THE EXPERT'S VOICE® IN JAVA

SECOND EDITION

Beginning JSP, JSF and Tomcat

Java Web Development

START BUILDING JAVA-BASED WEB APPLICATIONS NOW

Giulio Zambon

Apress®

Beginning JSP, JSF and Tomcat

Java Web Development

Giulio Zambon

Apress®

Beginning JSP, JSF and Tomcat

ISBN 978-1-4302-4623-7

ISBN 978-1-4302-4624-4 (eBook)

President and Publisher: Paul Manning
Lead Editor: Steve Anglin
Developmental Editor: Douglas Pundick, Ralph Moore
Technical Reviewer: Boris Minkin, Manuel Joran Elera
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editors: Katie Sullivan
Copy Editor: Michael Sandlin
Compositor: Bytheway Publishing Services
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code.

Contents at a Glance

Contents

About the Author

Giulio Zambon's first love was physics, but he decided to dedicate himself to software development more than 30 years ago: back when computers were still made of transistors and core memories, programs were punched on cards, and Fortran only had arithmetic IFs. Over the years, he learned a dozen computer languages and worked with all sorts of operating systems. His specific interests were in telecom and real-time systems, and he managed several projects to their successful completion.

In 2001 Giulio founded his own company offering computer telephony integration (CTI) services, and he used JSP and Tomcat exclusively to develop the web side of the service platform. Back in Australia after many years in Europe, he now dedicates himself to writing software to generate and solve numeric puzzles. His web site, http://zambon.com.au/, is written in JSP on his dedicated server, which, unsurprisingly, runs Tomcat!

About the Technical Reviewers

Boris Minkin is a senior technical architect at a major financial corporation. He has more than 20 years of experience working in various areas of information technology and financial services. Boris obtained his master's degree in information systems at Stevens Institute of Technology, New Jersey. His professional interests are in Internet technology, service-oriented architecture, enterprise application architecture, multi-platform distributed applications, cloud, distributed caching, Java, grid, and high performance computing. You can contact Boris at `bm@panix.com`.

Manuel Jordan Elera is an autodidactic developer and researcher who enjoys learning new technologies for his own experiments and creating new integrations. Manuel won the 2010 Springy Award-Community Champion. In his limited free time, he reads the Bible and composes music on his guitar. Manuel is a senior member in the Spring Community Forums known as dr_pompeii and a technical reviewer for important books about Spring Source projects, all published by Apress. Read more and contact him through his blog at http://manueljordan.wordpress.com and follow him on his Twitter account, @dr_pompei.

CHAPTER 1

Introducing JSP and Tomcat

Interactivity is what makes the Web really useful. By interacting with a remote server, you can find the information you need, keep in touch with your friends, or purchase something online. And every time you type something into a web form, an application "out there" interprets your request and prepares a web page to respond.

To understand JSP, you first need to have a clear idea of what happens when you ask your browser to view a web page, either by typing a URL into the address field of your browser or by clicking on a hyperlink. Figure 1-1 shows you how it works.

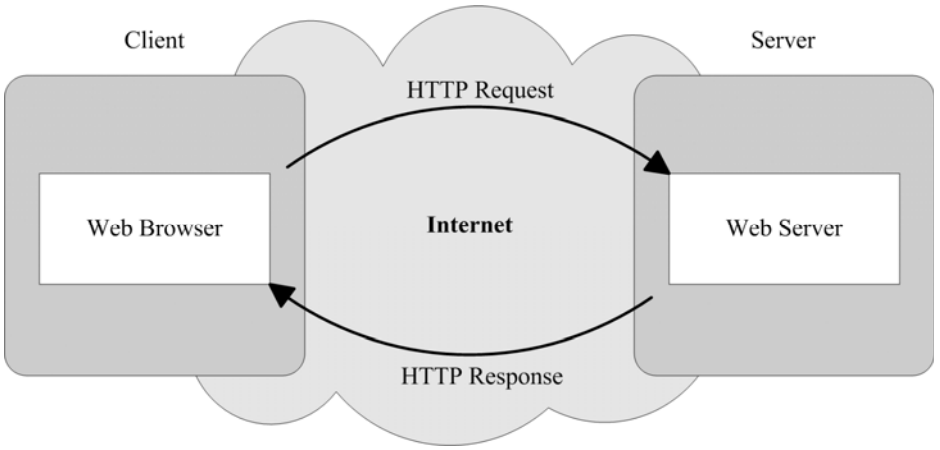

Figure 1-1. *Viewing a plain HTML page*

The following steps show what happens when you request your browser to view a static web page:

1. When you type an address such as `http://www.website.com/path/whatever.html` into the address field, your browser first resolves `www.website.com` (i.e., the name of the web server) into the corresponding Internet Protocol (IP) address, usually by asking the Domain Name Server provided by your Internet Service Provider (ISP). Then your browser sends an HTTP request to the newly found IP address to receive the content of the file identified by `/path/whatever.html`.

