
APACHE TOMCAT™
The world's most popular and lightweight open source Java application server deployed in over 50% of businesses, from small start-ups to large-scale production environments.
Tomitribe™
join the tribe
LEARN MORE AND JOIN THE OPEN SOURCE COMMUNITY
tomcat.apache.org
tomee.apache.org
@tomitribe
facebook.com/tomitribe
tomitribe.com

Apache Tomcat

By Alex Soto Bueno and Romain Manni-Bucau

ABOUT APACHE TOMCAT

Apache Tomcat is a pure Java open-source web server that implements the Java Servlet, JavaServer Pages, and Expression Language specifications.

According to the JRebel report, Tomcat is one of the most used web servers in the Java world with more than 50% of the market share.

INSTALLATION

DOWNLOAD

The quickest way to run Tomcat is to download and run a compiled version. Go to http://tomcat.apache.org/ and in the Download section choose the Tomcat version that fits your requirements and package file depending on your OS.

TOMCAT VERSION	SPECIFICATION
Tomcat 8.0.x	Servlet 3.1 / JSP 2.3 / EL 3.0 / WebSocket 1.1 / Java 7 and later
Tomcat 7.0.x	Servlet 3.0 / JSP 2.2 / EL 2.2 / WebSocket 1.1 / Java 6 and later (WebSocket requires Java 7)
Tomcat 6.0.x	Servlet 2.5 / JSP 2.1 / EL 2.1 / Java 5 and later

To run Tomcat, you have to first install a Java Runtime Environment (JRE). Make sure to install the right version depending on the Tomcat version you want to run (see table above).

```
$ wget http://repo.maven.apache.org/maven2/org/apache/
tomcat/tomcat/8.0.24/tomcat-8.0.24.tar.gz
$ tar -zxvf apache-tomcat-8.0.24.tar.gz
$ cd apache-tomcat-8.0.24
```

The root directory is known as CATALINA_HOME. Optionally, Tomcat may be configured for multiple instances by defining CATALINA_BASE for each instance. For a single installation, CATALINA_BASE is the same as CATALINA_HOME.

RUNNING

The main script to start Tomcat is ${CATALINA_HOME}/bin/catalina.sh and the most used start-up commands are:

COMMAND	DESCRIPTION
debug [-security]	Starts in a debugger
start [-security]	Starts in a separate window (or in the background)
run [-security]	Starts in the current window (in the foreground)
stop [-force]	Stops server waiting up to 5 seconds -force option kills the process if not stopped after 5 seconds
jpda start	Starts under JPDA debugger – Environment variable JPDA_ADDRESS defines the debug address (often just a port) and JPDA_SUSPEND to suspend execution immediately after start-up

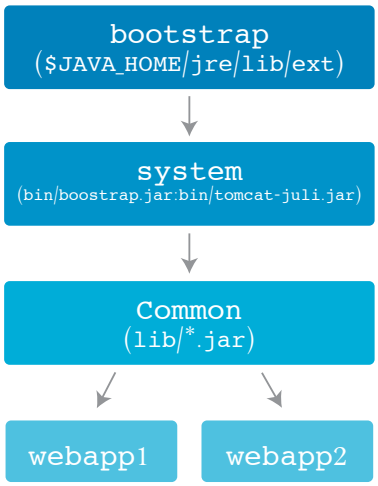
When Tomcat is started in the foreground, it can be stopped by pressing Ctrl+C.

There is a Windows package distribution that installs Tomcat as a Service on Windows operating systems. Most Linux distributions have their own packaging.

DIRECTORY LAYOUT

DIRECTORY	DESCRIPTION
/bin	Stores executable files for Windows/*nix systems to start and stop server.
/conf	Stores configuration files.
/lib	Stores libraries to share between all applications. By default, only Tomcat libraries are in this directory.
/logs	Stores Tomcat log files.
/temp	Stores temporary files created using the Java File API.
/webapps	Deploys .war files or exploded web applications.
/work	Stores intermediate files (such as compiled JSP files) during its work.

