1 Apache and Jakarta Tomcat

If you've written any Java servlets or JavaServer Pages (JSPs), chances are that you've downloaded Tomcat. That's because Tomcat is a free, feature-complete servlet container that servlet and JSP developers can use to test their code. Tomcat is also Sun's reference implementation of a servlet container, which means that Tomcat's first goal is to be 100% complaint with the versions of the Servlet and JSP specification that it supports.

However, Tomcat is more than just a test server: many individuals and corporations are using Tomcat in production environments because it has proven to be quite stable. Indeed, Tomcat is considered by many to be a worthy addition to the excellent Apache suite of products.

Despite Tomcat's popularity, it suffers from a common shortcoming among open source projects: lack of complete documentation. There is some documentation distributed with Tomcat (mirrored at http://jakarta.apache.org/tomcat/tomcat-4.0-doc/) and there's even an open source effort to write a Tomcat book (http://tomcatbook.sourceforge.net/). Even with these resources, however, there is much room for additional material.

We've created this book to fill in some of the documentation holes and use the combined experience of the authors to help Java developers and system administrators make the most of the Tomcat product. Whether you're looking to learn enough to just get started developing servlets or trying to understand the more arcane aspects of Tomcat configuration, you should find what you're looking for within these pages.

The first two chapters are designed to provide newcomers with some basic background information that will become prerequisite learning for future chapters. If you're a system administrator with no previous Java experience, you are advised to read them; likewise if you're a Java developer who is new to Tomcat. Finally, if you're well informed about Tomcat and Java, you'll probably want to jump straight ahead to Chapter 3, although skimming this chapter and its successor is likely to yield some additions to your present understanding.

We will cover the following points in this chapter:

- ❑ The origins of the Tomcat server
- ❑ The terms of Tomcat's license and how it compares to other open source licenses
- ❑ How Tomcat fits into the Java big picture
- ❑ How Tomcat can be integrated with Apache and other web servers

Humble Beginnings: The Apache Project

One of the earliest web servers was developed by Rob McCool at the National Center for Supercomputer Applications, University of Illinois, Urbana-Champaign, referred to colloquially as the NCSA project, or NCSA for short. In 1995, the NCSA server was quite popular, but its future was uncertain as Rob left NCSA in 1994. A group of developers got together and compiled all the NCSA bug fixes and enhancements they had found and patched them into the NCSA code base. The developers released this new version in April 1995, and called it Apache, which was a sort of acronym for "A PAtCHy Web Server".

Apache was readily accepted by the web-serving community from its earliest days, and less than a year after its release it unseated NCSA to become the most used web server in the world (measured by the total number of servers running Apache), a distinction that it has held ever since (according to Apache's web site). Incidentally, during the same period that Apache's use spread, NCSA's popularity plummeted and by 1999 was officially discontinued by its maintainers.

> *For more information on the history of Apache and its developers, see* *http://httpd.apache.org/ABOUT_APACHE.html.*

Today the Apache web server is available on pretty much any major operating system – as of this writing, downloads are available for 29 different operating systems. Apache can be found running on the some of the largest server farms in the world as well as on some of the smallest devices (including the Linux-based Sharp Zaurus hand-held). In Unix data centers, Apache is as ubiquitous as air conditioning and UPS systems.

While Apache was originally a somewhat mangy collection of miscellaneous patches, today's versions are state-of-the-art, incorporating rock-solid stability with bleeding edge features. The only real competitor to Apache in terms of market share and feature set is Microsoft's Internet Information Services (IIS), which is bundled free with certain versions of the Windows operating system. At the time of writing, Apache's market share was estimated at around 56%, with IIS at a distant 32% (statistics courtesy of http://www.netcraft.com/survey/, June 2002).

It is also worth nothing that Apache has a reputation of being much more secure than Microsoft IIS. When new vulnerabilities are discovered in either server, the Apache developers fix Apache far faster than Microsoft fixes IIS.

The Apache Software Foundation

In 1999, the same folks who wrote the Apache server formed the Apache Software Foundation (ASF). The ASF is a non-profit organization created to facilitate the development of open source software projects. According to their web site, the ASF accomplishes this goal by:

- ❑ Providing a **foundation** for open, collaborative software development projects by supplying hardware, communication, and business infrastructure
- ❑ Creating an independent legal entity to which companies and individuals can **donate resources** and be assured that those resources will be used for the public benefit
- ❑ Providing a means for individual volunteers to be sheltered from **legal suits** directed at the Foundation's projects
- ❑ Protecting the Apache **brand**, as applied to its software products, from being abused by other organizations

In practice, the ASF does indeed sponsor a great many open source projects. While the best known of these projects is likely the aforementioned Apache web server, the ASF hosts many other well-respected and widely used projects.

