Late Night Java by Koray Tugay

Hello world web application with Maven + Java + Servlets + Tomcat + IntelliJ

This is a simple walk-through.

My goal is to talk about:

- creating a web application with maven using an arche-type.
- adding a Servlet to the application.
- compiling it using maven.
- deploying it to Tomcat and running it.

The default maven directory layout for a web-app is as follows, we will need this as reference:

src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/main/filters	Resource filter files
src/main/assembly	Assembly descriptors
src/main/config	Configuration files
src/main/scripts	Application/Library scripts
src/main/webapp	Web application sources
src/test/java	Test sources
src/test/resources	Test resources
src/test/filters	Test resource filter files
src/site	Site
LICENSE.txt	Project's license
NOTICE.txt	Notices and attributions required by libraries that the project depends on
README.txt	Project's readme

First create a java web application from standard maven arche type. Type this under any directory you like.
I chose C:\Development.

```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-webapp -DarchetypeArtifactId=maven-arche
```

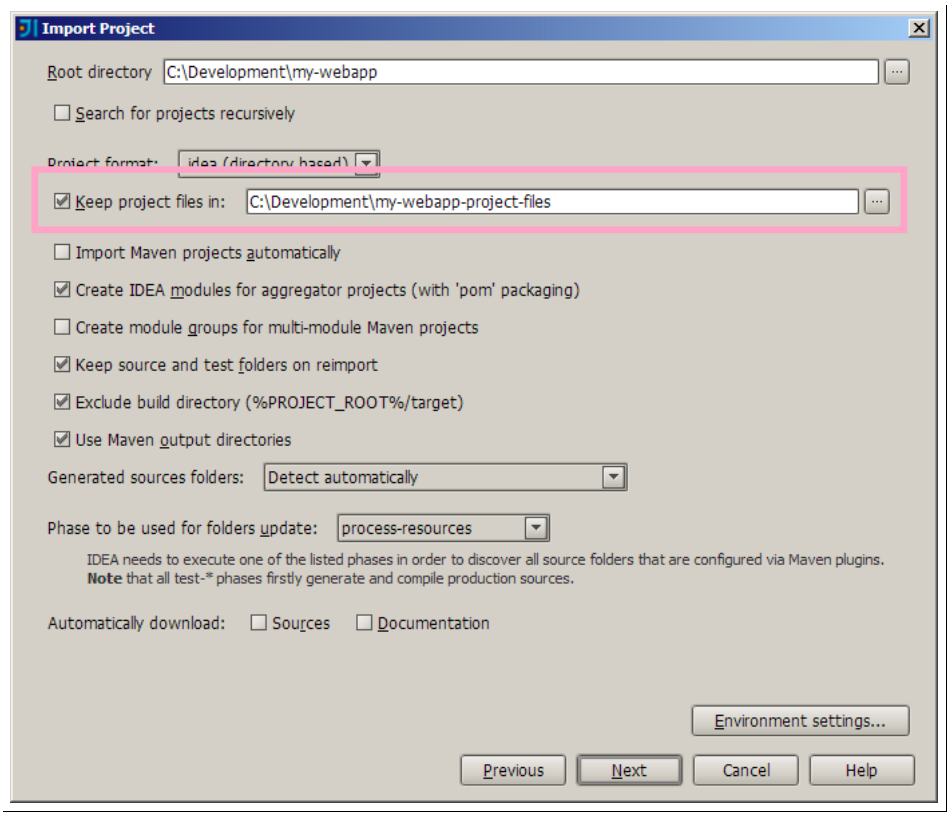
Following structure has been created by maven:

```
C:\Development\my-webapp>tree /F
Folder PATH listing
Volume serial number is 8CA6-3F63
C:.
│   pom.xml
│
└───src
    └───main
        ├───resources
        └───webapp
            │   index.jsp
            │
            └───WEB-INF
                    web.xml
```

