Supplement IV.E: Tutorial for Tomcat

For Introduction to Java Programming
By Y. Daniel Liang

This supplement covers the following topics:

- Obtaining and Installing Tomcat
- Starting and Stopping Tomcat
- Choosing a Different Port
- Compiling Servlets
- Mapping a Servlet to a URL
- Running Servlets
- Running JSP
- Deploying Web Application Using WAR files
- Setting a Secured Web Server

0 Introduction

Tomcat, developed by Apache (**www.apache.org**), is a standard reference implementation for Java servlets and JSP. It can be used standalone as a Web server or be plugged into a Web server like Apache, Netscape Enterprise Server, or Microsoft Internet Information Server. There are many versions of Tomcat. This tutorial uses Tomcat 5.5.9 as an example. The tutorial should also apply to all later versions of Tomcat.

1 Obtaining and Installing Tomcat

You can download Tomcat from http://tomcat.apache.org/download-60.cgi. On this page, select zip under Core to download a zip file (i.e., apache-tomcat-6.0.29.zip). You can use FilZip to extract it into c:\. FilZip is a free compression/decompression utility, which can be downloaded from http://www.filzip.com.

2 Starting and Stopping Tomcat

Before running the servlet, you need to start the Tomcat servlet engine. To start Tomcat, you have to first set the JAVA_HOME environment variable to the JDK home directory using the following command. (Please note no space before or after the = sign in the following line.)

```
set JAVA_HOME=c:\Program Files\java\jdk1.6.0_24
```

The JDK home directory is where your JDK is stored. On my computer, it is c:\Program Files\jdk1.6.0_24. You may have a

different directory. You can now start Tomcat using the command **startup** from c:\jakarta-tomcat-5.5.9\bin as follows:

```
c:\jakarta-tomcat-5.5.9\bin>startup
```

NOTE: If you are using Tomcat 7.0.2, the default directory is c:\apache-tomcat-7.0.2. We use Tomcat 5.5.9 as example in this tutorial.

NOTE: By default, Tomcat runs on part 8080. An error occurs if this port is currently being used. You can change the port number in c:\jakarta-tomcat-5.5.9\conf\server.xml, as discussed in the next section.

NOTE: To terminate Tomcat, use the **shutdown** command from c:\jakarta-tomcat-5.5.9\bin.

To prove that Tomcat is running, type the URL http://localhost:8080 from a Web browser, as shown in Figure 1.

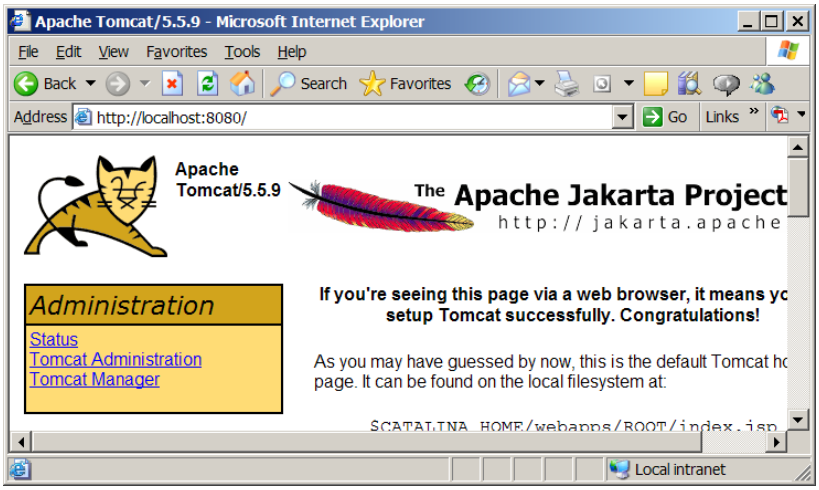

Figure 1
The default Tomcat page is displayed.