2. In reply, the web server sends an HTTP response containing a plain-text HTML page. Images and other non-textual components, such as sound and video clips, only appear in the page as references.

3. Your browser receives the response, interprets the HTML code contained in the page, requests the non-textual components from the server, and displays the lot.

JavaServer Pages (JSP) is a technology that helps you create such dynamically generated pages by converting script files into executable Java modules; JavaServer Faces (JSF) is a package that facilitates interactivity with the page viewers; and Tomcat is an application that can execute your code and act as a web server for your dynamic pages.

Everything you need to develop JSP/JSF web applications is available for free download from the Internet; but to install all the necessary packages and tools and obtain an integrated development environment, you need to proceed with care. There is nothing more annoying than having to deal with incorrectly installed software. When something doesn't work, the problem will always be difficult to find.

In this chapter, I'll introduce you to Java servlets and JSP, and I'll show you how they work together within Tomcat to generate dynamic web pages. But, first of all, I will guide you through the installation of Java and Tomcat: there wouldn't be much point in looking at code you can't execute on your PC, would there?

You'll have to install more packages as you progress. Do these installations correctly, and you will never need to second guess yourself. In total, you will need at least 300MB of disk space for Java and Tomcat alone and twice as much space to install the Eclipse development environment.

To run all the examples contained in this book, I used a PC with a 2.6GHz AMD Athlon 64x2 (nothing fancy, nowadays) with 1GB of memory and running Windows Vista SP2. Before performing any installation, I reformatted the hard disk and re-installed the OS from the original DVD. **I don't suggest for a moment that you do the same!** I did it for two opposite but equally important reasons: first, I didn't want existing stuff to interfere with the latest packages needed for web development; second, I didn't want to rely on anything already installed. I wanted to be sure to give you the full list of what you need.

At the time of this writing, the latest versions of all the packages you will need to install are:

Java: 1.7.0 update 3 (installation explained in this chapter)
Tomcat web server: 7.0.26 (installation also explained in this chapter)
Eclipse development environment: Indigo 3.7.2 (installation explained in Chapter 2)
MySQL database: 5.5.21.0 (installation explained in Chapter 6)
MySQL Java database connector (JDBC): 5.1.18 (installation also explained in Chapter 6)
JavaServer Faces: 2.1.7 (installation explained in Chapter 7)

I included Eclipse on the list because an integrated development environment is extremely useful for developing software. And MySQL is listed because any non-trivial web application is likely to need handling data.

Of course, after this book is published, there will most likely be newer releases of all the aforementioned packages. Nevertheless, you should be able to adapt my instructions to install the latest versions without any problem.

One last recommendation: to be sure that everything will work correctly, please follow the installation instructions to the letter. It will save you endless headaches.

'Nuff said. Here we go.

Installing Java

Nothing runs without Java, and you need two different Java packages: one is the runtime environment (JRE), which lets you execute Java, and the other is the Java Development Kit (JDK), which lets you compile Java sources into executable classes.

They are downloadable together from Oracle's web site. Here's what you need to do:

1. Go to the URL `http://www.oracle.com/technetwork/java/javase/downloads/index.html`.
2. Click on the big button marked "`Java Download`" (the latest version at the time of writing is `7u3`). This will take you to the page "Java SE Development Kit 7 Downloads."
3. Select "`Accept License Agreement`" and then click on the link `jdk-7u3-windows-i586.exe`.

   The actual link might refer to a version other than "`7u3,`" but you need to download either "`Windows x86 (32-bit)`" or "`Windows x64 (64-bit),`" according to type of processor of your PC. Although I am using a 64-bit PC, I have tested all the examples in this book with 32-bit packages because I didn't want to test everything twice.
4. Execute the file.
5. Accept the license agreement when requested and install everything.

At this point, you should have the folder `C:\Program Files\Java\` with two subfolders: `jdk1.7.0_03` and `jre7`, or the equivalent folders for the version you have downloaded.