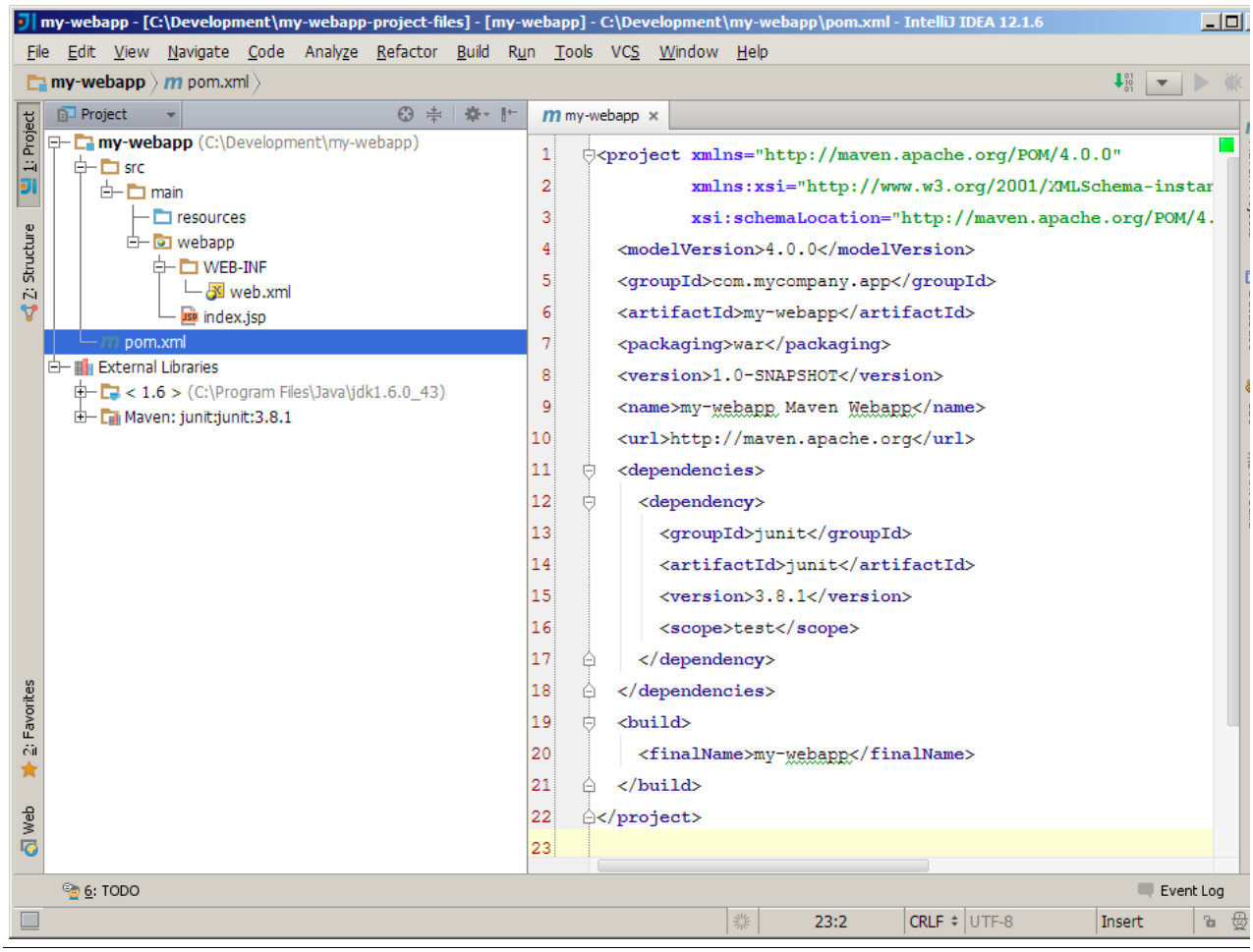
From now I will use IntelliJ but you can use any IDE you like I guess.

Import the project by starting IntelliJ and Import Project -> Pick the folder you created

I like to keep my project files in a seperate directory, so I check "Keep project files in:" and just pick a different folder..