Apache Projects

The following is a listing of the current Apache projects, all of which can be found at http://www.apache.org/:

Project Name	**Description**
HTTP (Web) Server	The famous Apache web server.
Apache Portable Runtime (APR)	A library of platform-independent C code that forms a portability layer for compiling applications on multiple platforms, such as Linux, Windows, BeOS, and OS/2.
Jakarta	Apache's Java-related efforts; described in detail later in this chapter.
Perl	Umbrella for Apache's well-known `mod_perl` project, as well as other Apache-specific Perl modules. `mod_perl` enables highly efficient integration with Apache and Perl programs.
PHP	PHP is a *very* popular scripting language for dynamically creating HTML or performing business logic in HTML pages. Like `mod_perl`, PHP can be efficiently integrated with Apache.
TCL	Umbrella for Apache's efforts to tightly integrate TCL with the Apache web server (like `mod_perl`).
XML	Umbrella for Apache's cross-platform XML projects, such as the Xerces and Crimson XML parsers and the AXIS SOAP engine.

The Jakarta Project

Of most relevance to this book is Apache's **Jakarta** project, of which the Tomcat server is a subproject. The Jakarta project is the umbrella under which the ASF sponsors the development of Java subprojects. At the time of writing, there is an impressive array of more than twenty of these. They are divided into three different categories: "Libraries, Tools, and APIs", "Frameworks and Engines", and "Server Applications". We will highlight two projects from the first category (Ant and Log4J), one from the framework category (Struts), and, of course, Tomcat.

Tomcat

The Jakarta Tomcat project has its origins in the earliest days of Java's servlet technology. Servlets plug into special web servers, called servlet containers (originally called servlet engines). Sun created the first servlet container, called the Java Web Server, which demonstrated the technology but wasn't terribly robust. Meanwhile, the ASF folks created the JServ product, which was a servlet engine that integrated with the Apache web server.

In 1999, Sun donated their servlet container code to the ASF, and the two projects were merged to create the Tomcat server. Today, Tomcat serves as Sun's official reference implementation (RI), which means that Tomcat's first priority is to be fully compliant with the Servlet and JSP specifications published by Sun. JSP pages are simply an alternative, HTML-like way to write servlets. We will discuss all this in more detail in the next chapter.

A reference implementation also has the side benefit of honing the specification. As developers seek to put in code that has been defined in the specifications, problems in implementation requirements and conflicts within the specifications are highlighted.

A reference implementation is in principal completely specification-compliant and therefore can be very valuable, especially for people who are using very advanced parts of the specification. The reference implementation is available at the same time as the public release of the specifications, which means that Tomcat is usually the first server out there that provides the enhanced specification features when a new specification version is completed.

The first version of Tomcat was the 3.x series, and it served as the reference implementation of the Servlet 2.2 and JSP 1.1 specifications. The Tomcat 3.x series was descended from the original code that Sun provided to the ASF in 1999.

In 2001, Tomcat 4.0 (codenamed Catalina) was released, and was a complete redesign of the Tomcat architecture and had a new code base. The Tomcat 4.x series, which is current as of this writing, is the reference implementation of the Servlet 2.3 and JSP 1.2 specifications.

At the time of writing, the latest stable version is 4.0.4. Hints of Tomcat 5.0 are on the horizon, as the new Servlet 2.4 and JSP 2.0 specifications are nearing release and Tomcat 5.0 will need to implement those specifications.

Ant

Ant is a tool to automate building and deploying applications that range from the very simple to the extremely complex. If you're familiar with Unix, you might think this sounds like the ubiquitous `make` tool. In fact, Ant was created by a group of people who wanted to create a replacement for `make`. You can read about their comments on the subject at http://jakarta.apache.org/ant/.

Ant can be used for building applications in any language, and it can be used on any platform that has a Java 1.1 virtual machine or better. Ant's versatility can also be extended with Java plug-ins. Ant won awards from both the Software Development and Java World magazines in 2002, and it is extremely popular amongst developers.

Log4J

Developers generally use logging for two purposes: debugging during development and monitoring when the system is in production. When developing systems, developers usually prefer logging to be as verbose as possible, and aren't concerned with its impact on the system's overall performance. However, when a system is deployed into production, developers want logging to impact performance as little as possible.