CLASSLOADING



In the case of web applications, and according to Servlet specifications, when a classloader is asked to load a class or resource, it will first try in its own classloader, and then in its parent classloader(s).

EMBEDDING

An embedded Tomcat instance can be started within the same JVM of the running application. Some dependencies must be added to a class path.

Gradle file with dependencies required:

```
dependencies {
 //Minimal dependencies
 compile 'org.apache.tomcat.embed:tomcat-embed-core:8.0.9'
 compile 'org.apache.tomcat.embed:tomcat-embed-logging-
juli:8.0.9'
 //Optional dependency for JSP
 compile 'org.apache.tomcat.embed:tomcat-embed-
jasper:8.0.9'
 compile 'org.eclipse.jdt.core.compiler:ecj:4.4'
}
```

Then you can instantiate a new Tomcat instance as any other Java class and call Tomcat operations such as registering Servlet, setting a webapp directory, or configuring resources programmatically.

```
//create a Tomcat instance to 8080 port
Tomcat tomcat = new Tomcat();
tomcat.setPort(8080);

//adds a new context pointing to current directory as base
dir.
Context ctx = tomcat.addContext("/", new File(".").
getAbsolutePath());

//registers a servlet with name hello
Tomcat.addServlet(ctx, "hello", new HttpServlet() {
            protected void service(HttpServletRequest req,
HttpServletResponse resp) throws Exception {
              Writer w = resp.getWriter();
              w.write("Hello World");
              w.flush();
            }
  });

//adds a new mapping so any URL executes hello servlet
ctx.addServletMapping("/*", "hello");

//starts Tomcat instance
tomcat.start();

//waits current thread indefinitely
tomcat.getServer().await();
```

CONFIGURATION

Apache Tomcat's main configuration is composed of four files: server.xml, context.xml, web.xml, and logging.properties.

SERVER.XML

server.xml is located in ${CATALINA_BASE}/conf and represents the server itself. Here is an example:

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">

 <Listener className="xxxxx" />
 <GlobalNamingResources>
   <Resource name="UserDatabase" auth="Container"
             type="org.apache.catalina.UserDatabase"
             description="User database" factory="org.
apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
 </GlobalNamingResources>

 <Service name="Catalina">
   <Executor name="tomcatThreadPool" namePrefix="catalina-
exec-"
maxThreads="150" minSpareThreads="4"/>

   <Connector port="8080" protocol="HTTP/1.1"
              connectionTimeout="20000"
              redirectPort="8443" />