Project is open in IntelliJ :



Lets first try a very simple Hello World from our application. For this we will need send a request to the web server and expect a response from it.

Our web-browser (in my case Chrome) will send a request to the server (Apache), and Apache will send a

response to the client (Chrome). The request is just some text data really. So is the response. However response must be created first.

So Chrome will send a request: "Get me this.."
And Apache will say: "Here is what you have asked for.."

But before Apache can send the response, it will first send the request to Tomcat. Tomcat will then get the request, create two objects: a HttpRequest and a HttpResponse object. We will send these 2 object to a class we have. Then we can do whatever we like to do with them...

To be able to get to this point we need these:

- Our class must extend HttpServlet (so Tomcat can make a connection with this class, like initialize it when required, and call methods from this class).
- We must define some sort of mapping so that Tomcat can know when to use this class(This servlet).

In **pom.xml** you have, add this:

```
<dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-api</artifactId>
    <version>6.0</version>
    <scope>provided</scope>
</dependency>
```

What we have done here?

We have added the javaee-api to our dependencies, so that we can extend from the HttpServletClass. We set the scope provided, because Tomcat is a web-container and already provides this class. More on this here:

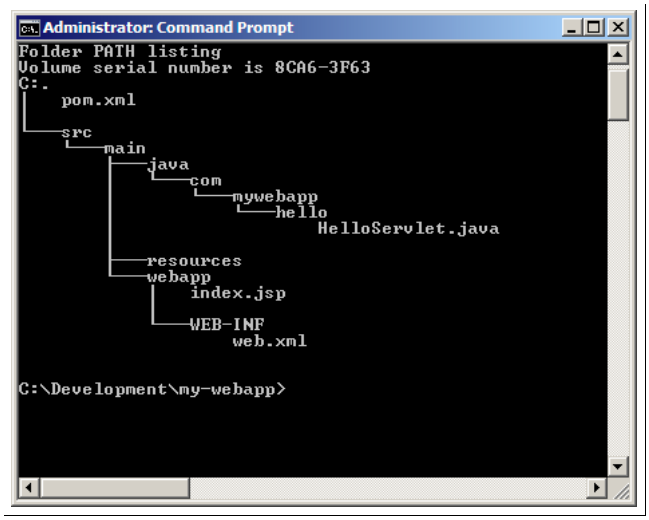
http://stackoverflow.com/questions/19471681/how-come-glassfish-does-not-fail-to-start-but-tomcat-does-if-scope-is-not-provid

So where should we put this class really? Check the default directory structure again. (The first screenshot in this tutorial.) It suggests us: **src\main\java**.. Let's do this then...

Now create a new folder called java under **src\main** (If you are using IntelliJ like I am, right click on it and click on "Mark Directory as Source Root").

Create these subfolders (or packages if you like):
com.mywebapp.hello and create a class called **HelloServlet**.

This is what the directory tree should look like:



Let's put some code in **HelloServlet.java**

```
package com.mywebapp.hello;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse)
        throws IOException {
        httpServletResponse.getWriter().print("Hello from HelloServlet");
    }
}
```

Now we need to do the mapping. We need to tell Tomcat this:

"When a request for a certain URL (address) comes, send the 2 object you have created to this class I have."

Open the **web.xml** file you have and make sure it looks like this:

```
<web-app version="2.5"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>com.mywebapp.hello.HelloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>

</web-app>
```

What have we done here?

