**MODULE 1**

**Introduction to Apache Tomcat 7**

In this module, we introduce the world of Apache Tomcat server. Throughout this module, we

* Describe the Apache Tomcat architecture
* Discuss the requirements for installing and configuring Tomcat

At the end of this module, you will understand the Tomcat architecture, have an instance of Tomcat server installed and running on your computer, and have a sample web application displayed in your browser.

**The Apache Tomcat Server**

The Apache Tomcat server is an open source, Java-based web application container that was created to run servlet and JavaServer Pages (JSP) web applications. It was created under the Apache-Jakarta subproject; however, due to its popularity, it is now hosted as a separate Apache project, where it is supported and enhanced by a group of volunteers from the open source Java community.

Apache Tomcat is very stable and has all of the features of a commercial web application container – yet comes under Open Source Apache License. Tomcat also provides additional functionality that makes it a great choice for developing a complete web application solution. Some of the additional features provided by Tomcat—other than being open source and free—include the Tomcat Manager application, specialized realm implementations, and Tomcat valves.

Currently supported versions on Apache Tomcat are 5.5X, 6.0X, 7.0X and 8.0X. Versions earlier than 5.5 are still available for download, but they are archived and no support is available for them, so users are encouraged to use the latest possible version of Tomcat where available.

Major versions on Apache Tomcat coincide with versions of the Java Servlet specification, or Java Servlet API, released. So, Tomcat 5.5X supports Servlet API 2.3, Tomcat 6.0X supports Servlet API 2.4, and the latest Tomcat 7.0 is a reference implementation of current Servlet API 3.0. In addition to Servlet API versions, Tomcat versions support corresponding JSP API versions.

The JVM compatibility also depends on the version chosen. Table 1-1 provides a cross-reference of Tomcat versions, supported JVM versions, and Servlet API and JSP API releases.

|  |  |  |  |
| --- | --- | --- | --- |
| **Apache Tomcat** | **Servlet API** | **JSP API** | **JDK** |
| 8.0 | 3.1 | 2.3 | 1.7 or later |
| 7.0 | 3.0 | 2.2 | 1.6 or later |
| 6.0 | 2.5 | 2.1 | 1.5 |
| 5.5 | 2.4 | 2.0 | 1.4 |
| 4.1 | 2.3 | 1.2 | 1.3 |
| 3.0 | 2.2 | 1.1 | 1.1 |

***Table 1-1.*** *Tomcat Versions and Supported API and JDK Versions*

This course will cover version 7 of the Apache Tomcat Server. However, most of the content can be applied to versions 5.5 and 6—where that is not possible, it will be clearly stated.

***Figure 1-2.*** *Tomcat architecture with main components*

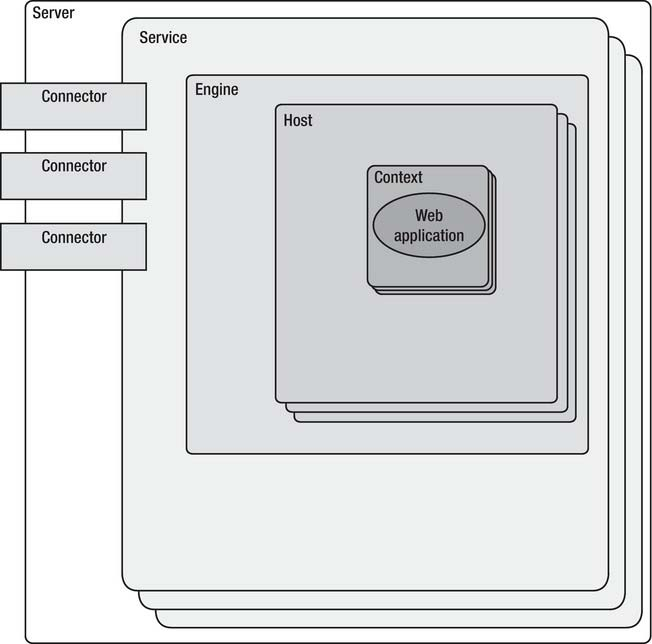
**The Tomcat Manager Web Application**

The Tomcat Manager web application is packaged with the Tomcat server. It is installed in the context path of /manager and provides the basic functionality to manage web applications running in the Tomcat server from any web browser. Some of the provided functionality includes the ability to install, start, stop, remove, and report on web applications. Module 4 covers the details of the Tomcat Manager web application.

**The Architecture of Tomcat**

A Tomcat instance is the top-level component in Tomcat’s container hierarchy. Only one Tomcat instance can live in a single Java Virtual Machine (JVM). This approach makes all other Java applications, running on the same physical machine as Tomcat server, safe in case Tomcat and/or its JVM crashes.

* Note! You can still run multiple instances on same physical box, but as separate Java processes running on separate network ports.



Tomcat instance consists of grouping of the application containers, which exist in the well-defined hierarchy. The key component in that hierarchy is the Catalina servlet engine. Catalina is

the actual Java servlet container implementation as specified in Java Servlet API. Tomcat 7 implements Servlet API 3.0, the latest specification from Sun.