In order to be able to compile Java from the command line, you need to add the JDK path to the `PATH` environment variable. From the Windows `Start` menu, select `Settings` ➤ `Control Panel` ➤ `System`. When the `System Properties` dialog opens, click on the "`Advanced system settings`" link that you find on the left-hand side and then on the `Advanced` tab. Finally, to reach the dialog that lets you modify the `PATH` variable, click on the "`Environment Variables`" button. You will see the double dialog window shown in Figure 1-2.

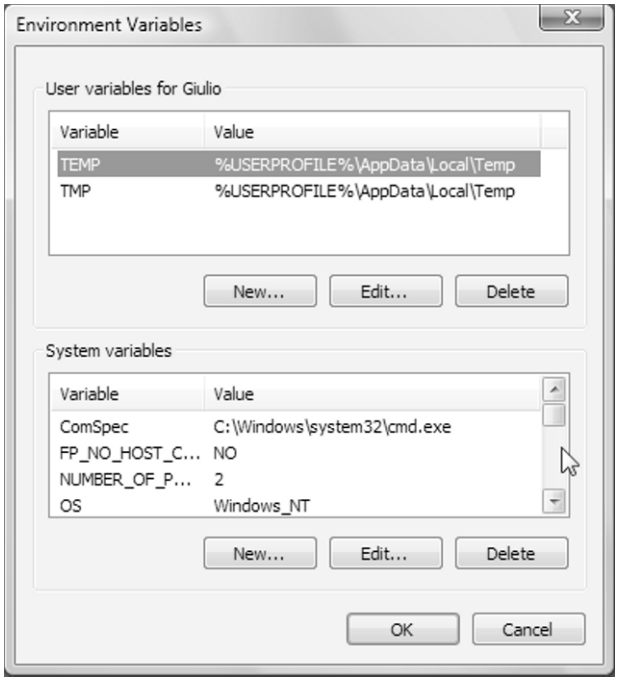

Figure 1-2. *The Environment Variables double dialog*

You might see a `PATH` variable on the top dialog, but what you need to do is scroll the bottom dialog by clicking on its sidebar until you see a variable named `Path`. Double-click it (or highlight it and click the "`Edit...`" button) and insert at the beginning of its value the text "`C:\Program Files\Java\jdk1.7.0_03\bin;`", as shown in Figure 1-3.

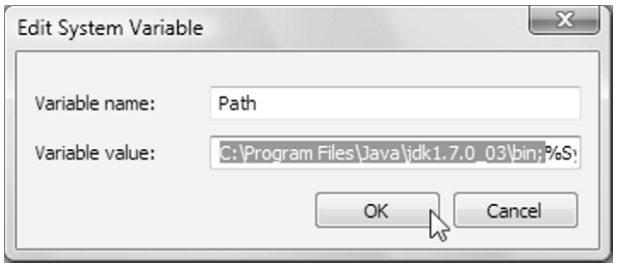

Figure 1-3. *Update the Path variable*

The semicolon at the end of the text is essential because it separates the new path from the existing ones. Do not insert additional spaces before or after.

Click on the "`OK`" button to save the changes. Then click this button another couple of times until the system dialog closes.

Java Test

To test the Java installation, you can use the little application shown in Listing 1-1.

Listing 1-1. *Exec_http.java*

```
/* Exec_http.java - Launches a web page
 *
 * Usage: Exec_http URL [arg1 [arg2 [...]]]
 * where URL is without "http://"
 *
 */
import java.io.*;
import java.net.*;
class Exec_http {
  public static void main(String[] vargs)
      throws java.net.MalformedURLException ,java.io.IOException
      {
    String  dest = "http://";

    if (vargs.length <= 0) {
      System.out.println("Usage: Exec_http page [args]");
      System.exit(1);
      }
    else {
      dest += vargs[0];
      for (int k = 1; k < vargs.length; k++) {
        dest += ((k == 1) ? "?" : "&") + vargs[k];
        }
      }
    System.out.println(dest);
    URL           url = new URL(dest);
    Object        obj = url.getContent();
    InputStream   resp = (InputStream)obj;
    byte[]        b = new byte[256];
    int           n = resp.read(b);
    while (n != -1) {
      System.out.print(new String(b, 0, n));
      n = resp.read(b);
      }
    }
  }
```

It lets you open a web page from the command line. Note that all the code described in this book is available for download from the Apress web site (`http://www.apress.com/9781430246237`). You don't need to retype it. You can find the examples in folders with the same names as the corresponding chapters. I will refer to the root directory of the software package associated with this book with the string `%SW_HOME%`.

