**Module 11**

**Tomcat 7 new features, 5.5, 6.0 differences**

Each version of Tomcat is supported for any stable Java release that meets the requirements of the final column in the table above.

Tomcat should also work on any Java early access build that meets the requirements of the final column in the table above. For example, users were successfully running Tomcat 8 on Java 8 many months before the first stable Java 8 release. However, users of early access builds should be aware of the following:

* It is not unusual for the initial early access builds to contain bugs that can cause problems for web applications running on Tomcat.
* If the new Java version introduces new language features then the default JSP compiler may not support them immediately. Switching the JSP compiler to javac may enable these new language features to be used in JSPs.
* If you do discover an problem using a Java early access build, please [ask for help](http://tomcat.apache.org/findhelp.html). The Tomcat user's mailing list is probably the best place to start.

The releases are described in more detail below to help you determine which one is right for you. More details about each release can be found in the associated release notes.

Please note that although we offer downloads and documentation of older releases, such as Apache Tomcat 6.x, we strongly encourage users to use the latest stable version of Apache Tomcat whenever possible. We recognize that upgrading across major versions may not be a trivial task, and some support is still offered on the mailing list for users of old versions. However, because of the community-driven support approach, the older your version the less people would be interested or able to support you.

**Alpha / Beta / Stable**

Almost every Tomcat release is initially released as an Alpha release. After a week or so of testing a vote is held to gather views on the stability of the release. If a major issue is identified during testing, then the vote may not take place and the release will remain as an Alpha release.

The download pages will always show the latest stable release and any newer Alpha or Beta release if one exists. Alpha and beta releases are always clearly marked on the download pages.

Stability is a subjective judgment and you should always read carefully the release notes for any version you intend to make use of. If you are an early adopter of a release, we would love to hear your opinion about its stability as part of the vote: it takes place on the development mailing list.

**Alpha** releases may contain large amounts of untested/missing functionality required by the specification and/or significant bugs and are not expected to run stably for any length of time.

**Beta** releases may contain some untested functionality and/or a number of relatively minor bugs. Beta releases are not expected to run stably.

**Stable** releases may contain a small number of relatively minor bugs. Stable releases are intended for production use and are expected to run stably for extended periods of time.

**Apache Tomcat 9.x**

**Apache Tomcat 9.x** is the current focus of development, It builds on Tomcat 8.0.x and will implement the **Servlet 4.0, JSP 2.4?, EL 3.1**? and **Web Socket 1.2**? specifications TBD (9.0.x is work in progress)

**Apache Tomcat 8.x**

Apache Tomcat 8.x builds on Tomcat 7.0.x and implements the **Servlet 3.1, JSP 2.3, EL 3.0** and **Web Socket 1.1** specifications. In addition to that, it includes the following significant improvements:

* A single, common resources implementation to replace the multiple resource extension features provided in earlier versions.

**Apache Tomcat 7.x**

Apache Tomcat 7.x builds upon the improvements made in Tomcat 6.0.x and implements the **Servlet 3.0, JSP 2.2, EL 2.2** and **Web Socket 1.1** specifications. In addition to that, it includes the following improvements:

* Web application memory leak detection and prevention
* Improved security for the Manager and Host Manager applications
* Generic CSRF protection
* Support for including external content directly in a web application
* Refactoring (connectors, lifecycle) and lots of internal code clean-up

**Apache Tomcat 6.x**

Apache Tomcat 6.x builds upon the improvements made in Tomcat 5.5.x and implements the Servlet 2.5 and JSP 2.1 specifications. In addition to that, it includes the following improvements:

* Memory usage optimizations
* Advanced IO capabilities
* Refactored clustering

**Apache Tomcat 5.x**

Apache Tomcat 5.x is available for download from the archives. Apache Tomcat 5.5.x supports the same Servlet and JSP Specification versions as Apache Tomcat 5.0.x. There are significant changes in many areas under the hood, resulting in improved performance, stability, and total cost of ownership. Please refer to the Apache Tomcat 5.5 Changelog for details.

