Troubleshooting and Diagnostics

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

Troubleshooting and Diagnostics techniques.

Table of Contents

Techniques & Reference

- How To: Capture a thread dump
- How To: Capture a heap dump
- How To: Examine a Stacktrace
- How To: Configure Tomcat for debugging
- FAQ: Developing
- FAQ: Memory
- Tomcat Memory Leak Protection
- Notes on using JMX clients

Tools

JMX Clients

- JJConsole: Documentation
- VisualVM: Documentation, Project

JDK tools

- jinfo - Prints JVM process info
- jstack - Prints thread stack traces
- jmap - Dumps heap and shows heap status
- jhat - Heap Analyzer Tool
- jcmd - Multitool intended to replace the above JDK tools

Profilers & Heap Analyzers

- Eclipse Memory Analyzer (MAT)
- YourKit Profiler
- VisualVM Docs

Notes on using JMX clients

When running a JMX client (JConsole, VisualVM) on the same machine as the target JVM process it is possible to connect without pre-configuring a JMX port, using the local connector stub. This method relies on being able to create a protected temporary file, accessible only to a user with administrator privileges. Java processes which are accessible via the local connector will automatically appear in the client.

NB(1) On Windows, this means that the temporary directory must be located on an NTFS formatted disk. See the following link for more details.

NB(2) On Windows, if Tomcat is started using a service wrapper, this will prevent JConsole & VisualVM from using the local JMX connector stub.

Java 5 JConsole and Remote Management FAQ

From Java 6 onward a process does not need to have the management agent enabled when it starts, as the Attach API permits the management agent to be activated on demand.

Common Troubleshooting Scenario

If you *have already looked into Tomcat logs*, there are no error messages, and you just want to find out what is going on, you may try the following

1. Look into Tomcat access log (the log file generated by AccessLogValve).
    a. If your request is not listed there, then it has not been processed by Tomcat. You need to look elsewhere (e.g. at your firewall).
    b. You will see what IP address your client is using, and whether it is using an IPv4 (`127.0.0.1`) or IPv6 address (`0:0:0:0:0:0:0:1`). Modern operating systems can use IPv6 addresses for localhost / local network access, while external network is still using IPv4.
2. Take a thread dump. This way you will find out what Tomcat is actually doing.
    a. If you are troubleshooting some process that takes noticeable time, take **several** (three) thread dumps with some interval between them. This way you will see if there are any changes, any progress.
3. Try debugging.
    a. A good place for a breakpoint is `org.apache.catalina.connector.CoyoteAdapter.service()` method. That is the entry point from Tomcat connectors and into the Servlet engine. At that place your request has already been received and its processing starts.
4. If you did a long-awaited upgrade, jumping over several years worth of Tomcat releases, and something broke, and you have no clue,
    a. Reading Migration guides may help.
    b. It may help to do a binary search (aka bisecting) to locate the version of Tomcat that triggered the change. If your issue is easy to reproduce, it may be pretty fast. Just 7-8 tries may cover a range of 100 versions. Once you know the version and its release date, the following resources are available:
        i. The release announcement.
        See "former announcements" link at the bottom of the front page of the Apache Tomcat site.
        An announcement mail message can also be found in the archives of the "announce@" mailing list.
        ii. The changelog. A release announcement usually has a link to it.
        iii. Archives of the "users@" mailing list. You may look for discussions that happened a month or two after the release.

Troubleshooting unexpected Response state problems

If you encounter problems that manifest themselves as accessing a request or response that is an inconsistent state, the main suspect is **your own web application** (or a library that it uses) keeping a reference to Request or Response objects outside of their life cycle. Examples: BZ 61289, BZ 58457.

The lifetime of the Response object is documented in the Servlet specification. Quoting from section "5.8 Lifetime of the Response Object" of Servlet 4.0 specification:

> "Each response object is valid only within the scope of a servlet's service method, or within the scope of a filter's doFilter method, unless the associated request object has asynchronous processing enabled for the component. If asynchronous processing on the associated request is started, then the response object remains valid until complete method on AsyncContext is called."

In case of asynchronous processing, when an error occurs Tomcat notifies all registered `AsyncListeners` and then calls `complete()` automatically if none of the listeners have called it yet. (Reference: 61768)

Also see sections "2.3.3.4 Thread Safety" and "3.13 Lifetime of the Request Object" of the same specification.

To troubleshoot the issue:

1. Make sure that your Tomcat is configured to discard facades to its internal objects when request processing completes. This makes it easier to spot illegal access when it happens, instead of waiting until side effects of such access become visible. Essentially, it protects Tomcat internals from misbehaving web applications.

    This feature is always on when you are running Tomcat with a Java Security Manager being enabled. Starting with Tomcat 10.0 this feature is enabled by default. It is **disabled** by default in earlier versions of Tomcat. The way this feature is configured differs between versions: it is controlled by an attribute on Connector element or by a system property.

    If you are running Tomcat 9.0 or earlier, do both of the following:

    - Set the following system property in Tomcat configuration:
    `org.apache.catalina.connector.RECYCLE_FACADES=true`

    - Add the following attribute to all Connector elements:
    `discardFacades`="true"

    The Connector attribute was added in Tomcat 10.0.0-M1, 9.0.31, 8.5.51 and 7.0.100. The system property is an older way to configure this feature. In case of a doubt, or if you are switching back and forth between versions while troubleshooting the issue, it is safer to configure both of them.

    This feature is also mentioned on the Security Considerations page in Tomcat documentation. You can also search the archives of the Tomcat users' mailing lists for previous discussions mentioning the RECYCLE_FACADES flag.

Accessing response objects after their lifetime can lead to security issues in your application, such as sending responses to wrong clients, mixing up responses. If you can reproduce the issue and the above diagnostic does not show your own bug, but a bug in Apache Tomcat, if the problem manifests as a security issue, see how to report it.

There are some known examples of broken libraries / APIs:

1. Read about Java ImageIO issue — an issue with javax.imageio.ImageIO API. It may have already been fixed as it is an old issue, but there are no clear records of it.
2. Read about an issue in PD4ML, a library that is used to generate PDF files, — fixed in their version 3.8.0, earlier versions may be affected.

Troubleshooting "Too many open file descriptors"

The code that opens the descriptors can be identified using a tool such as http://file-leak-detector.kohsuke.org/