3 Choosing a Different Port (Optional)

By default, Tomcat runs on part 8080. You can change it to a different port. To do so, open c:\jakarta-tomcat-5.5.9\conf\server.xml using a text editor such as NotePad. Search for 8080 and change it to a desired port number such as 8100 in the following context.

```
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
    port="8080" minProcessors="5" maxProcessors="75"
    enableLookups="true" redirectPort="8443"
    acceptCount="100" debug="0" connectionTimeout="20000"
    useURIValidationHack="false" disableUploadTimeout="true" />
```

4 Creating a Servlet

A servlet resembles an applet in some extent. Every Java applet is a subclass of the Applet class. You need to override appropriate methods in the Applet class to implement the applet. Every servlet is a subclass of the HttpServlet class. You need to override appropriate methods in the HttpServlet class to implement the servlet. Listed below is a simple servlet example that generates a response in HTML using the doGet method.

```
import javax.servlet.*;
import javax.servlet.http.*;

public class FirstServlet extends HttpServlet {
  /** Handle the HTTP <code>GET</code> method.
   * @param request servlet request
   * @param response servlet response
   */
  protected void doGet(HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, java.io.IOException {
    response.setContentType("text/html");
    java.io.PrintWriter out = response.getWriter();
    // output your page here
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("Hello, Java Servlets");
    out.println("</body>");
    out.println("</html>");
    out.close();
  }
}
```

5 Compiling Servlets

Suppose you have installed Tomcat 5.5.9 at c:\jakarta-tomcat-5.5.9. To compile FirstServlet.java, you need to add c:\jakarta-tomcat-5.5.9\common\lib\servlet-api.jar to the classpath from DOS prompt as follows:

```
set classpath=%classpath%;c:\jakarta-tomcat-5.5.9\common\lib\servlet-api.jar
```

servlet.jar contains the classes and interfaces to support servlets. Use the following command to compile the servlet:

```
javac FirstServlet.java
```

Copy the resultant .class file into c:\jakarta-tomcat-5.5.9\webapps***liangweb***\WEB-INF\classes so it can be found at runtime.

> TIP: You can compile FirstServlet directly into the target directory by using the –d option in the javac command as follows:

```
javac FirstServlet.java –d targetdirectory
```

6 Mapping a Servlet to a URL

Before you can run a servlet in Tomcat 5.5.9, you have to first create the web.xml file with a servlet entry and a mapping entry. This file is located in C:\jakarta-tomcat-5.5.9\webapps\liangweb\WEB-INF\web.xml. If the file already exists, insert a servlet entry and a mapping entry into web.xml for the servlet.

The servlet entry declares an internal servlet name for a servlet class using the following syntax:

```
<servlet>
   <servlet-name>Internal Name</servlet-name>
   <servlet-class>servlet class name</servlet-class>
</servlet>
```

The map entry maps an internal servlet name with a URL using the following syntax:

```
<servlet-mapping>
   <servlet-name>Internal Name</servlet-name>
   <url-pattern>URL</url-pattern>
</servlet-mapping>
```