Copy the file `%SW_HOME%\01 Getting Started\java\Exec_http.java` to a work directory. For simplicity, I use the desktop: but in my case, this makes sense because I use the computer exclusively to develop the examples used in this book.

Open a command-line window by clicking the Start button and selecting Programs ➤ Accessories ➤ Command Prompt. Then, after changing to your work directory, type "javac Exec_http.java" to compile the application. It should return the prompt without saying anything. If this happens, it means that you have correctly updated the Path system variable. If you want to know more about what the javac compiler is doing, type –verbose between javac and the name of the file.

You will see a file named Exec_http.class in your work directory.

Now, to run the application, type "java Exec_http" followed by the URL of the page you want to display. Any URL will do, but remember that the command-line accessory doesn't understand HTML. Therefore, if you display any commercial page, you will see a long stream of text filling the window.

To test the application, I placed on one of my web servers a one-line text file. Figure 1-4 shows what happened.

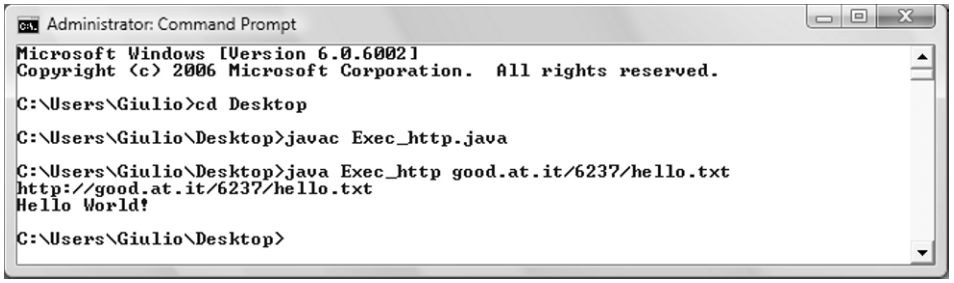

Figure 1-4. *Testing Java*

You are welcome to use hello.txt for your test. Hopefully, I will remember to keep it online!

Installing Tomcat

This is the Java web server, which is the servlet container that allows you to run JSP. If you have already installed an older version of Tomcat, you should remove it before installing a new version.

Tomcat listens to three communication ports of your PC (8005, 8009, and 8080). Before you install Tomcat, you should check whether some already-installed applications are listening to one or more of those ports. To do so, open a DOS window and type the command netstat /a. It will display a list of active connections in tabular form. The second column of the table will look like this:

```
Local Address
0.0.0.0:135
0.0.0.0:445
0.0.0.0:3306
```

The port numbers are the numbers after the colon. If you see one or more of the ports Tomcat uses, after installing Tomcat, you will have to change the ports it listens to, as explained in Chapter 10. There, you will also learn the purpose of those three ports.

Here's how to install Tomcat 7 correctly:

1. Go to the URL http://tomcat.apache.org/download-70.cgi. Immediately below the second heading ("Quick Navigation"), you will see four links: KEYS, 7.0.26, Browse, and Archives.

2. By clicking on 7.0.26, you will be taken toward the bottom of the same page to a heading with the same version number. Below the version heading, you will

see the subheading "Core". Below that, immediately above the next subheading, you will see three links arranged as follows: 32-bit/64-bit Windows Service Installer (pgp, md5).

3. Click on 32-bit/64-bit Windows Service Installer to download the file apache-tomcat-7.0.26.exe (8.2 MB).

4. Before launching the installer file, you have to check its integrity. To do so, you need a small utility to calculate its checksum. There are several freely available on the Internet. I downloaded WinMD5Free from http://www.winmd5.com/, and it worked for me, but this doesn't mean I consider it better than any other similar utility. It just happened to be the first one I saw. The program doesn't require any special installation: just unzip it and launch. When you open the Tomcat installer file, you will see a 32-digit hexadecimal number very much like this: 8ad7d25179168e74e3754391cdb24679.

5. Go back to the page from which you downloaded the Tomcat installer and click on the md5 link (the third one, and second within the parentheses). This will open a page containing a single line of text, like this:
8ad7d25179168e74e3754391cdb24679 *apache-tomcat-7.0.26.exe

   If the hex string is identical to that calculated by the checksum utility, you know that the version of Tomcat installer you have downloaded has not been corrupted or modified in any way.