Log4J represents more than five years of work towards creating the ideal logging solution for Java programs, combining the desire for generation of rich data at development time with the need for minimal performance degradation in production environments. If your current logging technique is executing something like `System.out.println()`, you owe it to yourself to investigate this project and see what else is possible with logging.

Log4J Versus JDK 1.4 Logging

Java 1.4 introduced a logging mechanism to Java as part of the standard J2SE platform. Log4J has been in its present form since late 1999, and thus predates the JDK 1.4 logging mechanism by a little more than 2 years (JDK 1.4 went final in early 2002). When it was learned that Java 1.4 would incorporate logging, the Log4J group lobbied to have its product incorporated into Java as the official logging mechanism for the platform. However, that did not happen.

With the release of Java 1.4, Log4J didn't disappear, and doesn't intend to. Log4J provides two advantages over the Java 1.4 logging mechanism: it has more features and it can be used with Java 1.1 or later.

Struts

The current architectural best practice for web applications is the Model View Controller (MVC) design pattern. Under this model, the application is divided into three logical layers (also called tiers): the View, which represents the user interface; the Model, which represents the business logic specific to the application including any persistent data store (for example, a database); and the Controller, which coordinates how the View and the Model interact, and takes care of any other general application behavior (for example, application lifecycle issues). We'll see more on the MVC architecture in the next chapter.

Servlets and JavaServer Pages are the standard Java way to create web applications. They provide an efficient interface to the Web's HTTP protocol. However, developers who wish to create an MVC architecture with servlets and JSP must still do quite a bit of work.

Many third-party frameworks have been created which attempt to relieve developers from the burden of implementing their own MVC architecture, freeing them to instead focus on solving the unique business problems of their organization. Struts is one of these frameworks. Struts has gained an excellent reputation in the development community as being well-designed and very flexible.

Other Jakarta Subprojects

There are many other Jakarta subprojects, including: Lucene, a full-featured search engine; Jetspeed, a portal server; and James, a mail server. See these and others at http://jakarta.apache.org/.

Distributing Tomcat

Tomcat is open source software, and as such is free and freely distributable. However, if you have much experience in dealing with open source software, you're probably aware that the terms of distribution can vary from project to project.

Most open source software is released with an accompanying license that states what may and may not be done to the software. There are at least forty different open source licenses out there, each of which has slightly different terms.

Providing a primer on all of the various open source licenses is beyond the scope of this chapter, but the license governing Tomcat will be discussed here and compared with a few of the more popular open source licenses.

Tomcat is distributed under the Apache License, which can be read from the `$CATALINA_HOME/LICENSE` file. The key points of this license state that:

- The Apache License must be included with any redistributions of Tomcat's sourcecode or binaries
- Any documentation included with a redistribution must give a nod to the ASF
- Products derived from the Tomcat sourcecode can't use the terms "Tomcat", "The Jakarta Project", "Apache", or "Apache Software Foundation" to endorse or promote their software without prior written permission from the ASF
- Tomcat has no warranty of any kind

However, through omission, the license contains these additional implicit permissions:

- Tomcat can be used by any entity, commercial or non-commercial, for free without limitation
- Those who make modifications to Tomcat and distribute their modified version do not have to include the sourcecode of their modifications
- Those who make modifications to Tomcat do not have to donate their modifications back to the ASF

Thus, you're free to deploy Tomcat in your company in any way you see fit. It can be your production web server or your test servlet container used by your developers. You can also redistribute Tomcat with any commercial application that you may be selling, provided that you include the license and give credit to the ASF. You can even use the Tomcat sourcecode as the foundation for your own commercial product

Comparison with Other Licenses

Among the previously mentioned rather large group of other open source licenses, there are two licenses which are particularly popular at the present time: the GNU General Public License (GPL) and the GNU Lesser General Public License (LGPL). Let's take a look at how each of these licenses compare to the Apache License.

GPL

The GNU Project created and actively evangelizes the GPL. The GNU Project is somewhat similar to the ASF, with the exception that the GNU Project would like all of the non-free (that is, closed source or proprietary) software in the world to become free; the ASF has no (stated) desire to do this and simply wants to provide free software.

What Does It Mean to Be Free?