   <Engine name="Catalina" defaultHost="localhost">
     <Realm className="..." />
     <Host name="localhost"  appBase="webapps"
           unpackWARs="true" autoDeploy="true">
       <Valve className="..." />
     </Host>
   </Engine>
 </Service>
</Server>
```

The XML file represents almost a 1:1 layout of the server itself, which conforms with Tomcat's hierarchical design.

Tomcat's Digester processes the XML files and allows for instances of any Java type to be added to the XML and configured in a very Spring-like fashion. Each node can get an attribute className to specify which implementation you want to use, as well as a set of attributes which will be set on the created instance.

These are the most important tags to know to use Tomcat:

NAME	ROLE
Server	The server (aggregate instance) and admin configuration (mainly shutdown socket definition) and hierarchy
Listener	Tomcat internal events listener (see org.apache.catalina.LifecycleListener)
GlobalNamingResources	Defines a set of container-wide resources (as opposed to application ones)
Resource	Defines a JNDI resource
Service	A set of connectors and an engine
Connector	Defines a protocol to use and its configuration (most common ones are HTTP and AJP)
Engine	The "processor" of an incoming request from a connector (a pipeline starting from the host, going through authentication if needed, the webapp and finally the servlet)

NAME	ROLE
Executor	The thread pool associated with a Service to handle the request
Realm	Security repository (login/password/roles), 0 or 1 can be linked to Host, Engine, and/or Context
Host	Virtual host representation ("domain"). It is a container of Contexts (explicit or implicit using appBase).
Context	A web application
Valve	An element of the request pipeline (can be added on Engine, Host, or Context)

CONTEXT.XML

context.xml is a configuration file for a web application. You can also use a Context element under the Host tag in server.xml, but it is recommended (if necessary) that you provide them either in the web application in META-INF/context.xml or (if the configuration is local to the Tomcat instance) in ${CATALINA_BASE}/conf/<engine name>/<hostname>/<warname>.xml.

This last option overrides the META-INF file if both exist. This is convenient for overriding a packaged version, for instance overriding a DataSource.

A shared web application configuration (across all applications) is done in ${CATALINA_BASE}/conf/context.xml (global) and ${CATALINA_BASE}/conf/<engine name>/<hostname>/context.xml.default (specific to a host).

The following are the main children tags of Context:

NAME	ROLE
InstanceListener	Internal events listener specific to filters/servlets
Listener	Tomcat internal events listener (see org.apache.catalina.LifecycleListener) for "Context scoped"
Loader	Defines the classloader to use for the webapp
Manager	Defines the session configuration
Parameter	Defines a context parameter
Resources (PreResources, JarResources, PostResources)	Where resources (JS, CSS, HTML, ..., .class) are taken from
ResourceLink	Link to a global JNDI entry
Valve	Same as in server.xml
WatchedResource	If the defined resource changes, Context will be reloaded (undeploy/deploy)
WrapperLifecycle	LifecycleListener for each wrapper (servlet)
WrapperListener	ContainerListener for each wrapper
JarScanner	The application scanner implementation (which classloader to scan and how to get JAR paths)
JarScanFilter	Which paths should be scanned/ignored, should tag libraries (TLDs) be scanned
Environment	Named values that will be made visible to the web application as environment entry resources

WEB.XML

This section does not refer to WEB-INF/web.xml, which is a standard deployment descriptor of the Servlet specification, but about ${CATALINA_BASE}/conf/web.xml.

This is a standard web.xml where mime types, default servlets/filters, default welcome files, and default session timeouts are defined. This configuration is inherited by all web applications.

Here are the default servlets and some examples of why you might need to update them:

SERVLET	GOAL
Default	Serves static resources. Encoding, is listing folder allowed are configurable.
JSP	Serves (and compiles if needed) JSP. Compiling information, should JSP be watched for updates (development), and pooling are configurable.
SSL (exists as filter too)	Supports server side includes on HTML pages; disabled by default.
CGI	Can execute an external binary to acquire content; disabled by default.

LOGGING

Tomcat uses an enhanced version of the Java Util Logging (JUL) API built into the JVM. As a container, Tomcat needs to support logging configuration at the container level for itself, but it also needs to support application specific configurations.

LOGGING.PROPERTIES

Tomcat uses a custom LogManager to be able to configure the logging more finely: org.apache.juli.ClassLoaderLogManager

The default configuration is in ${CATALINA_BASE}/conf/logging.properties. Each web application can embed its own configuration by defining a logging.properties file under WEB-INF/classes directory of the webapp.

CONFIGURATION OVERVIEW