6. Now that you have verified the correctness of the Tomcat installer, launch it.

7. After you've agreed to the terms of the license, you will then see the dialog shown in Figure 1-5. Click on the plus sign before the Tomcat item and select "Service" and "Native" as shown in the figure before clicking on the "Next >" button.

8. I chose to install Tomcat in the directory "C:\Program Files\Apache Software Foundation\Tomcat" instead of the default "Tomcat 7.0". This is because sometimes you might like to point to this directory (normally referred to as %CATALINA_HOME%) from within a program, and one day you might replace Tomcat 7.0 with Tomcat 8.0. By calling Tomcat's home directory "Tomcat" you are "safe" for years to come. You can also decide to leave the default. In general, by using the defaults, you are likely to encounter fewer problems, because the default settings of any applications are always tested best!

9. Next, the Tomcat installer will ask you to specify the connector port and UserID plus password for the administrator login. Leave the port set to 8080, because all the examples in this book refer to port 8080. If you want, you can always change it later to the HTTP standard port (which is 80). As UserID/Password, you might as well use your Windows user name and password. It is not critical.

10. Lastly, you will need to provide the path of a Java Runtime Environment. This is the path you saw when installing Java (see previous section). With the version of Java I installed, the correct path is C:\Program Files\Java\jre7.

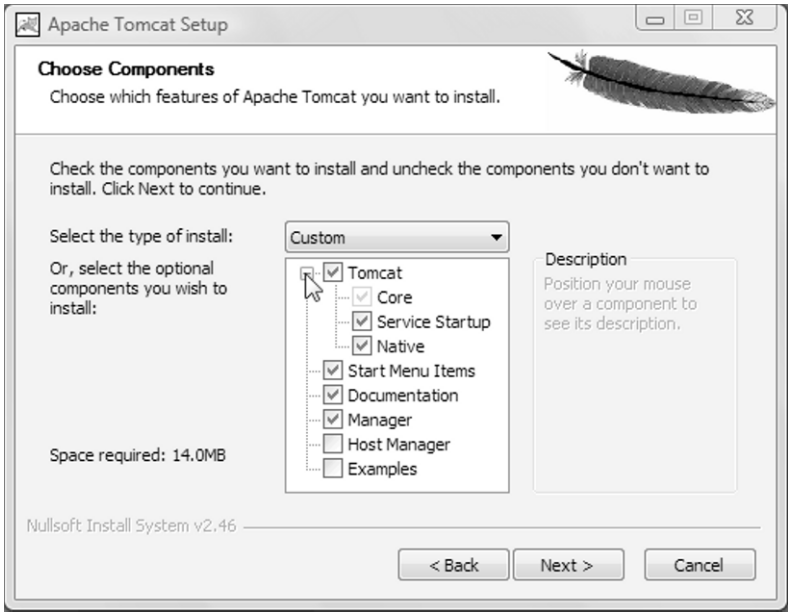

Figure 1-5. *Tomcat's Service and Native settings*

Tomcat runs as a Windows service. To start it and stop it, you can right-click the Apache Service Manager icon in the notification area of Windows' toolbar and select the corresponding operation. You can also achieve the same result by opening Windows' `Services` control panel (and right-clicking the Tomcat entry, as shown in Figure 1-6).

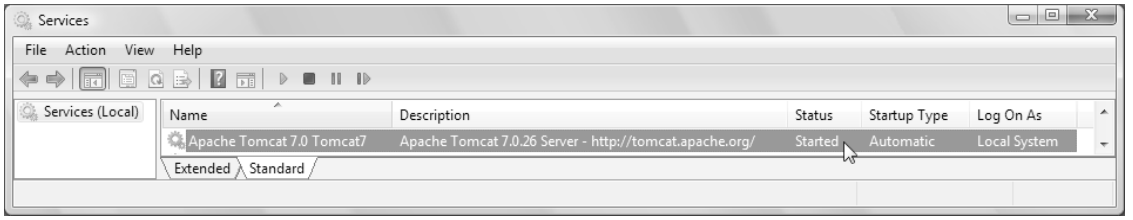

Figure 1-6. *Stopping and starting Tomcat from the Services control panel*

In any case, to go to the `Services` control panel, click on Windows' `Start` menu and select `Setting` ➤ `Control Panel` ➤ `Administrative Tools` ➤ `Services`. You will see dozens of entries, but when you reach the Tomcat services, its status should be "`Started`".

