Université Grenoble Alpes POLYTECH

Polytech'Grenoble

TP- Installing, launching & managing an Apache Tomcat HTTP server

Polytech/INFO4
F. Boyer, O. Gruber

Organisation	Binome
Evaluation	Code-based
Advised time	2h per binome
Tools	JDK >=1.7, IDE (eclipse or equivalent)
Contact	Fabienne.Boyer@imag.dfr, Olivier.Gruber@imag.fr

1- Installing the Apache server

The *Apache* HTTP Server Project is an open-source HTTP server developped in Java. Among the different components that are provided by the Apache project, the *Apache Tomcat* server allows to deploy and run Web applications serving both static and dynamic pages relying on servlets. The objective of the TP is to learn to install and use an *Apache Tomcat* server, through deploying simple Web applications.

In order to install an Apache Tomcat server, please follow these steps:

1. Download Tomcat *core binary distribution* version 10.1.7 from http://tomcat.apache.org/
2. Choose an installation path without any spaces in the directory names (*~/**your-path**)*
3. Then uncompress the downloaded file there (*~/**your-path/apache-tomcat**-10.1.7)*
4. Open a shell and define the *CATALINA_HOME* environment variable to point to your installation:
   - Unix/tcsh: *setenv CATALINA_HOME ~/your-path/apache-tomcat-10.1.7*
   - Unix/bash: *export CATALINA_HOME=~/your-path/apache-tomcat-10.1.7*
   - Windows: *set CATALINA_HOME=~/your-path/apache-tomcat-10.1.7*

   Note: to check the shell you are using, on Linux, you can use the command ***echo $SHELL***

5. Make sure that the ***JAVA_HOME*** environment variable is defined (check it with: *echo $JAVA_HOME*); otherwise, define it in such a way that it points to the *sdk* installation directory of Java (often placed under /usr/lib/jvm on Linux, under /Library/Java/JavaVirtualMachines on Mac os).

   ex: *export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk-11.0.8.jdk/Contents/Home*

6. [IMPORTANT] Execute the two commands *ls $CATALINA_HOME* and *ls $JAVA_HOME* to check your environment variables.

2- Starting the *Apache Tomcat Servlet* server

In order to start the servlet server, run the following script:

- *Linux:* *$CATALINA_HOME/bin/startup.sh*
- *Windows:* *$CATALINA_HOME/bin/startup.bat*

Nota Bene: do not forget to make the scripts executable if required (*chmod +x $CATALINA_HOME/bin/**).

Make sure that you see the message: “Tomcat started”. If you do not see this message, something went wrong, you server is likely not started.

Once the Apache Tomcat Servlet server is started, it will accept HTTP connections on the port specified in the configuration file *$CATALINA_HOME/conf/server.xml* . Look for the value of "*Connector port*" value. The default port is *8080*. It is set to **8080** and not **80** so as not to interfere with any other web server that may be running on your machine.

3- Stopping the *Apache Tomcat Servlet* server

In order to stop the Apache Tomcat Servlet server, please run the following script:

- *Linux:* *$CATALINA_HOME/bin/shutdown.sh*
- *Windows:* *$CATALINA_HOME/bin/shutdown.bat*

4- Running your first Web application

Please follow these steps to run a first web application:

- Start your Apache Tomcat server
- Open a web browser
- Access the URL http://localhost:8080/**,** that will print the welcome page of the Apache Tomcat server. From this welcome page, you can access several (static) pages.
- We will focus on the servlet documentation: follow the ***Example*** link in the **Developer Quick Start** part, then follow the ***Servlet example*** link.

As you can see, there are several examples of servlet programs.

- Start with the ***HelloWorld*** example, run it through the link ***Execute***.
- Then you can have a look at the link ***Source*** that gives a simplified view of the source code of the servlet.
- The **complete source code for the servlet is under** *$CATALINA_HOME/webapps/examples/WEB-INF/classes/HelloWorldExample.java*.

  You will see that the complete code is a little bit more complex than the simplified form given through the *source* link. For instance, the *doGet* method starts by invoking a *ResourceBundle* object, that allows to manage locale-specific data**. In the following of the TP, you may bypass such code lines. We are not interested in writing locale-independent code so far, we just want to experiment with the basics of servlets.

  ***From Wikipedia*: in computing, a **locale** is a set of parameters that defines the user's language, region and any special variant preferences that the user wants to see in their user interface. Usually a locale identifier consists of at least a language code and a country/region code.

5. Looking at Servlets in more details

Go to the directory *$CATALINA_HOME/webapps/* that contains one sub-directory per web application (knowing that *ROOT* corresponds to the default application). Examine the *examples* directory content to understand how a web application is organized:

- **/WEB-INF/web.xml** – this XML file is the *Web Application Deployment Descriptor* for your application. It describes the servlets and other components that make up your application. Have a look at the <servlet> and <servlet-mapping> definitions.

