HowTo FasterStartUp

How do I make Tomcat startup faster?

This section provides several recommendations on how to make your web application and Apache Tomcat as a whole to start up faster.

General

Before we continue to specific tips and tricks, the general advice is that if Tomcat hangs or is not responsive, you have to perform diagnostics. That is to **take several thread dumps** to see what Tomcat is really doing. See Troubleshooting and Diagnostics page for details.

JAR scanning

The Servlet 3.0 specification (chapter 8) introduced support for several "plugability features". Those exist to simplify a structure of a web application and to simplify plugging of additional frameworks. Unfortunately, these features require scanning of JAR and class files, which may take noticeable time. Conformance to the specification requires that the scanning were performed by default, but you can configure your own web application in several ways to avoid it (see below). It is also possible to configure which JARs Tomcat should skip.

For further talk, the features that require scanning are:

Introduced by Servlet 3.0:

- SCI (`javax.servlet.ServletContainerInitializer`)
- Web fragments (`META-INF/web-fragment.xml`)
- Resources of a web application bundled in jar files (`META-INF/resources/*`)
- Annotations that define components of a web application (`@WebServlet` etc.)
- Annotations that define components for 3-rd party libraries initialized by an SCI (arbitrary annotations that are defined in `@HandlesTypes` annotation on a SCI class)

Older features, introduced by earlier specifications:

- TLD scanning, (Discovery of tag libraries. Scans for Tag Library Descriptor files, `META-INF/**/*.tld`).

Among the scans the annotation scanning is the slowest. That is because each class file (except ones in ignored JARs) has to be read and parsed looking for annotations in it.

An example of a container-provided SCI that triggers annotation scanning is the WebSocket API implementation which is included with standard distribution in all versions of Tomcat 8 and with Tomcat 7 starting with 7.0.47. An SCI class declared there triggers scanning for WebSocket endpoints (the classes annotated with `@ServerEndpoint` or implementing `ServerApplicationConfig` interface or extending the abstract `Endpoint` class). If you do not need support for WebSockets, you may remove the WebSocket API and WebSocket implementation JARs from Tomcat (`websocket-api.jar` and `tomcat7-websocket.jar` or `tomcat-websocket.jar`).

A note on TLD scanning: In Tomcat 7 and earlier the TLD scanning happens twice,

- first, at startup time, to discover listeners declared in tld files (done by `TldConfig` class),
- second, by JSP engine when generating java code for a JSP page (done by `TldLocationsCache`).

The second scanning is more noticeable, because it prints a diagnostic message about scanned JARs that contained no TLDs. In Tomcat 8 the TLD scanning happens only once at startup time (in `JasperInitializer`).

Configure your web application

See chapter in Tomcat 7 migration guide.

There are two options that can be specified in your `WEB-INF/web.xml` file:

1. Set `metadata-complete="true"` attribute on the `<web-app>` element. 2. Add an empty `<absolute-ordering />` element.

Setting `metadata-complete="true"` disables scanning your web application and its libraries for classes that use annotations to define components of a web application (Servlets etc.). The `metadata-complete` option is not enough to disable all of annotation scanning. If there is a SCI with a `@HandlesTypes` annotation, Tomcat has to scan your application for classes that use annotations or interfaces specified in that annotation.

The `<absolute-ordering>` element specifies which web fragment JARs (according to the names in their `WEB-INF/web-fragment.xml` files) have to be scanned for SCIs, fragments and annotations. An empty `<absolute-ordering/>` element configures that none are to be scanned.

In Tomcat 7 the `absolute-ordering` option affects discovery both of SCIs provided by web application and ones provided by the container (i.e. by the libraries in `$CATALINA_HOME/lib`). In Tomcat 8 the option affects the web application ones only, while the container-provided SCIs are always discovered, regardless of `absolute-ordering`. In such case the `absolute-ordering` option alone does not prevent scanning for annotations, but the list of JARs to be scanned will be empty, and thus the scanning will complete quickly. The classes in `WEB-INF/classes` are always scanned regardless of `absolute-ordering`.

Scanning for web application resources and TLD scanning are not affected by these options.

Remove unnecessary JARs

Remove any JAR files you do not need. When searching for classes every JAR file needs to be examined to find the needed class. If the jar file is not there - there is nothing to search.

Note that a web application should never have its own copy of Servlet API or Tomcat classes. All those are provided by the container (Tomcat) and should never be present in the web application. If you are using Apache Maven, such dependencies should be configured with `<scope>provided</scope>`. See also a stackoverflow page.

Exclude JARs from scanning

In Tomcat 7 JAR files can be excluded from scanning by listing their names or name patterns in a system property. Those are usually configured in the `conf/catalina.properties` file.

In Tomcat 8 there are several options available. You can use a system property or configure a `<JarScanFilter>` element in the context file of your web application.

Disable WebSocket support

There exists an attribute on `Context` element, `containerSciFilter`. It can be used to disable container-provided features that are plugged into Tomcat via SCI API: WebSocket support (in Tomcat 7 and later), JSP support (in Tomcat 8 and later).

The class names to filter can be detected by looking into `META-INF/services/javax.servlet.ServletContainerInitializer` files in Tomcat JARs. For WebSocket support the name is `org.apache.tomcat.websocket.server.WsSci`, for JSP support the name is `org.apache.jasper.servlet.JasperInitializer`. e.g.:

```
<Context containerSciFilter="WsSci" />
```

The impact of disabling WebSocket support will depend on how many JARs were being scanned for WebSocket annotations and whether any other SCIs trigger annotation scans. Generally, it is the first SCI scan that has the biggest performance impact. The impact of additional scans is minimal.

References: Bug 55855, Tomcat 8 Context documentation

Entropy Source

Tomcat 7+ heavily relies on SecureRandom class to provide random values for its session ids and in other places. Depending on your JRE it can cause delays during startup if entropy source that is used to initialize SecureRandom is short of entropy. You will see warning in the logs when this happens, e.g.:

```
<DATE> org.apache.catalina.util.SessionIdGenerator createSecureRandom
INFO: Creation of SecureRandom instance for session ID generation using [SHA1PRNG] took [5172] milliseconds.
```

There is a way to configure JRE to use a non-blocking entropy source by setting the following system property: `-Djava.security.egd=file:/dev/./urandom`

Note the "`/./`" characters in the value. They are needed to work around known Oracle JRE bug #6202721. See also JDK Enhancement Proposal 123. It is known that implementation of SecureRandom was improved in Java 8 onwards.

Also note that replacing the blocking entropy source (/dev/random) with a non-blocking one actually reduces security because you are getting less-random data. If you have a problem generating entropy on your server (which is common), consider looking into entropy-generating hardware products such as "EntropyKey".

Starting several web applications in parallel

With Tomcat 7.0.23+ you can configure it to start several web applications in parallel. This is disabled by default but can be enabled by setting the `startStopThreads` attribute of a **Host** to a value greater than one.

Other

Memory

Tweak memory parameters - Google is your friend.

Config

Trim the config files as much as possible. XML parsing is not cheap. The less there is to parse - the faster things will go.

Web application

1. Remove any web applications that you do not need. (So remove the all the web applications installed with tomcat) 2. Make sure your code is not doing slow things. (Use a profiler)

CategoryFAQ