With Java and Tomcat in place, we can finally begin playing with JSP!

Simple Tomcat Test

To see that Tomcat is working properly, open a browser and type `localhost:8080`. You should see the page shown in Figure 1-7 (Firefox in the example).

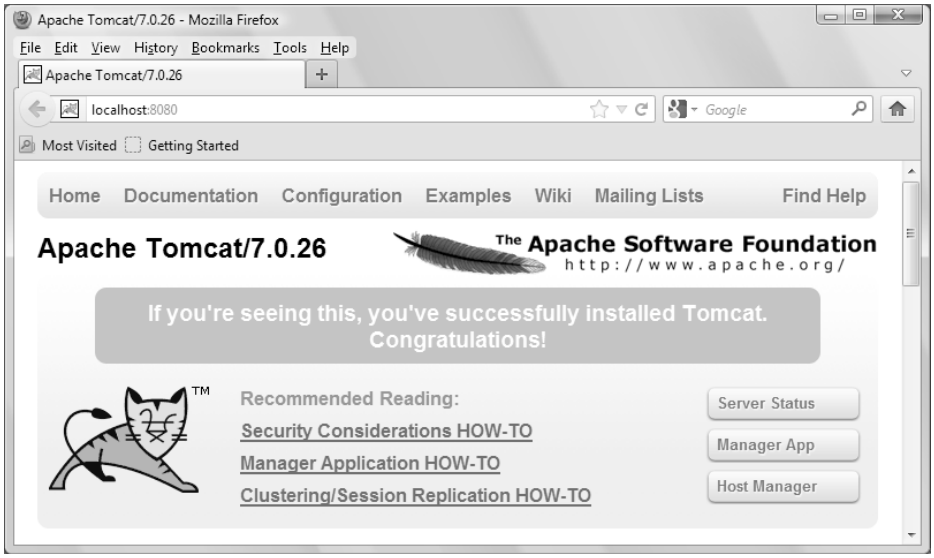

Figure 1-7. *The localhost home page*

Windows might state that it needs to block a startup program that requires permission. To solve this problem, turn off Windows' User Account Control by going into the `User Accounts` control panel and clicking on "`Turn User Account Control on or off`". This User Account Control can be a nuisance anyway, because it asked for authorization every time I worked with files in directories considered protected, including all directories in the "`Program Files`" folder.

At some point after rebooting, if you are running Vista, Windows' Program Compatibility Assistant might display a dialog stating that the Sun Java Scheduler, located at `C:\Program Files\Common Files\Java Update\jusched.exe`, is incompatible with your version of Windows. To get rid of this problem, you need to select "`Run`" after clicking on "`Start,`" type in the text field "`msconfig`" (without double quotes), and hit `Enter`. Select the "`Startup`" tab in the dialog, find the entry for the Java updater, and remove it.

What Is JSP?

JSP is a technology that lets you add dynamic content to web pages. In absence of JSP, to update the appearance or the content of plain static HTML pages, you always have to do it by hand. Even if all you want to do is change a date or a picture, you must edit the HTML file and type in your modifications. Nobody is going to do it for you, whereas with JSP, you can make the content dependent on many factors, including the time of the day, the information provided by the user, the user's history of interaction with your web site, and even the user's browser type. This capability is essential to provide online services in which you can tailor each response to the viewer who made the request, depending on the viewer's preferences and requirements. A crucial aspect of providing meaningful online services is for the system to be able to *remember* data associated with the service and its users. That's why databases play an essential role in dynamic web pages. But let's take it one step at a time.

HISTORY

Sun Microsystems introduced JSP in 1999. Developers quickly realized that additional tags would be useful, and the JSP Standard Tag Library (JSTL) was born. JSTL is a collection of custom tag libraries that encapsulates the functionality of many JSP standard applications, thereby eliminating repetitions and making the applications more compact. Together with JSTL also came the JSP Expression Language (EL).

In 2003, with the introduction of JSP 2.0, EL was incorporated into the JSP specification, making it available for custom components and template text, not just for JSTL, as was the case in the previous versions. Additionally, JSP 2.0 made it possible to create custom tag files, thereby perfecting the extendibility of the language.

In parallel to the evolution of JSP, several frameworks to develop web applications became available. In 2004, one of them, JavaServer Faces (JSF), focused on building user interfaces (UIs) and used JSP by default as the underlying scripting language. It provided an API, JSP custom tag libraries, and an expression language.

