1

Apache and Jakarta Tomcat

If you've written any Java Servlets or JavaServer Pages (JSPs), chances are good that you've downloaded Tomcat. That's because Tomcat is a free, feature-complete **Servlet container** that developers of Servlets and JSPs can use to test their code. Tomcat is also Sun Microsystems' reference implementation of a Servlet container, which means that Tomcat's first goal is to be 100 percent compliant with the versions of the Servlet and JSP API specifications that it supports. Sun Microsystems (Sun) is the creator of the Java programming language and functions as its steward.

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 of which it is a member.

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

This book has been created to fill in some of the documentation holes, and uses the combined experience of the authors to help Java developers and system administrators make the most of the Tomcat product. Whether you're trying 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 subsequent chapters. If you're a system administrator with no previous Java experience, you are advised to read these first two chapters, and likewise if you're a Java developer who is new to Tomcat. If you're well informed about Tomcat and Java, you'll probably want to jump straight ahead to Chapter 3, "Tomcat Installation," although skimming this chapter and its successor is likely to add to your present understanding.

The following points are discussed 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"
- An overview of integrating Tomcat 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 (NCSA), 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 because McCool 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 somewhat of an 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 was spreading, NCSA's popularity was plummeting, and by 1999, NCSA 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 just about any major operating system (as of this writing, binary downloads of Apache are available for 29 different operating systems, and Apache can be compiled on dozens more). Apache can be found running on some of the largest server farms in the world, as well as on some of the smallest devices (including several hand-held devices). 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 Server (IIS), which is bundled free with certain versions of the Windows operating system. As of this writing, Apache's market share is estimated at around 67 percent, with IIS at a distant 21 percent (statistics courtesy of `http://news.netcraft.com/archives/web_server_survey.html, January 2004`).

It is also worth noting 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 nonprofit organization that was created to facilitate the development of open source software projects. Tomcat is developed under the auspices of the ASF. According to their Web site, the ASF accomplishes this goal by the following:

- 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 ASF 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, including such respected industry standards as the following:

- *PHP* — Perhaps the world's most popular Web scripting language
- *Xerces* — A Java/C++ XML parser with JAXP bindings
- *Ant* — A Java-based build system (and much more)
- *Axis* — A Java-based Web Services engine

The list of ASF-sponsored projects is growing fast. Visit `www.apache.org` to see the latest list.

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 an umbrella under which the ASF sponsors the development of many Java subprojects. As of this writing, there is an impressive array of more than 20 such projects. They are divided into the following three categories:

- Libraries, tools, and APIs
- Frameworks and engines
- Server applications

Tomcat fits into the latter of these three.

Tomcat

The Jakarta Tomcat project has its origins in the earliest days of Java's Servlet technology. **Servlets** are a certain type of Java application that plugs 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 **JavaServer Pages** (JSP) specifications published by Sun. JSP pages are simply an alternative, HTML-like way to write Servlets. This is discussed in more detail in Chapter 2, "JSP and Servlets."

An RI also has the side benefit of refining the specification. As an RI team seeks to implement a committee-created specification (for example, the Servlet specification) in the real world, unanticipated problems emerge that must be resolved before the rest of the world can successfully make use of the specifications. As a corollary, if an RI of a specification is successfully created, it demonstrates to the rest of the world that the specification is technically viable.

The RI is in principle completely specification-compliant and therefore can be very valuable, especially for people who are using very advanced parts of the specification. The RI is available at the same time as the public release of the specification, which means that Tomcat is usually the first server to provide 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 (code-named Catalina) was released. Catalina was a complete redesign of the Tomcat architecture, and built on a new code base. The Tomcat 4.x series is the RI of the Servlet 2.3 and JSP 1.2 specifications.

Tomcat 5.0, the latest release of Tomcat, is an implementation of the new Servlet 2.4 and JSP 2.0 API specifications. In addition to supporting the new features of these specifications, Tomcat 5 also introduces many improvements over its predecessor, such as better JMX support and various performance optimizations.

Earlier in this chapter, it was mentioned that Tomcat is Sun's RI of the Servlet and JSP APIs. Yet, it is the ASF that develops Tomcat, not Sun. It turns out that Sun provides resources to the ASF in the form of Sun employees paid to work on Tomcat. Sun has a long history of donating resources to the open source community in this and other ways.