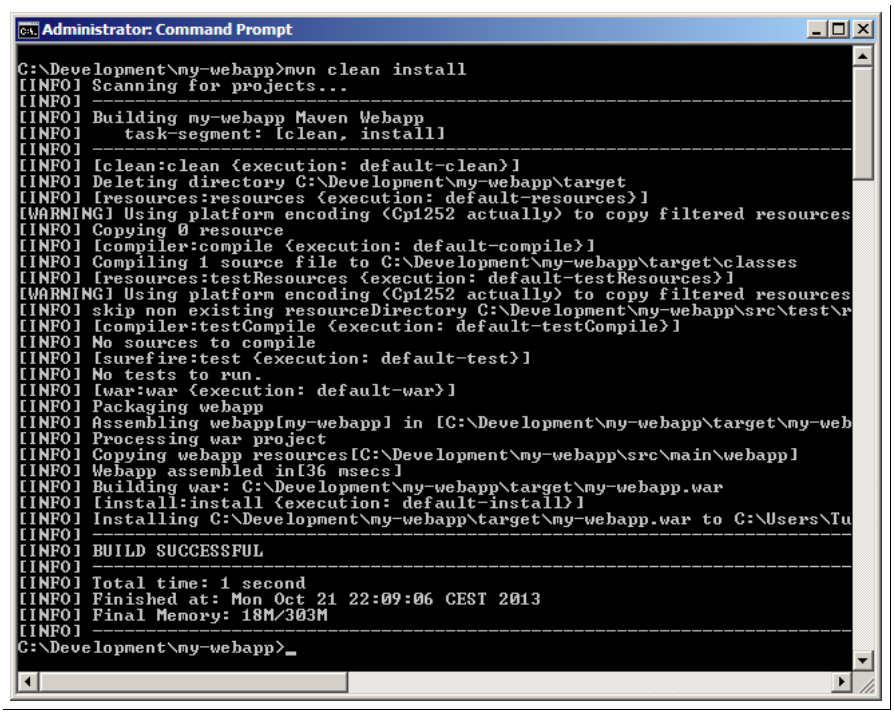
We told Tomcat that we have a Servlet, and the class is com.mywebapp.hello.HelloServlet. And we told Tomcat that we would like to call it HelloServlet..

Then we told Tomcat that, any request coming to /hello, just let HelloServlet handle everything..

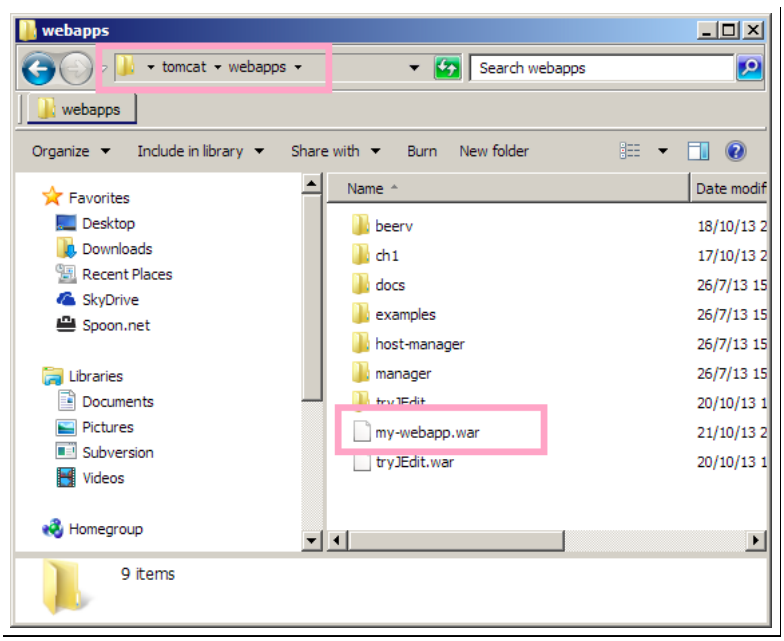
Ok, lets try to compile our code and deploy it in Tomcat.