Free software can mean one of two entirely different things: software that doesn't cost anything, and software that can be freely copied, distributed, and modified by anyone (thus the sourcecode is included or available); such software can be distributed either for free or for a fee. A simpler way to explain the difference between these two types of free is "free as in free beer" and "free as in free speech". The GNU Project's goal is to create free software of the latter category. All uses of the phrase "free software" in the remainder of this section will use this definition.

The differences between the Apache License and the GPL thus mirror the distinct philosophies of the two organizations. Specifically, the GPL has these key differences from the Apache License:

- No non-free software may contain GPL-licensed products or use GPL-licensed sourcecode. If non-free software is found to contain GPL-licensed binaries or code, it must remove such elements or become free software itself.
- All modifications made to GPL-licensed products must be released as free software if the modifications are also publicly released.

These two differences have huge implications for commercial enterprises. If Tomcat were licensed under the GPL, any product that contained Tomcat would also have to be free software.

Furthermore, while the Apache License permits an organization to make modifications to Tomcat and sell it under a different name as a closed source product, the GPL would not allow any such act to occur; the new derived product would also have to be released as free software.

LGPL

The LGPL is similar to the GPL, with one major difference: non-free software may contain LGPL-licensed products. The LGPL license is intended primarily for software libraries that are themselves free software but whose authors want them to be available for use by companies who produce non-free software.

If Tomcat were licensed under the LGPL, it could be embedded in non-free software, but Tomcat could not itself be modified and released as a non-free software product.

For more information on the GPL and LGPL licenses, see http://www.gnu.org/.

Other Licenses

Understanding and comparing open source licenses can be a rather complex task. The explanations above are an attempt to simplify the issues. For more detailed information on these and other licenses, there are two specific resources that can help you:

- The Open Source Initiative (OSI) maintains a database of open source licenses. Visit them at http://www.opensource.org/.
- The GNU Project, mentioned above, has an extensive comparison of open source licenses with the GPL license. See it at http://www.gnu.org/licenses/license-list.html.

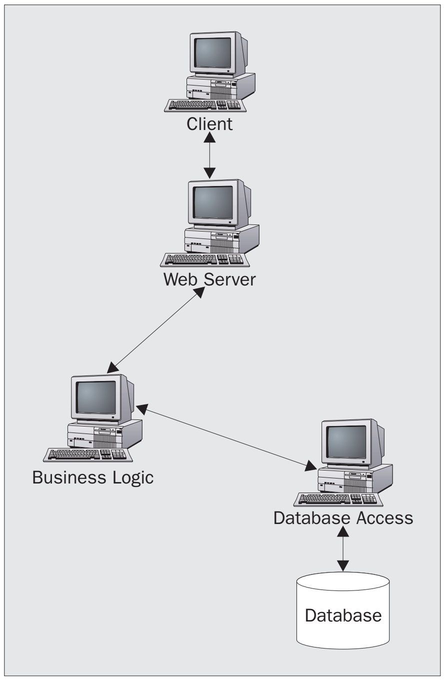
The Big Picture: J2EE

As a servlet container, Tomcat is a key component of a larger set of standards collectively referred to as the Java 2 Platform, Enterprise Edition (J2EE). J2EE defines a group of Java-based code libraries (called APIs in the Java world) that are suited to creating web applications for the enterprise (that is, a large company). To be sure, companies of any size can take advantage of the J2EE technologies, but J2EE is especially designed to solve the problems associated with the creation of large software systems. Java developers can download all of the J2EE APIs in a single ZIP archive, which comes complete with binaries and documentation.

Distributed Systems

We saw in the previous section that J2EE is designed with the creation of large software systems in mind. You, the reader, may ask, "What is so different about large software systems versus small systems?" The answer lies in the notion of **distributed systems**.

A distributed system is one in which the various components of the system are distributed across multiple different machines. For example, in a given web application, one server might handle receiving and responding to HTTP requests while another server handles all the business logic, and another server handles all the database I/O:



So why create a distributed system? Large web applications and enterprise systems can have enormous demands placed upon them. Frequently, these demands are larger than a single server can meet. While one may be tempted to simply upgrade the server, adding more processors and more memory to it, these upgrades can only stretch the server's capacity so far. Every server, no matter how expandable and efficient it may be, will run up against limitations.

In these situations, where a single server simply cannot meet the needs of a software system, a distributed system is ideal. Multiple servers can work together to meet demands that a single server could not.

But, why bother creating a distributed system? Why not just copy the same application and deploy it on multiple servers? Surely load balancing – splitting up the requests amongst the different servers – can then handle the load?