For example, before running FirstServlet.class, you may insert the following lines to the web.xml file:

```
<web-app>
  <servlet>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-class>FirstServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>FirstServlet</servlet-name>
    <url-pattern>/FirstServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

NOTE

<Side Remark: download web.xml>

For your convenience, I have created the web.xml that contains the decriptions for running all servlets in this chapter. You can download it from www.cs.armstrong.edu/liang/intro6e/supplement/web.xml.

TIP

<side remark: Web development tools>

You can use an IDE such as NetBeans, Eclipse, and JBuilder to simplify development of Web applications. The tool can automatically create the directories and files. For more information, see the tutorials on NetBeans, Eclipse, and JBuilder on the Companion Website.

7 Running Servlets

To run the servlet, start a Web browser and type ***http://localhost:8080/liangweb/FirstServlet*** in the URL, as shown in Figure 2.

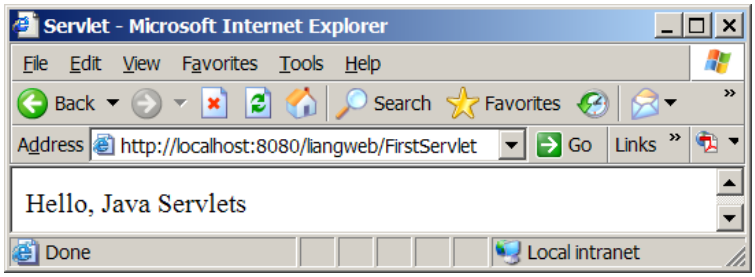

Figure 2

You can request a servlet from a Web browser.

NOTE: You can use the servlet from anywhere on the Internet if your Tomcat is running on a host machine on the Internet. Suppose the host name is liang.armstrong.edu; use the URL ***http://liang.armstrong.edu:8080/liangweb/FirstServlet*** to test the servlet.

NOTE: If you have modified the servlet, you need to shut down and restart Tomcat.

8 Running JSP

To run a JSP (e.g., Factorial.jsp), store it in webapps/liangweb, start a Web browser and type ***http://localhost:8080/liangweb/Factorial.jsp*** in the URL, as shown in Figure 3.

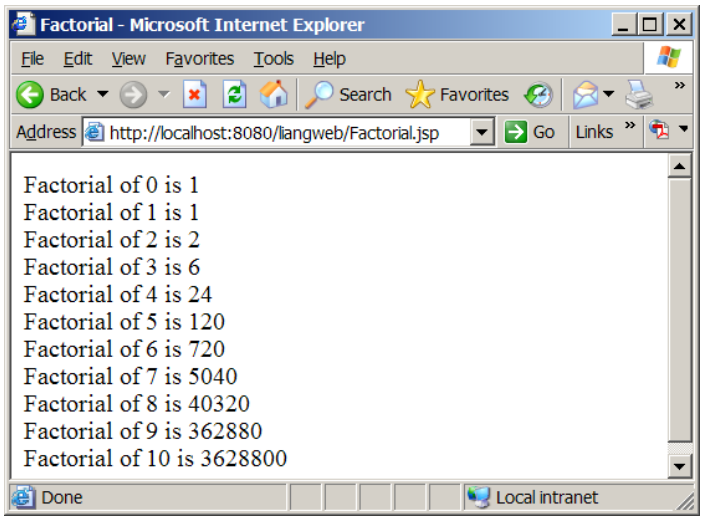

Figure 3
You can request a JSP from a Web browser.

9 Deploying Web Application Using WAR Files

Servlets, JSPs and their supporting files are placed in the appropriate subdirectories under the webapps directory in Tomcat. For convenience, you can create all the files under the webapps directory into one compressed file. This file is known as the *WAR* (Web Application Archive) and ends with the .war file extension. You can deploy a Web application by placing a WAR file in the webapps directory. When a Web server begins execution, it extracts the WAR file's contents into the appropriate webapps subdirectories.

9.1 Creating WAR Files

You can create a WAR file using a tool or the jar command. If you use NetBeans, JBuilder, or Eclipse, a WAR file is automatically created every time you build the Web project. (e.g., you can find this file under the dist node in a Web project). To create a WAR file for the examples in Tmocat, change the directory to c:\jakarta-tomcat-5.5.9\webapps\liangweb, and type the following command:

```
jar –cvf liangweb.war .
```

A WAR file named liangweb.war is now created for all files and directories under \webapps\liangweb. The file is in \webapps\liangweb\liangweb.war.

9.2 Deploying WAR Files

You can now deploy the WAR file to a new machine. Follow the steps below to test liangweb.war:

1. Place liangweb.war to \webapps\liangweb.war under your

Tomcat home directory.
2. Delete the entire directory of liangweb under webapps if it exists (this is necessary to enable new contents to be created).
3. Start Tomcat (if it is already running, you need to shut it down first.) Tomcat automatically extracts the contents in liangweb.war into the appropriate directories. You will see a new directory named liangweb created under the webapps directory.
4. Enter URL http://localhost:8080/liangweb/FirstServlet to test the servlet.

10 Setting a Secured Server

You can set the Tomcat server to activate basic HTTP authentication. You need to specify a security realm, which is a term that defines authorized users. There are three types of realms: *Memory*, *Database*, and *JNDI*. This section introduces how to set Memory and Database realms.

10.1 Setting a Secured Server Using Memory Realm

To set a Memory realm, do the following:

1. Inside the <Engine> ... </Engine> tag in conf/server.xml, insert the following line:

```
<Realm className="org.apache.catalina.realm.MemoryRealm" />
```

TIP: The server.xml file already has this line commented. Simply uncomment it to set a Memory realm.

2. Edit conf/tomcat-users.xml to add user name, password, and the role for each user. For example, the following content defines two users with the role named test.

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="test"/>
  <user username="john" password="cat" roles="test"/>
  <user username="scott" password="tiger" roles="test"/>
</tomcat-users>
```

NOTE: The user authentication information is defined in the file tomcat-users.xml. This information is loaded to an object stored in the memory upon Tomcat startup. That is why it is called the Memory realm.

3. Uncomment the following lines in conf/web.xml and modify the <role-name> field to test. This is the role used for the users in the Memory realm defined in Step 2.

```
...
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>
        Protected Site
      </web-resource-name>
      <!-- This would protect the entire site -->
      <url-pattern> /* </url-pattern>
      <!-- If you list http methods,
            only those methods are protected -->
      <http-method> DELETE </http-method>
      <http-method> GET </http-method>
      <http-method> POST </http-method>
      <http-method> PUT </http-method>
    </web-resource-collection>
    <auth-constraint>
      <!-- Roles that have access -->
      <role-name> test </role-name>
    </auth-constraint>
  </security-constraint>

  <!-- BASIC authentication -->
  <login-config>
    <auth-method> BASIC </auth-method>
    <realm-name> Example Basic Authentication </realm-name>
  </login-config>

  <!-- Define security roles -->
  <security-role>
    <description> Test role </description>
    <role-name> test </role-name>
  </security-role>
..
```

4. Start Tomcat.

5. Type http://localhost:8080/liangweb/FirstServlet from a Web browser, you will be prompted to enter user name and password, as shown in Figure 4.

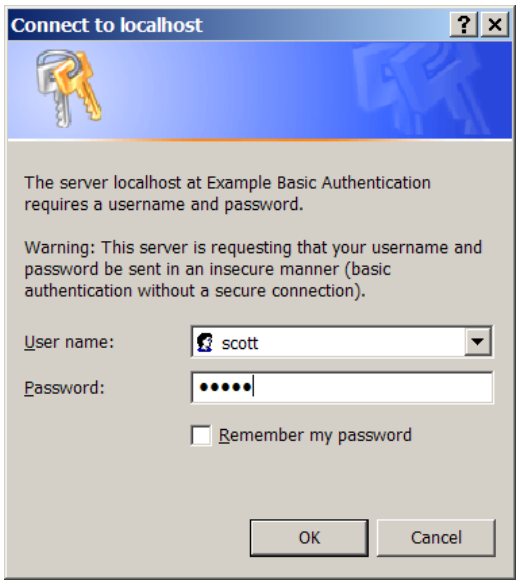

Figure 4
Tomcat supports basic authentication using the Memory realm.