In \my-webapp type

```
\my-webapp\mvn clean install
```

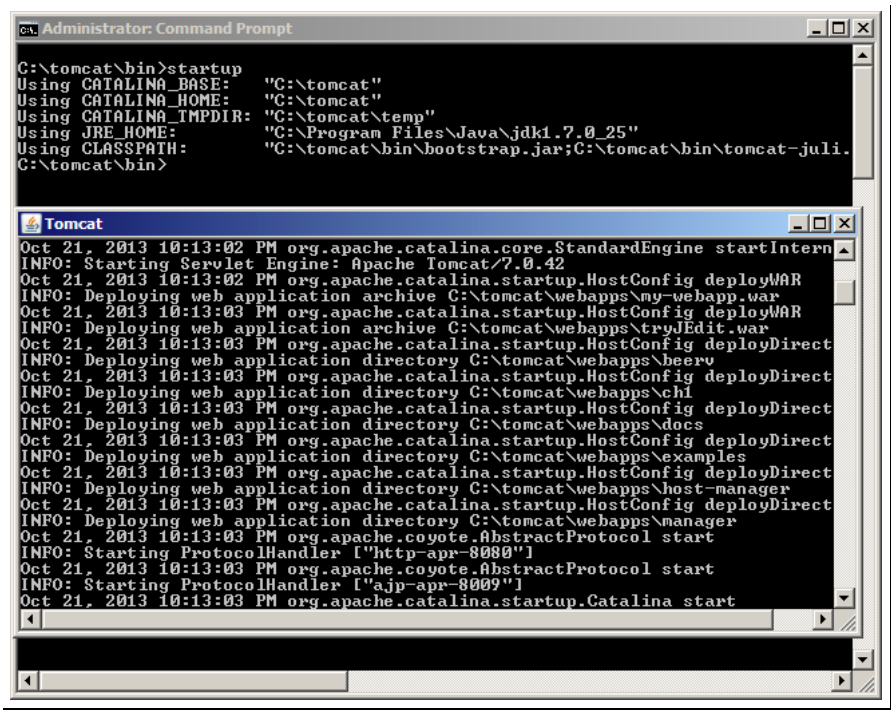


Now, in target folder you should have a my-webapp.war. This is like a jar file.. Tomcat knows how to deal with this file. Copy this war file into: webapps folder under Tomcat directory.

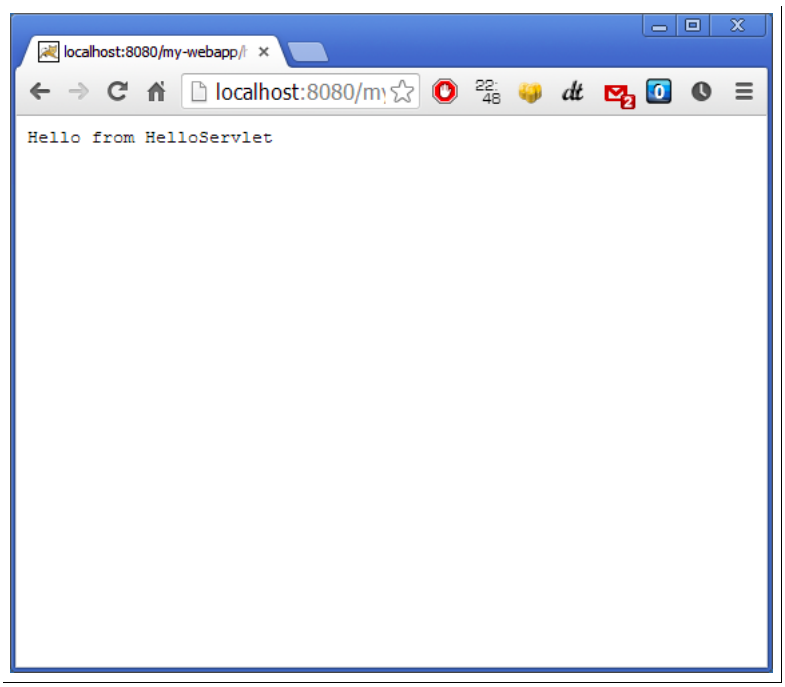


Now go to tomcat\bin folder and type:

```
\tomcat\bin\startup
```



Our server is running and we are ready to give it a try. Go to **localhost:8080/my-webapp/hello**



Great, it is working!

Etiketler: java, web-app

Newer Post Home

Me

Blog Archive

► 2040 (2)
► 2017 (30)

Contact Form

Name