This instance can be broken down into a set of containers including a server, a service, a connector, an engine, a host, and a context. By default, each of these containers is configured using the server.xml file, which we describe later in more detail.

**The Server**

The first container element referenced in this snippet is the *<Server>* element. It represents the entire Catalina servlet engine and is used as a top-level element for a single Tomcat instance. The <Server> element may contain one or more *<Service>* containers.

**The Service**

The next container element is the <*Service>* element, which holds a collection of one or more *<Connector>* elements that share a single *<Engine>* element. N-number of *<Service>* elements may be nested inside a single *<Server>* element.

**The Connector**

The next type of element is the *<Connector>* element, which defines the class that does the actual handling requests and responses to and from a calling client application.

**The Engine**

The third container element is the *<Engine>* element. Each defined *<Service>* can have only one *<Engine>* element, and this single *<Engine>* component handles all requests received by all of the defined *<Connector>* components defined by a parent service.

**The Host**

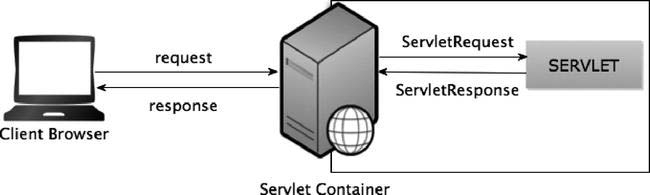
The *<Host>* element defines the virtual hosts that are contained in each instance of a Catalina *<Engine>*. Each *<Host*> can be a parent to one or more web applications, with each being represented by a *<Context>* component.

**The Context**

The *<Context>* element is the most commonly used container in a Tomcat instance. Each *<Context>* element represents an individual web application that is running within a defined *<Host>*. There is no limit to the number of contexts that can be defined within a *<Host>.*

**Servlets**

A Java servlet is a platform-independent web application component that is hosted in a servlet container. Servlets communicate with web clients using a request/response model managed by a servlet container, such as Apache Tomcat. When the user submits a request to the server (i.e., clicks on the link in the browser), the servlet container will accept the request. The servlet container then checks the servlets deployed on it, locates the one that should handle the incoming request (based on the request URL and servlet mapping), and sends the request to it, wrapped as an instance of the ServletRequestclass. After processing the request, the servlet produces a response in the form of a ServletResponse instance, and sends it back to the servlet container. Finally, the servlet container sends the response to the calling client, resulting in the web page rendered in the client browser. Figure 1-2 depicts the execution of a Java servlet.

***Figure 1-2.****The execution of a Java servlet*

The servlet architecture comprises two Java packages: javax.servlet and javax.servlet.http. The javax.servlet package contains the generic interfaces and classes that are implemented and extended by all servlets. The second package is the javax.servlet.http package, which contains all the servlet classes that are specific to HTTP, such as a simple servlet that responds using HTML.

At the heart of this architecture is the interface javax.servlet.Servlet. The base class for all servlets, the Servlet interface, defines five methods. The three most important of these methods and their functions are

* the init(..) method, which initializes the servlet;
* the service() method, which services client requests;
* the destroy() method, which performs cleanup.

**Servlet API 3.0**

Version 3 of the Java Servlet specification was released in December 2009. In addition to the evolution and enhancement of existing features, Servlet API 3.0 brought a few revolutionary changes to the servlet development, namely annotation support for easier configuration, plug-ability using web fragments, and asynchronous servlets support.

**Java Server Pages**

Java Server Pages (JSPs) are a simple but powerful technology used most often to generate dynamic HTML on the server side. They are a direct extension of Java servlets with the purpose of allowing the developer to embed Java logic directly into a requested document. Using JSP Expression Language, you can develop powerful dynamic web pages powered by Java servlets without any Java code. A JSP document must end with a .jsp extension.

The following code snippet contains a simple example of a JSP file:

*<HTML>*

*<BODY>*

*<% out.println("HELLO JSP READER"); %>*

*</BODY>*

*</HTML>*

You can see that this document looks like any other HTML document, with a non-HTML block containing Java code. The source code is stored in a file called hello.jsp and copied to the document directory of the web application that this JSP will be deployed to. When a request is made for this document, the server recognizes the .jsp extension and realizes that special handling is required. The JSP is then passed off to the JSP engine, which is just another servlet that is mapped to the extension .jsp, for processing.

# Java EE at a Glance

Java Platform, Enterprise Edition (Java EE) is the standard in community-driven enterprise software. Java EE is developed using the Java Community Process, with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals. Each release integrates new features that align with industry needs, improves application portability, and increases developer productivity.   
  
Today, Java EE offers a rich enterprise software platform,  and with over 20 compliant Java EE 6 implementations to choose from, low risk and plenty of options.

In every industry, businesses face the challenge of accommodating ever greater demands for high-speed data access, diverse clients, and secure transactions without incurring extensive additional costs. To extend existing IT investments while meeting these demands, developers have consistently adopted the Java Platform, Enterprise Edition.   
  
The de-facto standard for delivering secure, robust, scalable multi-platform applications and services, Java EE technology and its success is predicated on compatibilty which brings Java technology's mission of "Write Once, Run Anywhere" capability to the server. Developers can write applications to the Java EE specification -- and companies can purchase such applications -- and be assured that they are portable across all the Java compatible, Enterprise Edition products available today.

