Maven Example

With thanks to the Java Tutorial Network

Source: https://javatutorial.net/java-servlet-example

Updated by Dr. Spiegel

Java Servlet Example

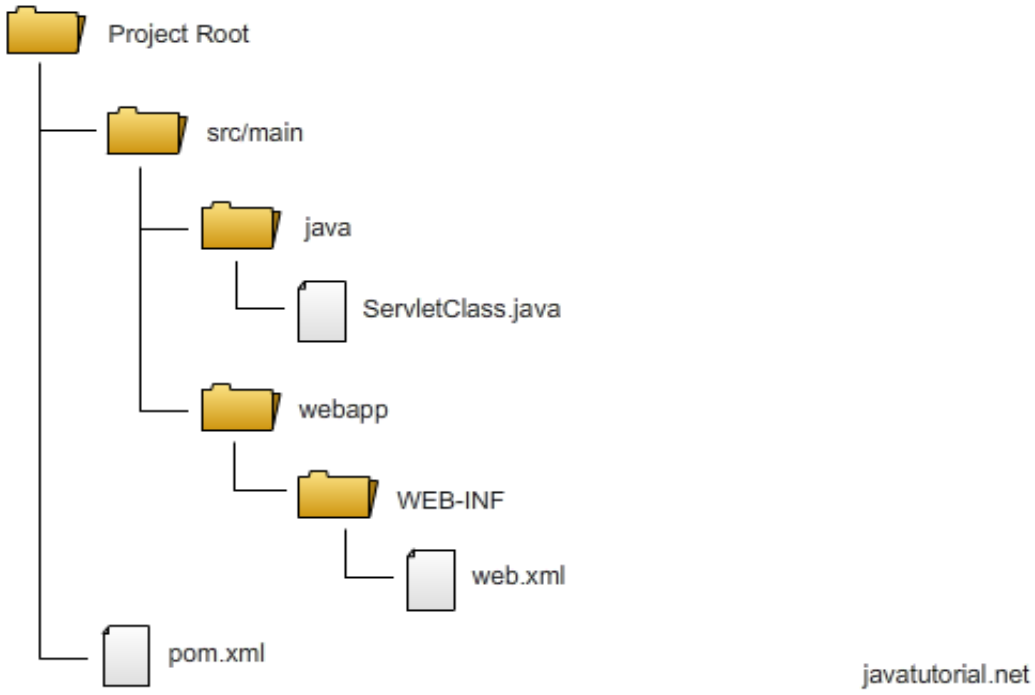
In this tutorial I will show you how to create Servlets and deploy them in Apache Tomcat 8

What are Servlets?

Servlets are the building blocks of almost every java web application. They provide the core functionality to accept HTTP requests and return HTTP responses to the user. Even if you use JSP to build your web pages, the JSP files are eventually compiled to Servlets by the application server or web container, cf. Glassfish or Tomcat. Servlets serve GET, POST, and also HEAD, PUT, DELETE, OPTIONS and TRACE requests and return a response to web clients. You are most probably familiar with GET and POST requests. It is recommend to invest some time and read about the other types of request listed above. More information about them can be found in W3Schools: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html .

Servlet Project Structure

For this example the SimpleServlet example will be built and deployed using Maven. One Servlet class will be created. The `web.xml` file, also called deployment descriptor, gives information to the web container how to handle your servlets. The following picture shows the file structure of the project



Note that any *html*, *jsp* or other files that would be in the project directory are placed in **webapp**.

Maven Example

This layout can be created automatically. The command:

```
mvn archetype:generate -DgroupId=com.javacodegeeks
   -DartifactId=SimpleServlet -DarchetypeArtifactId=maven-archetype-webapp
     -DinteractiveMode=false
```

executed from the **work** directory creates the setup above but without the **java** directory.

Project POM File

The basic file Maven uses to do its work is the Project Object Model (POM) file, an XML file that resides in the base directory of the project as pom.xml.

The POM file contains information about the project and various configuration details used by Maven to build project(s).

The POM file also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal.