It turns out that by separating the different logical components of an application, system administrators are free to tune the various components of the distributed system according to their unique needs. For example, the servers that read and write to the database will require different optimizations than those servers which are communicating with web clients via HTTP over TCP.

The J2EE APIs

Creating the distributed systems that we've described in the preceding paragraphs is not an easy task. We mentioned a few paragraphs ago that J2EE is a collection of code libraries called APIs. The J2EE APIs are designed to work together to simplify the creation of robust and efficient distributed systems. Here is a list of some of the key J2EE technologies and a brief description of each:

J2EE API	**Description**
Enterprise JavaBeans (EJB)	EJB technology provides a simple mechanism for creating distributed business logic components. EJB authors follow a simple pattern to write business logic and the rest of the low-level details relating to lifecycle, distribution, persistence, and so on are handled automatically.
Java Message Service (JMS)	JMS provides asynchronous messaging capability to J2EE applications.
Java Naming and Directory Service (JNDI)	JNDI enables J2EE applications to communicate with registries and directories. A registry/directory is a centralized location for storing business information. JNDI supports the industry standard LDAP protocol and interfaces with many other popular registry standards. JNDI makes it possible to centralize the configuration of a distributed system.
Servlets	As explained earlier in this chapter, servlets work with special servers called servlet containers to process HTTP requests and send HTTP responses. Servlets often work directly with EJBs.
JavaServer Pages (JSP)	JSP technology is an alternative, HTML-like interface for creating servlets. At runtime, the servlet container converts a JSP into a servlet.

Table continued on following page

J2EE API	Description
Java Transaction API (JTA)	JTA enables a web application to gracefully handle failures in one or more of its components by establishing transactions. During a transaction, multiple events can occur, and if any one of them fails, the state of the application can be rolled back to how it was before the transaction began. This technology provides the robustness of relational database transactional technology across an entire distributed application.
CORBA	CORBA technology in Java enables J2EE web applications to communicate with distributed components written in other languages that support the language-independent CORBA standard.
JDBC	JDBC enables Java programs to communicate with relational databases.
Connector	The Connector API enables Java programs to create an abstraction layer for connecting with legacy systems that don't implement other J2EE-supported technologies.
JavaMail	JavaMail provides the ability to send and receive e-mail via the industry standard POP/SMTP/IMAP protocols.
Java XML Parser (JAXP)	JAXP gives J2EE applications the ability to speak XML.

Note that the J2EE APIs are not designed to help developers write their own server software. Rather, they are to designed to work with a special kind of server, called a J2EE **application server** (app server). J2EE app servers are available on many operating systems, including Windows, Linux, Solaris, and Mac OS X. To be considered an official J2EE app server, the app server vendor must show that the app server complies with the J2EE standards and then purchase a J2EE license. Such vendors are referred to as **J2EE licensees**.

Agree On Standards, Compete On Implementation

If developers follow the J2EE standards, they can use a compliant, licensed J2EE app server from any vendor and it is guaranteed to work with their application. This flexibility is intended to help companies avoid vendor lock-in problems, and thus they can enjoy the benefits of a competitive marketplace. The Java slogan along these lines is "Agree on standards, compete on implementation," meaning that the vendors all cooperate on establishing universal J2EE standards and then work hard to create the best app server that supports those standards.

Pick and Choose

However, not all developers want to create the kinds of distributed systems that the entire collection of J2EE APIs are designed to support. Fortunately, it is possible to pick and choose those APIs that are necessary for your projects. For example, if all you need to do is write a GUI LDAP client, the JNDI API is the only J2EE API you'll need to use.

If you want to create a dynamic web site but don't need the additional J2EE APIs, the Servlet and JSP APIs may be all you'll want. As was mentioned earlier in this chapter, a servlet plugs in to a special server called a servlet container (and JSP is an alternative way of writing a servlet). The J2EE app server contains a servlet container, but if all you want to is write servlets/JSP, then you don't need the full app server – you just need a servlet container.

Tomcat is just such a servlet container. As a servlet container, Tomcat is only required to implement the Servlet and JSP APIs, and thus is not considered a J2EE app server. However, as J2EE app servers must themselves contain a servlet container to support the servlet/JSP APIs, J2EE app servers can embed Tomcat into their code to provide support for the Servlet and JSP APIs. One example of just such an application server is the popular open source JBoss J2EE app server (http://www.jboss.org/).

For more information on J2EE and its various APIs, visit http://java.sun.com/j2ee/.