Apache Tomcat 5.0.x improves on Apache Tomcat 4.1 in many ways, including:

* Performance optimizations and reduced garbage collection
* Refactored application deployer, with an optional standalone deployer allowing validation and compilation of a web application before putting it in production
* Complete server monitoring using JMX and the manager web application
* Scalability and reliability enhancements
* Improved Taglibs handling, including advanced pooling and tag plugins
* Improved platform integration, with native Windows and Unix wrappers
* Embedding using JMX
* Enhanced Security Manager support
* Integrated session clustering

**What’s New in Tomcat 7**

**Servlet 3.0**

The most prominent change is the support of Servlet 3.0 specification which supports such great features as

* asynchronous servlets – not used by default
* web-fragment.xml –  you can specify parts of the configuration in your libraries which you plan to reuse

The effective deployment descriptor (web.xml) now is the sum total of all the fragment configurations contained in the application’s libraries. This means that you need to be more careful when using 3rd party libraries because they can contain some configuration portion that will prevent your application from working like you expected. But to my understanding you can fixate the effective web.xml used and prevent tomcat from scanning all the libraries for configuration fragments. Now deployment descriptors support programmatic configuration i.e. you can decide how to configure an application at startup by considering the environment the server is in and other variables.

Another interesting thing to note is that before the Servlet 3.0 specification implementation tomcat used two methods of session tracking: cookies and rewriting the URL with the JSESSIONID parameter. The latter in tomcat was mandatory meaning you could not easily switch it off. I actually tried some workarounds, but they produced some side-effects. If anyone knows a proper method to avoid ever seeing the JSESSIONID parameter in Tomcat 6 please let me know. In Tomcat 7 the URL rewriting method is no longer mandatory and a new session tracking method was added based on SSL session. It is more secure, but I believe it needs two-way SSL configured. Don’t quote me on this, because I might be wrong about the two-way SSL, will have to check it. While we’re at it I can mention that now there’s the ability to change the session cookie’s default name, domain and path.

*<session-config>  
   <tracking-mode>COOKIE</tracking-mode>  
</session-config>*

**JSP 2.2**

Not much to talk about other than saying that Tomcat 7 supports it. And maybe worth mentioning that you can now specify the defaults for the content type you’re returning and the buffer size. Also there’s a nice configuration option  added called error-on-undeclared-namespace which quoting the documentation does the following:

The default behavior when a tag with unknown namespace is used in a JSP page (regular syntax) is to silently ignore it.  If set to true, then an error must be raised during the translation time when an undeclared tag is used in a JSP page.  Disabled (false) by default.

And here’s an example of how you would use this option

*<jsp-config>  
    <jsp-property-group>  
        <url-pattern>\*.jsp</url-pattern>  
        <error-on-undeclared-namespace>true</error-on-undeclared-namespace>  
    </jsp-property-group>  
</jsp-config>*

Also note that Tomcat 7 implements EL 2.2 and now is much stricter about the EL specifications, but according to Mark it’s unlikely to affect many users unless of course you’re doing some kind of hack with it.

**Memory leak protection**

This is a nice new feature in Tomcat 7 which has been backported to Tomcat 6 and if you’re using version 6.0.24 or later, you can enjoy the benefits. It’s implemented as a listener which tries to detect and fix the possible memory leaks whenever it can. These are the leak cases that it tries to prevent:

* JDBC drivers – they should be de-registered upon application stop
* ThreadLocals – if you create it when processing the request, you must set it to null before completing the response
* Threads – if a webapp starts a thread (e.g. scheduler) it must stop it, Tomcat doesn’t do that for you because it might crash the JVM, but it does report the problem in the log files
* ResouceBundle – Tomcat automatically fixes this one, but if you want some details turn on debug logging

**Alias support**

This is said to be a much requested feature which allows you to map a .war file or a directory with static content that you could for example share among other applications deployed on that server. The main advantage over using symlinks is that when using a symlink Tomcat follows it and undeploys the content along with the application. This is done by specifying the attribute alias in the *<context>* tag that looks something like this *alias=”/aliasPath=docBase”* and remember to use the absolute path when specifying the content location.

**Even more changes**

There are more changes like the ones in the manager application or the jdbc-pool library, but I won’t try to review them here. Other ones include embedding improvements which allow you to easily embed Tomcat using the class org.apache.catalina.startup.Tomcat and then do programmatic configuration of the server. This seems extremely useful for things like integration tests and in fact most of Tomcat 7 unit tests use embeded Tomcat. Tomcat 7 prevents session fixation attacks by default by changing session ID on authentication. Internal changes include switching from Valves to Filters. This change is ongoing and should eventually replace all Valves. And one thing you might notice when running the new Tomcat is that access log valve is enabled by default. This means that you’ll get another file in your logs directory which will log all access to Tomcat server. It’s formatted just like the Apache HTTP server’s access log and is useful in a number of situations.