```
my.logger.name.level = FINEST # a java.util.logging.Level
my.logger.name.handlers = myHandler1, myHandler2,...
my.logger.name.useParentHandlers = true # false by default
```

PREFIXES

You can prefix a logger name with a String starting with a digit and ending with a dot (for instance 1prefix.my.logger.name). This is useful to simultaneously configure the same handler (a file, for instance) with different settings.

DYNAMIC VALUES

You can use place holders like ${xxx}. They get replaced automatically at runtime with the value of the Java system property with the key xxx.

ROOT LOGGER

Root logger is configured using an empty name:

.handlers = ...

HANDLER

Tomcat provides several additional handlers like org.apache.juli.FileHandler, which supports log buffering; AsyncFileHandler, which is asynchronous; and some formatters like JdkLoggerFormatter, which uses the log4j equivalent format %r %-15.15c{2} %-1.1p %m %n.

SERVLETCONTEXT LOGGING

ServletContext.log(...) output is configurable using property org.apache.catalina.core.ContainerBase.[${engine}].[${host}].[${context}].

REPLACE JUL

You can use log4j for Tomcat itself. To do this, you need to add log4j.jar, tomcat-juli-adapters.jar, and log4j.properties in ${CATALINA_HOME}/lib and replace tomcat-juli.jar from the bin/ directory with tomcat-juli.jar from the extras modules (See Tomcat download page). Don't forget to remove the conf/logging.properties file so that JUL does not create empty files.

CLUSTERING

Tomcat supports clustering (session distribution and deployments) out of the box. To activate it, add a Cluster tag in your Host or Engine.

The cluster tag supports these configurations:

TAG	ROLE
Cluster	Defines the cluster class and configuration.
Manager	Session replication strategy (by default it uses DeltaManager to send delta to nodes, synchronously or not).
Channel	How cluster nodes are connected together (backed by Apache Tribes). Channel has sub tags like Membership (discovery of nodes), Sender (send message on the cluster), Receiver (how cluster messages are received) and Interceptor (listen for/send cluster messages).
Valve	Creates and replicates files when needed within the cluster (end of request by default).
Deployer	Allows deploy/undeploy actions across the entire cluster applications.
ClusterListener	Observes cluster messages.

Here is the most basic example:

```
<Cluster className="org.apache.catalina.ha.tcp.
SimpleTcpCluster"/>
```

NOTE: Don't forget your session attributes need to be Serializable. This is also a requirement if you want the session to survive Tomcat cycles (a.k.a. HTTP session serialization).

IDES

Apache Tomcat is supported by major IDE vendors.

ECLIPSE

To register Tomcat on Eclipse WTP (Web Tools Platform).

- Open **Window -> Preferences -> Server -> Installed Runtimes**
- Click on **Add** and in **New Server Runtime**, select **Apache -> Apache Tomcat v8.0**
- Click **Next** and fill in your Tomcat installation directory

INTELLIJ IDEA

To register Tomcat in IntelliJ IDEA Ultimate Edition (communtiy edition is not supported):

- Open **File -> Settings -> (IDE Settings) Application Servers**
- Click on the **+** symbol and select **Tomcat Server**
- In the dialog box fill **Tomcat Home** with your Tomcat installation directory

NETBEANS

Apache Tomcat comes pre-bundled with the Java EE distribution of Netbeans. Registering your own installation of Tomcat can be done as well.

- Open **Windows -> Services**
- Right-Click on **Servers -> Add Server**
- In the dialog box choose **Apache Tomcat**
- Click **Next** then fill **Server Location** with your Tomcat installation directory

I/O

Following today's needs, Tomcat proposes several I/O solutions.

CONNECTORS

As explained in the Configuration section, Tomcat handles I/O thanks to connectors. Most of them share a common configuration, the full details of which are available at http://tomcat.apache.org/tomcat-8.0-doc/config/http.html.

NIO/NIO2

The NIO and NIO2 connectors allow you to use Java NIO to handle incoming requests. With the release of Tomcat 8, they are now the default connectors to handle HTTP requests.

Note that to use NIO2 you need to specify the class org.apache.coyote.http11.Http11Nio2Protocol on your Connector tag in the class-name attribute. Keep in mind it is still a "beta" connector, but it can bring your application some significant speed enhancements.

BIO

The BIO connector allows you to use the old blocking I/O. It was the default connector to handle HTTP requests for prior Tomcat 8 versions.

The main difference between NIO and BIO is that with the BIO connector you generally need more threads to handle requests concurrently, and they don't guarantee better performances. These connectors impose you to get one thread by connection!

For Tomcat 9+ versions, the BIO connector has been completely removed.

AJP

AJP is a protocol like HTTP. Its main goal is to get higher performances. It is a binary version of HTTP, and also a connected protocol as opposed to HTTP. It can be used with httpd (a.k.a Apache Web Server) and most of the time with mod_jk or mod_proxy_ajp httpd modules. If you are using SSL or have a lot of static resources, this is the fastest connector!

COMETD

CometD is the old way to handle asynchronous I/O (i.e. let the container call you when data are available). This only works with APR or NIO connectors.

To use CometD, you first need to implement it within a servlet org.apache.catalina.comet.CometProcessor (eliminating portability) and then handle events in event(CometEvent):