Other Jakarta Subprojects

Wise Java Web application developers who want to save valuable time will familiarize themselves with the other Jakarta projects. These peer projects of Tomcat include the following:

- *Commons* — A collection of commonly needed utilities, such as alternative implementations of the Collection Framework interfaces, an HTTP client for initiating HTTP requests from a Java application, and much more
- *JMeter* — An HTTP load simulator used for determining just how heavy a load Web servers and applications can withstand
- *Lucene* — A high-quality search engine written by at least one of the folks who brought us the Excite! search engine
- *Log4J* — A popular logging framework with more features than Java 1.4's logging API, and support for all versions of Java since 1.1
- *ORO* and *Regexp* — Two different implementations of Java-based regular expression engines
- *POI* — An effort to create a Java API for reading/writing the Microsoft Office file formats
- *Struts* — Perhaps the most popular Java framework for creating Web applications

This list is by no means comprehensive, and more projects are added frequently.

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. At least 40 different open source licenses are in use, 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 is 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 the following:

- The Apache License must be included with any redistributions of Tomcat's source code or binaries.
- Any documentation included with a redistribution must give a nod to the ASF.
- Products derived from the Tomcat source code 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 the following additional implicit permissions:

- Tomcat can be used by any entity (commercial or noncommercial) for free without limitation.
- Those who make modifications to Tomcat and distribute their modified version do not have to include the source code of their modifications.
- Those who make modifications to Tomcat do not have to donate their modifications 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 source code as the foundation for your own commercial product.

Comparison with Other Licenses

Among the previously mentioned and rather large group of other open source licenses, two licenses 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 compares 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 nonfree (that is, closed source or proprietary) software in the world to become free. The ASF has no such (stated) desire and simply wants to provide free software.

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 source code is included or is easily accessible). Such software can be distributed either free or for a fee. A simpler way to explain the difference between these two types of free is to compare "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 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 the following key differences from the Apache License:

- No nonfree software may contain GPL-licensed products or use GPL-licensed source code. If nonfree 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: Nonfree 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 nonfree software.

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

For more information on the GPL and LGPL licenses, see `www.gnu.org`.

Other Licenses

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

- The Open Source Initiative (OSI) maintains a database of open source licenses. Visit them at `www.opensource.org`.
- The GNU Project has an extensive comparison of open source licenses with the GPL license. See it at `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 Enterprise Edition (**J2EE**) platform. The J2EE standard defines a group of Java-based **APIs** that are suited to creating Web applications for **enterprises** (that is, large companies). 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.

J2EE is built on the Java 2 Standard Edition (J2SE), which includes the Java binaries (such as the JVM and bytecode compiler), as well as the core Java code libraries. J2EE depends on J2SE to function. Both the J2SE and J2EE can be obtained from `http://java.sun.com`. Both J2SE and J2EE are referred to as **platforms**, as they provide core functionality that acts as a sort of platform or foundation upon which applications can be built.

Java APIs

As mentioned, J2EE is a standardized collection of Java APIs. The term **API** (or **application programming interface**) is used by software developers in general to describe services made available to applications by an underlying service provider (such as an operating system). In the Java world, this term is used to describe many of the services that the Java Virtual Machine (JVM) and its code libraries make available to Java programs.

An important characteristic of APIs is that they are separated from the services that provide them. In other words, an API is a kind of technical contract defining the functionality that two parties must provide: a service provider (often called an **implementation**), and an application. If both parties adhere to the contract, an API is **pluggable** (that is, a new service provider can be plugged into the relationship). Of course, if a service provider fails to conform to the contract, the applications that use the API will fail to function properly.

The Java Community Process (JCP)

APIs in the Java world are created and modified by a standards body known as the Java Community Process (JCP). The JCP is composed of hundreds of **Java Specification Requests** (**JSRs**). Each JSR is a request to either change an existing aspect of Java (including its APIs) or introduce a new API or feature to Java. New JSRs can be submitted by a **member** of the JCP. Anyone can become a member of the JCP and, notably, individuals may do so at no cost (organizations pay a nominal fee). Once submitted, the JCP **Executive Committee** must approve the JSR. The Executive Committee consists of JCP members who have been elected to three-year terms in an annual election.