Using Tomcat with the Apache Web Server

Tomcat's purpose is to provide standards-compliant support for servlets and JSP pages. The purpose of servlets and JSP pages is to generate web content such as HTML files or GIF files on demand using changing data. Web content that is generated on demand is said to be dynamic. On the other hand, web content that never changes and is served up as-is is called static. Web applications commonly include a great deal of static content, such as images or Cascading Style Sheets (CSS).

While Tomcat is capable of serving both dynamic and static content, it is not as fast or feature-rich as the Apache web server with regard to static content. While it would be possible for Tomcat to be extended to support the same features that Apache does for serving up static content, it would take a great deal of time; Apache has been under development for many years. Also, because Apache is written entirely in C and takes advantage of platform-specific features, it is unlikely that Tomcat, a 100% Java application, could ever perform well as Apache.

Recognizing that the advantages of Apache would complement the advantages of Tomcat, the earliest versions of Tomcat included a **connector** that enabled Tomcat and Apache to work together. In this arrangement, Apache receives all of the HTTP requests made to the web application. Apache then recognizes which requests are intended for servlets/JSP pages, and passes these requests to Tomcat. Tomcat fulfils the request and passes the response back to Apache, which then returns the response to the requestor.

The Apache connector was initially crucial to the Tomcat 3.x series, because its support for both static content and its implementation of the HTTP protocol were somewhat limited.

The Tomcat 4.x series, however, features a much nicer implementation of HTTP and better support for serving up static content, and should by itself be sufficient for people who aren't looking to max out performance but simply need HTTP standards compliance. However, as mentioned above, Apache will most likely always have superior performance and options when it comes to serving up static content and communicating with clients via HTTP, and, for this reason, anyone who is using Tomcat for high-traffic web applications may want to consider using Apache and Tomcat together.

Apache Connectors

For interfacing with Apache, Tomcat 4.x supports two different types of connectors: AJP and WARP; AJP and WARP refer to two different protocols that govern how the connector communicates with Apache. The Apache JServ Protocol (AJP) dates back to the Apache JServ product that we mentioned earlier in this chapter. The first connector to implement this protocol, called `mod_jserv`, was written for the initial JServ product and continued to function with the Tomcat 3.x series. The newest AJP-based connector is `mod_jk2`.

The WARP protocol was created for the Tomcat 4.x series, and `mod_webapp` is the name of the only connector that currently implements this protocol. The WARP protocol is intended to provide greater flexibility and greater performance than the AJP protocol.

Getting the Apache connectors to work properly can be tricky, and finding helpful documentation is even trickier. However, Chapters 11-13 will help to clarify this sometimes murky subject.

Tomcat and Other Web Servers

The AJP protocol introduced above can also be used to integrate Tomcat with two other web servers: Microsoft's Internet Information Services (IIS) and Netscape Enterprise Server (NES). This topic will be covered in detail in Chapter 14.

If you're not using either Apache, IIS, or NES then don't give up hope entirely. It is still very possible to integrate Tomcat with other web servers, even one that resides on the same machine. All one has to do is set up Tomcat to run on a port other than 80 – the default HTTP port. Note that, by default, Tomcat runs on port 8080. Thus, any normal web requests to a server will go to an HTTP server sitting on port 80, and any requests to port 8080 will go to Tomcat. You can then design your web application's HTML to request its static resources from the web server on port 80. For more information on this topic, see Chapter 5.

Summary

As we draw this chapter to a close, let's review some of the key points that have been covered:

- The Apache Software Foundation (ASF) is a non-profit organization created to provide the world with quality open source software.
- The ASF maintains a whole series of open source projects. The ASF's Java projects are collected under the umbrella of a parent project called Jakarta.
- Tomcat is one of the most popular subprojects in the Jakarta project.
- Tomcat can be freely used in any organization. It can be freely redistributed in any commercial project so long as its license is also included with the redistribution and proper recognition is given.
- J2EE is a series of technologies designed to make the creation of complex, distributed applications easier.
- Tomcat is a fully J2EE-compliant servlet container, and is the official Reference Implementation for the Java Servlet and JavaServer Pages technologies.
- While Tomcat can also function as a web server, it can also be integrated with other web servers.
- Tomcat has special support for integrating with the Apache, IIS, and NES servers.

This chapter has been an overview of Tomcat, but what do Tomcat-served web applications look like? What files comprise them? We've covered that a little bit in this chapter, but in the next chapter we'll go into this subject in detail.