```
public class MyCometDServlet extends HttpServlet
implements CometProcessor {
 @Override
  public void event(final CometEvent event)
       throws IOException, ServletException {

    final HttpServletRequest request = event.
getHttpServletRequest();
    final HttpServletResponse response = event.
getHttpServletResponse();

    if (event.getEventType() == CometEvent.EventType.BEGIN)
{     //keep the response to bulk send data to everybody
    } else
      if (event.getEventType() == CometEvent.EventType.
ERROR || event.getEventType() == CometEvent.EventType.END)
{
      // state cleanup if needed
         event.close();
         // we are done
    } else
    if (event.getEventType() == CometEvent.EventType.READ)
{
           final InputStream is = request.getInputStream();
           do {
              // read is
           } while (is.available() > 0);
       }
   }
```

WEBSOCKET

WebSockets (RFC 6455) have been supported by Tomcat since version 7, but only versions 7.0.43 and later implement the specification (a.k.a JSR-356). Before (version 7.0.27 and earlier) Tomcat was using a proprietary API and implementation didn't scale as much as today -- so ensure you are using an up-to-date version.

The Tomcat WebSocket implementation needs at least Java 7. If you are running on Java 7, it is activated by default and you just need to deploy your code inside Tomcat:

```
@ServerEndpoint("/tchat")
public class TchatEndpoint {

  @OnMessage
  public void onMessage(Session session, String msg) {
    try {
      session.getBasicRemote().sendText(msg);
    } catch (IOException e) { ... }
  }
}
```

WebSocket doesn't require you to use an APR or NIO connector like CometD does, but it is highly recommended, as you will likely have a high number of connections.

JNDI

Tomcat provides a JNDI InitialContext implementation instance for each web application. As mentioned in the Configuration section, JNDI resources can be registered in conf/server.xml, conf/web.xml, or any context.xml file. But Tomcat also follows the Java EE standard for /WEB-INF/web.xml file to reference/define resources.

In order to use a JNDI resource, specify the JNDI name defined in the web application's deployment descriptor:

```
context.xml
<context ...>
  <!-- Definition of DataSource in context.xml -->
  <Resource name="jdbc/employeeDB" type="javax.sql.
DataSource" global="jdbc/employeeDB" .../>
</context>
web.xml
<web-app>
  <!-- Declaring a reference to DataSource in web.xml -->
  <resource-ref>
    <description>Employee Datasource</description>
    <res-ref-name>jdbc/employeeDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
  <env-entry>
    <env-entry-name>message</env-entry-name>
    <env-entry-value>Hello World</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
</webapp>
```

Java Code

```
Context initCtx = new InitialContext();
DataSource ds = (DataSource) initCtx.lookup("java:comp/env/
jdbc/EmployeeDB");
String message = (String)initCtx.lookup("java:comp/env/
message");
```

TOMCAT LISTENERS

A Listener element defines a component that performs actions when specific events occur, usually Tomcat starting or Tomcat stopping.

Listeners are registered inside conf/server.xml inside Server, Engine, Host, and Context tags.

Each Listener must implement org.apache.catalina.LifecycleListener interface.

CUSTOM LISTENER

```
public class PrintLifecycleStatesListener implements
LifecycleListener {
  @Override
  public void lifecycleEvent(LifecycleEvent e) {
    System.out.println(e.getLifecycle().getStateName());
  }
}
```

The JAR file containing this class is stored at $CATALINA_BASE/lib directory.

Add the next line at conf/server.xml.