**Java EE Applications**

Java EE applications are made up of components such as JavaServer Pages (JSP), Java servlets, and Enterprise JavaBeans (EJB) modules. These components enable software developers to build large-scale, distributed applications. Developers package Java EE applications in Java Archive (JAR) files (similar to zip files), which can be distributed to production sites. Administrators install Java EE applications onto the Application Server by deploying Java EE JAR files onto one or more server instances (or clusters of instances).

**Containers**

Each server instance includes two containers: web and EJB. A container is a runtime environment that provides services such as security and transaction management to Java EE components. Web components, such as Java Server Pages and servlets, run within the web container. Enterprise JavaBeans run within the EJB container.

**Java EE Services**

The Java EE platform provides services for applications, including:

* **Naming -** A naming and directory service binds objects to names. A Java EE application can locate an object by looking up its Java Naming and Directory Interface (JNDI) name.
* **Security -** The Java Authorization Contract for Containers (JACC) is a set of security contracts defined for the Java EE containers. Based on the client’s identity, containers can restrict access to the container’s resources and services.
* **Transaction management -** A transaction is an indivisible unit of work. For example, transferring funds between bank accounts is a transaction. A transaction management service ensures that a transaction is either completed, or is rolled back.
* **Message Service** - Applications hosted on separate systems can communicate with each other by exchanging messages using the Java™ Message Service (JMS). JMS is an integral part of the Java EE platform and simplifies the task of integrating heterogeneous enterprise applications.

**Web Services**

Clients can access a Java EE application as a remote web service in addition to accessing it through HTTP, RMI/IIOP, and JMS. Web services are implemented using the Java API for XML-based RPC (JAX-RPC). A Java EE application can also act as a client to web services, which would be typical in network applications.

Web Services Description Language (WSDL) is an XML format that describes web service interfaces. Web service consumers can dynamically parse a WSDL document to determine the operations a web service provides and how to execute them. The Application Server distributes web services interface descriptions using a registry that other applications can access through the Java API for XML Registries (JAXR).

**Client Access**

Clients can access Java EE applications in several ways. Browser clients access web applications using hypertext transfer protocol (HTTP). For secure communication, browsers use the HTTP secure (HTTPS) protocol that uses secure sockets layer (SSL).

Rich client applications running in the Application Client Container can directly lookup and access Enterprise JavaBeans using an Object Request Broker (ORB), Remote Method Invocation (RMI) and the internet inter-ORB protocol (IIOP), or IIOP/SSL (secure IIOP). They can access applications and web services using HTTP/HTTPS, JMS, and JAX-RPC. They can use JMS to send messages to and receive messages from applications and message-driven beans.

Clients that conform to the Web Services-Interoperability (WS-I) Basic Profile can access Java EE web services. WS-I is an integral part of the Java EE standard and defines interoperable web services. It enables clients written in any supporting language to access web services deployed to the Application Server.

The best access mechanism depends on the specific application and the anticipated volume of traffic. The Application Server supports separately configurable listeners for HTTP, HTTPS, JMS, IIOP, and IIOP/SSL. You can set up multiple listeners for each protocol for increased scalability and reliability.

Java EE applications can also act as clients of Java EE components such as Enterprise JavaBeans modules deployed on other servers, and can use any of these access mechanisms.

**External Systems and Resources**

On the Java EE platform, an external system is called a resource. For example, a database

management system is a JDBC resource. Each resource is uniquely identified and by its Java Naming and Directory Interface (JNDI) name. Applications access external systems through the following APIs and components:

* **Java Database Connectivity (JDBC) -** A database management system (DBMS) provides facilities for storing, organizing, and retrieving data. Most business applications store data in relational databases, which applications access via JDBC. The Application Server includes the PointBase DBMS for use sample applications and application development and prototyping, though it is not suitable for deployment. The Application Server provides certified JDBC drivers for connecting to major relational databases. These drivers are suitable for deployment.
* **Java Message** Service - Messaging is a method of communication between software components or applications. A messaging client sends messages to, and receives messages from, any other client via a messaging provider that implements the Java Messaging Service (JMS) API. The Application Server includes a high-performance JMS broker, the Sun Java System Message Queue. The Platform Edition of Application Server includes the free Platform Edition of Message Queue. Sun GlassFishEnterprise Server includes Message Queue Enterprise Edition, which supports clustering and failover.
* **Java EE Connectors** - The Java EE Connector architecture enables integrating Java EE applications and existing Enterprise Information Systems (EIS). An application accesses an EIS through a portable Java EE component called a **connector** or **resource adapter**, analogous to using JDBC driver to access an RDBMS. Resource adapters are distributed as standalone Resource Adapter Archive (RAR) modules or included in Java EE application archives. As RARs, they are deployed like other Java EE components. The Application Server includes evaluation resource adapters that integrate with popular EIS.
* **JavaMail -** Through the JavaMail API, applications can connect to a Simple Mail Transport Protocol (SMTP) server to send and receive email.