value
Type Name Value
ref out null
ref in null
ref buff GET /badapp/oome.jsp HTTP/1.1..host:localhost:80800..us
ref charset sun.nio.cs.ISO_8859_1 @ 0xe0fd2ce0
int end 20
int start 4
int limit -1
boolean isSet true
boolean hasHashCode false
int hashCode 0
java_pid19122.hprof
Overview dominator_tree thread_overview [selection of 'TaskThread @ 0xe2e78730 http-nio-8080-exec-1']
Object / Stack Frame Name Shallow Heap Retained Heap Context Cl
at java.util.Arrays.copyOf([Ljava/lang/Object;I)[Ljava/lang/Object; (Arrays.java:3689)
at java.util.ArrayList.grow(I)[Ljava/lang/Object; (ArrayList.java:237)
at java.util.ArrayList.grow()[Ljava/lang/Object; (ArrayList.java:242)
at java.util.ArrayList.add(Ljava/lang/Object;[Ljava/lang/Object;I)V (ArrayList.java:485)
at java.util.ArrayList.add(Ljava/lang/Object;)Z (ArrayList.java:498)
at org.apache.jsp.oome_jsp._jspService(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http/HttpServl
at org.apache.jasper.runtime.HttpJspBase.service(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/http
at javax.servlet.http.HttpServlet.service(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V (Htt
<local> org.apache.jsp.oome_jsp @ 0xe2e789c8 24 24
<local> org.apache.catalina.connector.RequestFacade @ 0xe2e789e0 16 16
<class> class org.apache.catalina.connector.RequestFacade @ 0xe161e620 8 32
request org.apache.catalina.connector.Request @ 0xe2e78e50 168 29,856
<class> class org.apache.catalina.connector.Request @ 0xe0f8ec48 32 4,328
asyncSupported java.lang.Boolean @ 0xe0dd7fd0 false 16 16
connector org.apache.catalina.connector.Connector @ 0xe0fa39f8 112 344
session org.apache.catalina.session.StandardSession @ 0xe2bd7678 88 328
coyoteRequest org.apache.coyote.Request @ 0xe2be9090 176 2,888
<class> class org.apache.coyote.Request @ 0xe102d318 16 80
response org.apache.coyote.Response @ 0xe2be26b8 112 784
inputBuffer org.apache.coyote.http11.Http11InputBuffer @ 0xe2be9038 88 448
<class> class org.apache.coyote.http11.Http11InputBuffer @ 0xe1623170 16 216
wrapper org.apache.tomcat.util.net.NioEndpoint$NioSocketWrapper @ 0xe2be4bd8 160 528
request org.apache.coyote.Request @ 0xe2be9090 176 2,888
<class> class org.apache.coyote.Request @ 0xe102d318 16 80
response org.apache.coyote.Response @ 0xe2be26b8 112 784
inputBuffer org.apache.coyote.http11.Http11InputBuffer @ 0xe2be9038 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 @ 0xe2bed360 48 144
<class> class org.apache.tomcat.util.buf.MessageBytes @ 0xe1626450 32 48
byteC 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?

Red Hat

THANK YOU!

Coty Sutherland

github.com/csutherl

linkedin.com/in/cotysutherland

twitter.com/cotysutherland