```
<Server ...>
  <Listener className="org.superbiz.
PrintLifecycleStatesListener"/>
</Server>
```

SHIPPED LISTENERS

LISTENER	DESCRIPTION
org.apache.catalina.core.AprLifecycleListener	Checks for the presence of the APR/native library and loads the library if it is present.
org.apache.catalina.mbeans.GlobalResourcesLifecycleListener	Initializes the Global JNDI resources defined in server .xml as part of the Global Resources element.
org.apache.catalina.core.JreMemoryLeakPreventionListener	Prevents memory leaks by providing workarounds, for instances where the JRE uses the context classloader to load a singleton.
org.apache.catalina.security.SecurityListener	Performs a number of security checks when Tomcat starts and prevents Tomcat from starting if they fail; not enabled by default.
org.apache.catalina.core.ThreadLocalLeakPreventionListener	Triggers the renewal of threads in Executor pools when a Context is being stopped to avoid ThreadLocal-related memory leaks.
org.apache.catalina.startup.UserConfig	Maps a request URI starting with a tilde character ("~") and a username to a directory in that user's home directory on the server; not enabled by default.
org.apache.catalina.mbeans.JmxRemoteLifecycleListener	Fixes the ports used by the JMX/RMI Server; not enabled by default.

DEPLOYING

After installing and configuring Tomcat, you will need to deploy web applications.

DROP-IN WAR

Deploying an application to Tomcat can be as simple as dropping a WAR file inside the *$CATALINA_BASE/webapps* directory; this will deploy the application.

HOT DEPLOYMENT

To deploy a new application, you must restart the server in order for the application to function. To fix this problem, Tomcat provides a hot deployment option, which deploys an application without the need to restart.

To enable hot deployment, the autoDeploy attribute from host tag in server.xml must be set to true. See the configuration section for an example.

TOMCAT MANAGER

WEB APPLICATION

Tomcat also comes with a web application called Manager, which allows for deploying an application from the web console. To use the Manager, you need to add a username and password to conf/tomcat-users.xml with role manager-gui.

```
<tomcat-users>
  <role rolename="manager-gui"/>
  <user username="tomcat" password="s3cret"
roles="manager-gui"/>
</tomcat-users>
```

Next, access **http://<host>:<port>/manager** to access the web console.

URI

The Tomcat Manager also provides URI commands to upload applications. They follow this schema:

```
http://{host}:{port}/manager/text/{command}?{parameters}
```

Some examples of deploying an application using URI commands:

```
http://{host}:{port}/manager/text/deploy?path=/foo
```

The above uploads the WAR file with path /foo to the remote server. The WAR is provided as the body of the HTTP PUT.

```
http://{host}:{port}/manager/text/deploy?path=/foo&war=file:/
path/to/foo.war
```

This deploys applications stored in the server directory **path/to/foo.war**.

ARQUILLIAN

Arquillian is an integration and functional testing platform that can be used for Java middleware testing. The main goal of Arquillian is to make tests that require a server to be as simple as writing unit tests.

You can use Arquillian Tomcat adapters to write integration/functional tests.

Arquillian Tomcat 8 adapter release is coming soon.

Maven coordinates can be found at:

http://arquillian.org/modules/arquillian-tomcat-embedded-7-container-adapter/

http://arquillian.org/modules/arquillian-tomcat-managed-7-container-adapter/

http://arquillian.org/modules/arquillian-tomcat-remote-7-container-adapter/

http://arquillian.org/modules/arquillian-tomcat-embedded-8-container-adapter/

When using the embedded adapter, Tomcat embedded dependencies are required as well.

When using the remote adapter, the Tomcat instance has to expose a remote JMX MbeanConnection.