When a JSR is approved, the submitter becomes the **Spec Lead**. The Spec Lead forms an **Expert Group** composed of JCP members who assist the Spec Lead in creating a specification detailing the change or addition to the Java language. The Expert Group shepherds the specification along through various review processes (to other JCP members and to the public) until, finally, the JSR is judged completed and is approved by the Executive Committee. If a JSR results in an API, the Expert Group must also provide a reference implementation of the API (discussed earlier in this chapter in the context of Tomcat) and a **technology compatibility kit** (TCK) that other implementers can use to verify compatibility with the API.

Thus, via the JCP, any Java developer can influence the Java platforms, either by submitting a JSR, by becoming a member of an existing JSR's Expert Group, or by simply giving feedback to JSR Expert Groups. While not the first attempt to create a technology standards body, the JCP is probably the world's best combination of accessibility and influence. As a contrast, the influential World Wide Web Consortium (W3C) standards body charges almost $6,000 for individuals to join. Visit the JCP at `www.jcp.org`.

The J2EE APIs

As mentioned, the J2EE 1.4 platform consists of many individual APIs. The Servlet and JSP APIs are two of these. The following table describes some of the other J2EE APIs.

J2EE API	Description
Enterprise JavaBeans (EJB)	Provides a mechanism that is intended to make it easy for Java developers to use advanced features in their components, such as remote method invocation (RMI), object/relational mapping (that is, saving Java objects to a relational database), distributed transactions across multiple data sources, statefulness, and so on.
Java Message Service (JMS)	Provides high-performance asynchronous messaging. Among other things, enables J2EE applications to communicate with non-Java systems on top of various transports.

J2EE API	Description
JAX-RPC	Binds Java objects to Web services. This is the key API around which J2EE Web services support revolves.
Java Management Extensions (JMX)	Standardizes a mechanism for interactively monitoring and managing applications at run-time.
Java Transaction API (JTA)	JTA enables applications 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. JTA provides the functionality of database-transactions technology across an entire distributed application.
Connector	Provides an abstraction layer for connecting with enterprise information systems, especially those that have no knowledge of Java and expose no Java-compatible interfaces (such as JDBC drivers).
JavaMail	Provides the capability to send and receive e-mail via the industry-standard POP/SMTP/IMAP protocols.

In addition to the J2EE-specific APIs, J2EE applications also rely heavily on J2SE APIs. In fact, over the years, several of the J2EE APIs have been migrated to the J2SE platform. Two such APIs are the Java Naming and Directory Interface (JNDI), used for interfacing with LDAP-compliant directories (and much more), and the Java API for XML Processing (JAXP), which is used for parsing and transforming XML (using XSLT). The vast collection of J2EE and J2SE APIs form a platform for enterprise software development unparalleled in the industry. In the coming years, Microsoft's .NET platform may present itself as a viable alternative to J2EE, but that day is still far off.

J2EE Application Servers

As mentioned, an API simply defines services that a service provider (i.e., the implementation) makes available to applications. Thus, an API without an implementation is useless. While the JCP does provide RIs of all the APIs, most of the J2EE API reference implementations are inefficient and difficult to use (with the exception of Tomcat, of course). Furthermore, the various J2EE RIs are not well integrated, making it all the more difficult to write applications that make use of several different APIs. Enter the **J2EE application server**.

Various third parties provide commercial-grade implementations of the J2EE APIs. These implementations are typically packaged as a **J2EE application server**. Whereas Tomcat provides an implementation of the Servlet and JSP APIs (and is thus called a **Servlet container**), application servers provide a superset of Tomcat's functionality: the Servlet and JSP APIs plus all the other J2EE APIs, and some J2SE APIs (such as JNDI).

Dozens of vendors have created **J2EE-compatible** application servers. Being "J2EE-compliant" means that a vendor of an application server has paid Sun a considerable sum and passed various compatibility tests. Such vendors are said to be **J2EE licensees**.

For a list of the J2EE licensees, visit ***`http://java.sun.com/j2ee/licensees.html`***.

It is worth mentioning that several open source J2EE application servers are emerging. Currently, none of these products have paid Sun the requisite fees to become officially J2EE-compatible, but the products make informal claims stating that they are as good as such. One example is the popular JBoss project. The ASF has itself recently begun a project to develop a J2EE-compatible application server named Geronimo.

"Agree on Standards, Compete on Implementation"

Developers who use the J2EE APIs can use a J2EE-compatible application server from any vendor, and it is guaranteed to work with their applications. This flexibility is intended to help customers avoid vendor lock-in problems, enabling users to 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 in establishing universal J2EE standards (through participation in the JCP) and then work hard to create the best application server implementation of those standards.