  You will see that the <servlet> tag gives the mapping between a servlets names and their corresponding servlet classes, while the <servlet-mapping> tag gives the mapping between a *url* and the name of servlet that should be invoked.

- **/WEB-INF/classes/** - contains Java class files including both Servlet and non-Servlet classes. If your classes are organized into Java packages, you must reflect this in the directory hierarchy. For example, a Java class named `com.mycompany.mypkg.MyServlet` would need to be stored in a file placed in `/WEB-INF/classes/com/mycompany/mypkg/MyServlet.class`.
- **/WEB-INF/classes/** also contains the sources of Java classes, but this is specific to the current tutorial, a choice made for simplicity.
- **WEB-INF/lib/** - contains JAR files defining Java classes (and associated resources), such as third-party libraries or JDBC drivers.
- ***.html, etc.** - contains HTML pages, along with other resource files that may be transferred to the client browser (such as images or text files)

Verify that any servlet can be accessed through an URL of the form below, where *url-servlet* is specified in the *url-pattern* field of the *web.xml* file: http://host:port/<application-name>/<url-servlet>

Especially, check that the URL below refers to the ***examples*** application, and executes the servlet associated to the local path *servlets/servlet/HelloWorldExample* in the application directory.

http://localhost:8080/examples/servlets/servlet/HelloWorldExample

6. Defining a new servlet

You will now define a new servlet in the **examples** application. Start from the given HelloWorld example and follow these steps:

1. In the WEB-INF/classes directory, copy the given *HelloWorldExample.java* in a new file named *MyHelloWorldExample.java*.
2. Declare the new servlet in the application configuration file (*web.xml)*
3. Compile your newly defined servlet through the *javac* command. The servlet API is in the Java Archive *servlet-api.jar* in the directory *$CATALINA_HOME/lib.*
4. Try to run your newly defined servlet, with the following URL:

   http://localhost:8080/examples/servlets/servlet/MyHelloWorldExample

   It does not work? It may be normal. When you add new servlets, or when you modify the code of a servlet that has already been loaded by the Tomcat server, you may have to shutdown and restart the server to take into account your changes. So shutdown and restart your server, and check that the previous url is ok.

5. Now modify the class *MyHelloWorldExample* such that the servlet receives a GET request with two parameters: a name and a surname. Return a web page that displays "Hello world <name> <surname>".
6. Try to run your newly defined servlet, with the following URL:

   http://localhost:8080/examples/servlets/servlet/MyHelloWorldExample?name=bob&surname=marley

7. Session management

We will now consider session management, through the notions of *sessions* and *cookies*.

Session Example:

- Execute the *SessionExample* servlet from the *Examples* dir.
- Shutdown the Servlet server (while keeping the web browser running) and then restart the server and reload the *SessionExample* servlet. What is the produced result?
- Have a look at the file context.xml under the directory apache-tomcat-10.7.1/conf, you can setup session persistence there.
- Understand the source of the *SessionExample* servlet. For you to know, the *URL.encode()* is a utility method that manages the particular case where a browser does not accept cookies, which makes *session tracking* with cookies impossible. URL rewriting is then a solution that automatically embeds the session-identifier within the URL. Additionally, the *HTMLFilter* class is a utility class that performs some filtering on an URL to eliminate sensitive characters. Do not spend much time on this aspect, it is secondary.
- Modify your *MyHelloWorldExample* servlet to integrate in the response the number of times the servlet has been invoked from the same client.
- Run the newly defined servlet.

Cookies:

- Execute the *CookieExample* servlet from the *Examples* dir.
- Understand the Java source of the *CookieExample* servlet.
- Shutdown the Servlet server (while keeping the web browser running) and then restart the server and reload the *CookieExample* servlet.
  - What is the produced result?
  - How do you explain it?
- Modify your *MyHelloWorldExample* servlet to integrate in the response the number of times the servlet has been invoked from the same client, but this time use cookies instead of sessions.
- Run the newly defined servlet.

8. Small *WhoBringsWhat* application [OPTIONAL]

You are now ready to develop your own simple servlet-based application.

We suggest that you develop an application to manage *who brings what* at a party. To this end, you may first program a servlet that returns a welcome page allowing to get the party name. Then you may program a second servlet that, being given a party name as argument, allows an invitee to indicate its name and what (s)he intends to bring to the party. This servlet may also allow an invitee to see what others intend to bring to the party.

To manage the persistency of the party's data, you may simply use a local file, either a text file or a binary file, according to the format you wish to adopt to save your data.