The Java Community Process (JCP), formed in 1998, released in May 2006 the Java Specification Request (JSR) 245 titled *JavaServer Pages 2.1*, which effectively aligned JSP and JSF technologies. In particular, JSP 2.1 included a Unified EL (UEL) that merged the two versions of EL defined in JSP 2.0 and JSF 1.2 (itself specified as JSR 252). Sun Microsystems includes JSP 2.1 in its Java Platform, Enterprise Edition 5 (Java EE 5), finalized in May 2006 as JSR 244.

The latest version of Java is 7 (specified in JSR 342 and released in July 2011). It includes JSP 2.2, Servlets 3.1 (JSR 340), EL 3.0 (JSR 341), and JSF 2.2 (JSR 344). Version 8 is expected in mid-2013. At the time of this writing, Java 7 is only available as part of the JSE (Java Standard Edition) platform. The latest version of Java released in the JEE (Java Enterprise Edition) platform is 6 (update 32).

The latest version of Tomcat (7.0), supports Servlets 3.0 and JSF 2.1.7.

Viewing a JSP Page

With JSP, the web page doesn't actually exist on the server. As you can see in Figure 1-8, the server creates it fresh when responding to each request.

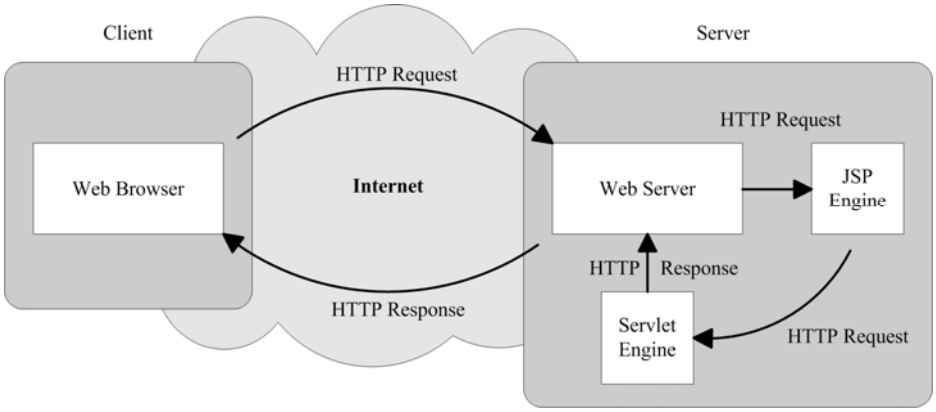

Figure 1-8. *Viewing a JSP page*

The following steps explain how the web server creates the web page:

1. As with a normal page, your browser sends an HTTP request to the web server. This doesn't change with JSP, although the URL probably ends in `.jsp` instead of `.html` or `.htm`.

2. The web server is not a normal server, but rather a Java server, with the extensions necessary to identify and handle Java servlets. The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine.

3. The JSP engine loads the JSP page from disk and converts it into a Java servlet. From this point on, this servlet is indistinguishable from any other servlet developed directly in Java rather than JSP, although the automatically generated Java code of a JSP servlet is not always easy to read, and you should never modify it by hand.

4. The JSP engine compiles the servlet into an executable class and forwards the original request to another part of the web server called the *servlet engine*. Note that the JSP engine only converts the JSP page to Java and recompiles the servlet if it finds that the JSP page has changed since the last request. This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.

5. The servlet engine loads the servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.

6. The web server forwards the HTTP response to your browser.

7. Your web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page. In fact, static and dynamic web pages are in the same format.

You might ask, "Why do you say that with JSP, the page is created fresh for each request, if the server only converts and compiles the JSP source if you have updated it since the previous request?"

What reaches your browser is the output generated by the servlet (that is, by the converted and compiled JSP page), not the JSP page itself. The same servlet produces different outputs depending on the parameters of the HTTP request and other factors. For example, suppose you're browsing the products offered by an online shop. When you click on the image of a product, your browser generates an HTTP request with the product code as a parameter. As a result, the servlet generates an HTML page with the description of that product. The server doesn't need to recompile the servlet for each product code.

The servlet queries a database containing the details of all the products, obtains the description of the product you're interested in, and formats an HTML page with that data. This is what dynamic HTML is all about!

Plain HTML is not capable of interrogating a database, but Java is, and JSP gives you the means of including snippets of Java inside an HTML page.

Hello World!