That's the theory, at least. In reality, this happy vision of vendor neutrality and open standards is slightly marred by at least two factors. First, each application server is likely to have its own eccentricities and bugs. This leads to a popular variation on the famous "Write Once, Run Anywhere" Java slogan: "Write Once, Test Everywhere." Second, vendors are rarely altruistic. Each application server typically includes a series of powerful features that are outside the scope of the J2EE APIs. Once developers take advantage of these features, their application is no longer portable, resulting in vendor lock-in. Developers must, therefore, be vigilant to maintain their application's portability, if such a capability is desirable.

Tomcat and Application Servers

Up to this point, Tomcat has been referred to as an implementation of the Servlet/JSP APIs (i.e., a Servlet container). However, Tomcat is more than this. It also provides an implementation of the JNDI and JMX APIs. However, Tomcat is not an application server; it doesn't provide support for even a majority of the J2EE APIs.

Interestingly, many application servers actually use Tomcat as their implementation of the Servlet and JSP APIs. Because Tomcat permits developers to embed Tomcat in their applications with only a one-line acknowledgment, many commercial application servers quietly rely on Tomcat without emphasizing that fact. The JBoss application server mentioned previously makes explicit use of Tomcat (although it can also use other Servlet/JSP implementations).

Developers seeking to create Java Web applications that utilize the Servlet, JSP, JNDI, and JMX APIs will find Tomcat an excellent solution. However, those seeking support for additional APIs will probably be better served to either find an application server, or use Tomcat in addition to an application server. A third option is to find an implementation of the individual J2EE APIs required and use them in conjunction with Tomcat. This piecemeal approach is perfectly valid, although integration problems are likely to manifest themselves.

Tomcat and Web Servers

Tomcat's purpose is to provide standards-compliant support for Servlets and JSPs. The purpose of Servlets and JSPs 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**. Conversely, 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 Web servers written specifically to serve up static content. While it would be possible for Tomcat to be extended to support many additional features for serving up static content, it would take a great deal of time. The popular Apache Web server (and others like it) has been under development for many years. In addition, because most Web servers are written in low-level languages such as C and take advantage of platform-specific features, it is unlikely that Tomcat (a 100-percent Java application) could ever perform as well as such products.

Recognizing that Tomcat could enjoy a synergistic relationship with conventional Web servers, the earliest versions of Tomcat included a connector that enabled Tomcat and Apache to work together. In such a relationship, Apache receives all of the HTTP requests made to the Web application. Apache then recognizes which requests are intended for Servlets/JSPs, and passes these requests to Tomcat. Tomcat fulfills 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.

Starting with the 4.x series, Tomcat features a much more complete implementation of HTTP and better support for serving up static content, and should by itself be sufficient for people who aren't looking for maximum performance, but do need compliance with HTTP. However, as mentioned above, Apache and other Web servers will most likely always have superior performance and options when it comes to serving up static content and communicating with clients via HTTP. For this reason, anyone who is using Tomcat for high-traffic Web applications may want to consider using Tomcat together with another Web server.

This book describes how to integrate Tomcat with the Apache and Internet Information Server (IIS) Web servers in Chapters 11–13.

If you're not using either Apache or IIS, 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 you have 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 are sent to an HTTP server sitting on port 80, and any requests to port 8080 are sent to Tomcat. You can then design your Web application's HTML to request its static resources from the Web server on port 80.

Summary

To conclude this chapter overview of Tomcat, let's review some of the key points that have been discussed:

- The Apache Software Foundation (ASF) is a nonprofit organization created to provide the world with quality open source software.
- The ASF maintains an extensive collection of open source projects. Many of 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 Java APIs designed to facilitate the creation of complex enterprise applications. J2EE-compatible application servers provide implementations of the J2EE APIs.
- Tomcat is a J2EE-compliant Servlet container and is the official reference implementation for the Java Servlet and JavaServer Pages APIs. Tomcat also includes implementations of the JNDI and JMX APIs, but not the rest of the J2EE APIs, and is not, thus, an application server.
- 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 Netscape Enterprise Server (NES) servers.

This chapter has provided a basic introduction to Tomcat. Chapter 2 describes what Tomcat-served Web applications look like and what files comprise them.