10.2 Setting a Secured Server Using Database Realm

To set a Database realm, do the following:

1. Inside the <Engine> ... </Engine> tag in conf/server.xml, insert the following lines:

```
<Realm className="org.apache.catalina.realm.JDBCRealm"
  driverName="oracle.jdbc.driver.OracleDriver"
  connectionURL="jdbc:oracle:thin:@liang.armstrong.edu:1521:orcl"
  connectionName="scott" connectionPassword="tiger"
  userTable="Users" userNameCol="userName" userCredCol="user_pass"
  userRoleTable="UserRoles" roleNameCol="roleName" />
```

- The className is always the same.
- The driverName specifies a fully qualified JDBC driver name (e.g., oracle.jdbc.driver.OracleDriver for Oracle database). The connectionURL specifies the database URL. Note that the JAR file for the driver should be placed in the /common/lib directory.
- The connectionName is the database user name in which authorized Tomcat users are stored.
- The connectionPassword specifies the password for the database user.
- The userTable specifies the table where the authorized Tomcat user information is stored.
- The userNameCol specifies the column name in the userTable table for storing the authorized Tomcat user names.
- The userCredCol specifies the column name in the

userTable table for storing the authorized Tomcat user password.

- The roleNameCol specifies the column name in the userCredCol table for storing the role names.

2. Create the Users table and UserRoles table tables in the database as follows:

```
create table Users (
  userName varchar(15) not null primary key,
  userPass varchar(15) not null
);

create table UserRoles (
  userName varchar(15) not null,
  roleName varchar(15) not null,
  primary key (userName, roleName)
);
```

Authorize two users named student1 and student2 with password pstudent1 and pstudent2. Store the information in the Users table, as follows:

```
insert into Users values ('student1', 'pstudent1');
insert into Users values ('student2', 'pstudent2');
```

Assign student1 and studnet2 to the role named test and store the information in the UserRoles table, as follows:

```
insert into UserRoles values ('student1', 'test');
insert into UserRoles values ('student2', 'test');
```

3. Uncomment the following lines in conf/web.xml and modify the <role-name> field to test. This is the role used for the users in the Memory realm defined in Step 2.

```
...
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>
        Protected Site
      </web-resource-name>
      <!-- This would protect the entire site -->
      <url-pattern> /* </url-pattern>
      <!-- If you list http methods,
            only those methods are protected -->
      <http-method> DELETE </http-method>
      <http-method> GET </http-method>
      <http-method> POST </http-method>
      <http-method> PUT </http-method>
    </web-resource-collection>
    <auth-constraint>
```

```
      <!-- Roles that have access -->
      <role-name> test </role-name>
    </auth-constraint>
  </security-constraint>

  <!-- BASIC authentication -->
  <login-config>
    <auth-method> BASIC </auth-method>
    <realm-name> Example Basic Authentication </realm-name>
  </login-config>

  <!-- Define security roles -->
  <security-role>
    <description> Test role </description>
    <role-name> test </role-name>
  </security-role>
..
```

4. Start Tomcat.

5. Type http://localhost:8080/liangweb/FirstServlet from a Web browser, you will be prompted to enter user name and password, as shown in Figure 4.

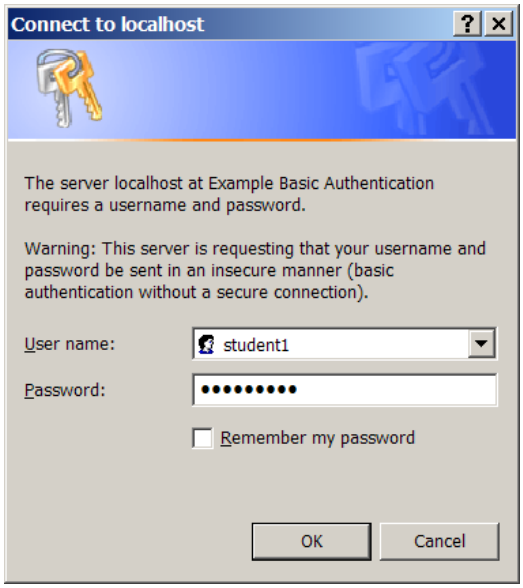

Figure 5
Tomcat supports basic authentication using the Database realm.