A small example of JSP will give you a more practical idea of how JSP works. Let's start once more from HTML. Listing 1-2 shows you a plain HTML page to display "Hello World!" in your browser's window.

Listing 1-2. *hello.html*

```
<html>
<head><title>Hello World static HTML</title></head>
<body>
Hello World!
</body>
</html>
```

Create the folder `%CATALINA_HOME%\webapps\ROOT\tests\` and store in it `hello.html`. Then type the following URL in your browser to see the web page:

```
http://localhost:8080/tests/hello.html
```

Normally, to ask your browser to check that the syntax of the page conforms to the XHTML standard of the World Wide Web Consortium (W3C), you would have to start the page with the following lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

You'd also have to replace

```
<html>
```

with

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

However, for this simple example, I prefer to keep the code to what's essential. Figure 1-9 shows you how this page will appear in your browser.

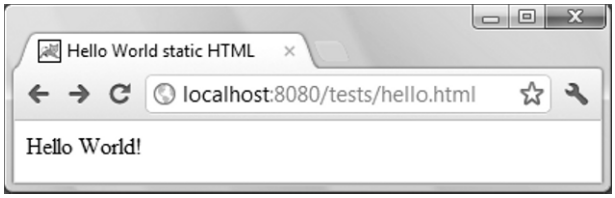

Figure 1-9. *"Hello World!" in plain HTML*

If you direct your browser to show the page source, not surprisingly, you'll see exactly what's shown in Listing 1-2. To obtain the same result with a JSP page, you only need to insert a JSP directive before the first line, as shown in Listing 1-3, and change the file extension from `.html` to `.jsp`.

Listing 1-3. *"Hello World!" in a Boring JSP Page*

```
<%@page language="java" contentType="text/html"%>
<html>
<head><title>Hello World not-so-dynamic HTML</title></head>
<body>
Hello World!
</body>
</html>
```

Obviously, there isn't much point in using JSP for such a simple page. It only pays to use JSP if you include dynamic content. Check out Listing 1-4 for something more juicy.

Listing 1-4. *hello.jsp*

```
<%@page language="java" contentType="text/html"%>
<html>
<head><title>Hello World dynamic HTML</title></head>
<body>
Hello World!
<%
  out.println("<br/>Your IP address is " + request.getRemoteAddr());

  String userAgent = request.getHeader("user-agent");
  String browser = "unknown";

  out.print("<br/>and your browser is ");
  if (userAgent != null) {
    if (userAgent.indexOf("MSIE") > -1) {
      browser = "MS Internet Explorer";
      }
    else if (userAgent.indexOf("Firefox") > -1) {
      browser = "Mozilla Firefox";
      }
    else if (userAgent.indexOf("Opera") > -1) {
      browser = "Opera";
      }
    else if (userAgent.indexOf("Chrome") > -1) {
```

```
      browser = "Google Chrome";
      }
    else if (userAgent.indexOf("Safari") > -1) {
      browser = "Apple Safari";
      }
    }
  out.println(browser);
  %>
</body>
</html>
```

As with `hello.html`, you can view `hello.jsp` by placing it in Tomcat's `ROOT\tests` folder.

The code within the `<% ... %>` pair is a scriptlet written in Java. When Tomcat's JSP engine interprets this module, it creates a Java servlet like that shown in Listing 1-5 (with some indentation and empty lines removed).

Listing 1-5. *Java Code from the "Hello World!" JSP Page*

```
out.write("\r\n");
out.write("<html>\r\n");
out.write("<head><title>Hello World dynamic HTML</title></head>\r\n");
out.write("<body>\r\n");
out.write("Hello World!\r\n");
out.write('\r');
out.write('\n');
out.println("<br/>Your IP address is " + request.getRemoteAddr());
String userAgent = request.getHeader("user-agent");
String browser = "unknown";
out.print("<br/>and your browser is ");
if (userAgent != null) {
  if (userAgent.indexOf("MSIE") > -1) {
    browser = "MS Internet Explorer";
    }
  else if (userAgent.indexOf("Firefox") > -1) {
    browser = "Mozilla Firefox";
    }
  else if (userAgent.indexOf("Opera") > -1) {
    browser = "Opera";
    }
  else if (userAgent.indexOf("Chrome") > -1) {
    browser = "Google Chrome";
    }
  else if (userAgent.indexOf("Safari") > -1) {
    browser = "Apple Safari";
    }
  }
out.println(browser);
out.write("\r\n");
out.write("</body>\r\n");
out.write("</html>\r\n");
```