The pom.xml used by Maven to perform the build for our Servlet projects looks like this:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>edu.kutztown.csc521</groupId>
        <artifactId>SimpleServlet</artifactId>
        <version>1</version>
        <packaging>war</packaging>

        <name>SimpleServlet</name>
        <url>http://faculty.kutztown.edu/spiegel/csc521/java/Servlets/SimpleServlet/</url>

        <properties>
                <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        </properties>

        <dependencies>
                <dependency>
                        <groupId>javax.servlet</groupId>
                        <artifactId>javax.servlet-api</artifactId>
                        <version>3.1.0</version>
                        <scope>provided</scope>
                </dependency>
        </dependencies>

    <build>
        <sourceDirectory>src/main/java</sourceDirectory>
```

1

2

Maven Example

```
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.3</version>
                <configuration>
                    <warSourceDirectory>src/main/webapp</warSourceDirectory>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

3

4

Here are explanations of numbered items above:

1. The packaging needs to be set to war. Web applications in Tomcat are packed into WAR files (Web archives) encapsulating all the data of the app. The name of the WAR file is built upon the `arifactId` and the version parameters you set in your Maven pom file. The name of our WAR file in this case will be SimpleServlet-1.war

2. The dependency we need to create servlets is javax.servlet-api . We set the scope to “provided” because Tomcat has all the libraries required already build in.

3. We need the Maven WAR plugin to be able to create the WAR file

4. In the compiler section of the WAR the plugin is set to 1.7, not 1.8, because our installation of Tomcat still uses Java 1.7’s compiler.

Simple Servlet Example

The source code of SimpleServlet is found here:

http://faculty.kutztown.edu/spiegel/csc521/java/Servlets/SimpleServlet/

Maven Example

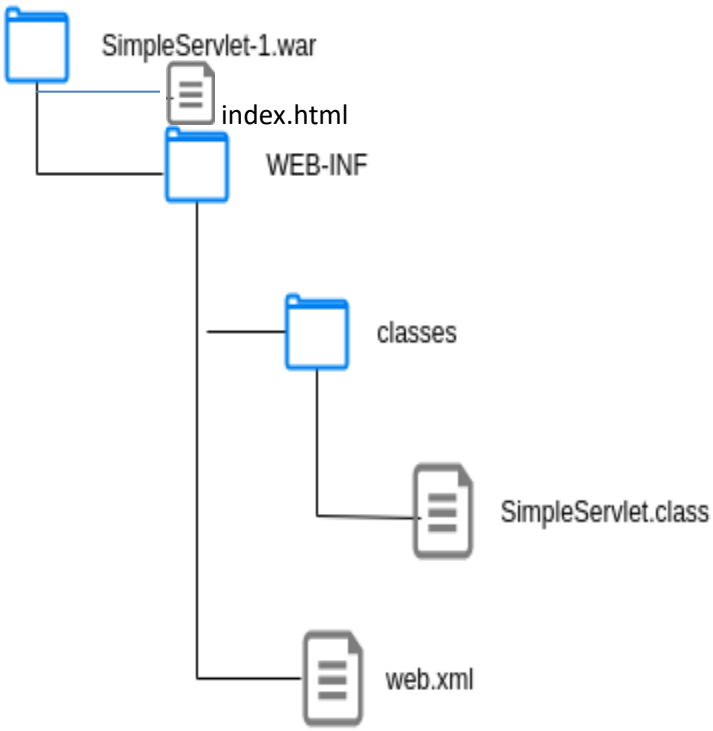
Build/Deploy the Example

Maven automates the creation of the war file. Using the command

mvn deploy

the build will list failure, but only because of the lack of a repository, i.e. a place to put the completed build. All is well, however, as a new directory under your project directory will be created, named *target*. In that directory will be your war file, which can be copied to the **webapps** directory to deploy.

The deployment setup as in the image below is also available in a subdirectory with the same name as the servlet in *target*.



WAR file structure

Do not edit anything under target. Edit under src/main, then let Maven do the work.