KnownIssues

FAQ / Known Issues

Permalink to this page: https://cwiki.apache.org/confluence/x/DColBg

Questions

Answers

What are the known issues in any given Tomcat version?

To determine the known issues for any given Tomcat version, you'll need to review the following:

- The currently open bugs and enhancement requests in Bugzilla
- The latest (from svn) change log entries for all newer versions

See chapter Looking for known issues on Tomcat web site.

What are the known issues with the Oracle JRE?

- jps.exe and jvisualvm.exe cannot detect tomcat using jdk1.6.0_23 onwards — Fixed in Java 1.6.0_25.

What are the known issues with the OpenJDK?

- There have been reports that java.util.logging does not work properly in OpenJDK 1.7.0.9 and OpenJDK6 1.6.0_32. The symptom is "`java.lang.ClassNotFoundException: 1catalina.org.apache.juli.FileHandler`" errors when you start Tomcat. See these threads from March 2013 and July 2013. This issue was absent in earlier versions and should be fixed in a later version of those JDKs.

I'm using the Java ImageIO to dynamically serve images and get strange Exceptions from time to time. Is this a bug in Tomcat?

Imagine you have a servlet which dynamically generates images and serves them via javax.imageio.ImageIO. To write the image to the OutputStream, perhaps you are doing something like this:

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
        BufferedImage img = createMyImage(); // makes a BufferedImage

        response.setContentType("image/png");
        try (OutputStream out = response.getOutputStream()) { // try-with-resources
            ImageIO.write(img, "PNG", out);
        } catch (IOException ex) {
            // Client aborted connection
        }
    }
```

Now, although there shouldn't be any Exception logged (because the IOException which occurs when the client aborted the connection is ignored), you see strange Exceptions in Tomcat's log which may belong to other Servlets/JSP (at least with Sun/Oracle JVM on Windows), saying that the response has already been committed, although you didn't write anything to it at that time. For example:

```
13.07.2011 00:13:51 org.apache.catalina.core.StandardWrapperValve invoke
SEVERE: Servlet.service() for servlet [myApp.MyServlet] in context with path [] threw exception
java.lang.IllegalStateException: Cannot create a session after the response has been committed
        at org.apache.catalina.connector.Request.doGetSession(Request.java:2734)
        ...
```

or maybe you use the ISAPI Redirector for IIS on Windows, and get these logs:

```
[Tue Jul 12 06:04:49.812 2011] [4124:2444] [error] ajp_connection_tcp_get_message::jk_ajp_common.c (1296):
wrong message format 0xdaed from 127.0.0.1:8019
```

Is this a bug in Tomcat?

Actually, it's a bug (or at least a strange behavior) in the Java ImageIO. When the ImageIO writes to an OutputStream and gets an IOException during writing, it could happen that some later time, when the ImageWriter is garbage-collected, the flush() method is called on that OutputStream. Tomcat recycles OutputStream objects to save resources, so it could be that when flush() is called from the ImageIO, the particular OutputStream object already belongs to another Response, which can produce the above errors, when the Servlet tries to get a Session for example, or can generally lead to broken responses.

See also here or this Bug report.

So how to resolve the errors?

To resolve this, I'm using an OutputStream decorator class which decorates Tomcat's OutputStream and prevents any flush() calls. Additionally, when close() is called on that Stream, it nulls-out the reference to Tomcat's OutputStream and prevents any other operations:

```
/**
 * A OutputStream which can be used to write Images
 * with the ImageIO in servlets.
 */
public class MyImageIOOutputStream extends OutputStream {

    private OutputStream out;
    private volatile boolean isActive = true;

    public MyImageIOOutputStream(OutputStream out) {
        this.out = out;
    }

    @Override
    public void close() throws IOException {
        if (isActive) {
            isActive = false; // deactivate
            try {
                out.close();
            } finally {
                out = null;
            }
        }
    }

    @Override
    public void flush() throws IOException {
      if(isActive) {
        out.flush();
      }
      // otherwise do nothing (prevent polluting the stream)
    }

    @Override
    public void write(byte[] b, int off, int len) throws IOException {
        if (isActive)
            out.write(b, off, len);
    }

    @Override
    public void write(byte[] b) throws IOException {
        if (isActive)
            out.write(b);
    }

    @Override
    public void write(int b) throws IOException {
        if (isActive)
            out.write(b);
    }
}
```

Now you just have to use this Decorater class instead of using Tomcat's OutputStream directly:

```
    response.setContentType("image/png");
    try (OutputStream out = new MyImageIOOutputStream(response.getOutputStream())) {
        ImageIO.write(img, "PNG", out);
    } catch (IOException ex) {
        // Client aborted connection
    }
```

and the errors should be gone away.

An alternative would be to write the Image contents to a ByteArrayOutputStream, and using its writeTo() method to write the contents to the Servlet's Response. However that would require some additional memory, as the contents have to be buffered.

Are there any other corresponding cases of this bug?

The third party PDF generating software module PD4ML has had a corresponding problem when calling the render() methods in class org.zefer.pd4ml.
PD4ML with response.getOutputStream() as argument. That causes the response stream to be closed from a finalizer() method of a class called PD4Device. When using an Apache/Tomcat connector, this unexpected stream close from the finalizer thread has occationally caused responses to be sent to wrong requestor (request/response mix up). The workarounds described above for ImageIO works perfectly in this case too.

A general way to protect the response output streams from misbehaving web applications is to set the system property org.apache.catalina.connector.
RECYCLE_FACADES=true, since that makes Tomcat create new stream instances for each request (of course at the cost of performance).

PD4ML has fixed this bug in their latest releases, but sites using older versions of the library can still be affected. PD4ML version 3.2.3 definitely has this flaw, but the currently latest version 3.8.0 is fixed. The release notes document gives no clues where in between the problem was fixed, and the vendor was not able to tell either in this bug report.