```
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.jmxremote.
port=8089"
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.jmxremote.
ssl=false"
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.jmxremote.
authenticate=false"
```

MAVEN

Maven is still one of the most used build tools and therefore Tomcat integration is available.

This integration is mainly (in addition to deploying artifacts on central) a Maven plugin.

Today, there are two plugins, tomcat6-maven-plugin for Tomcat 6 and tomcat7-maven-plugin for Tomcat 7; tomcat8-maven-plugin is coming soon.

Their usage is more or less the same:

```
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <configuration>
    <url>http://localhost:8080/manager</url>
  </configuration>
</plugin>
```

Here are the main configurations of the Maven plugin:

PARAMETER	GOAL
url	Tomcat Manager URL (to deploy/undeploy/ start/stop/reload)
server	Use settings.xml server for authentication on the Tomcat Manager application
charset	Encoding to use to communicate with the Tomcat Manager application
Username/ password	Credential of the manager application

Once these configurations are set up, the available goals are as follows:

GOAL	DESCRIPTION
deploy/ undeploy/ redeploy	(Un)deploys a WAR
Exec-war / standalone-war	Create a runnable JAR or WAR running Tomcat and your application (java -jar mytomcatapp.jar)
run	Start Tomcat embedded with current project deployed as dynamic project
run-war	Same as run, using a WAR instead of resources
shutdown	Stop started servers (useful if using pre-integration-test phase to start Tomcat and post-integration-test to stop it)

Note: Often goals will exist with the "-only" suffix. For example, the deploy-only goal will not fork a Maven package lifecycle.

Each goal has further configurations, for example the "context" path within the "deploy" application goal.

You can find more details at http://tomcat.apache.org/maven-plugin-trunk/tomcat7-maven-plugin/plugin-info.html.

JAVA EE AND BEYOND

Tomcat is a Servlet container following Servlet/JSP/EL specification. However, if you are required to use any other Java EE specifications like CDI, JPA, EJB, JAX-RS, or Bean Validation, you can integrate them yourself (assuming you understand the perils of doing it by yourself), or you can simply use Apache TomEE. Apache TomEE (pronounced "Tommy") is the Java Enterprise Edition of Apache Tomcat (Tomcat + Java EE = TomEE) and it is certified as Java EE 6 Web Profile. It maintains the lightness and simplicity of Tomcat but with the full power of Java EE.

ABOUT THE AUTHORS

Alex Soto Bueno is a Java EE software architect specializing in enterprise technologies, test automation and continuous delivery on the Java platform. He is heavily involved in the open source world, leading NoSQLUnit and Asciidoctor projects and contributing to Arquillian and Tomitribe. He is an international speaker, having presented at conferences like Devoxx, JavaOne, or JavaLand. Alex is the author of Arquillian in Action and also the curator of lordofthejars.com blog.

Romain Manni-Bucau is a contributor of the Apache TomEE project since July 2010 and a Senior Software Engineer at Tomitribe. In his plethora of Apache projects, he's involved in OpenEJB, OpenWebBeans, Geronimo, CFX, BVal and DeltaSpike. Romain is a founding member of the Sirona and BatchEE project and brought JCache implementation to the Apache Commons JCS project. He regularly speaks at JUG and conferences to spread the word about all the Apache goodness. Since Java EE 6 he is convinced that REST architectures and design are the future of Java EE and not only for web technologies. Romain blogs at http://rmannibucau.wordpress.com.

RECOMMENDED BOOK

Tomcat: The Definitive Guide is the best available book-length introduction to the world's most popular open-source implementation of Java Servlet and JSP server technologies. Includes installation, configuration, performance tuning, debugging, integrations, security, clustering, and more.

BUY NOW

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795

VERSION 1.0 $7.95