Standalone web application with executable Tomcat

When it comes to deploying your application, simplicity is the biggest advantage. You'll understand that especially when project evolves and needs some changes in the environment. Packaging up your whole application in one, standalone and self-sufficient JAR seems like a good idea, especially compared to installing and upgrading Tomcat in target environment. In the past I would typically include Tomcat JARs in my web application and write thin command-line runner using Tomcat API. Luckily there is a `tomcat7:exec-war` maven goal [http://tomcat.apache.org/maven-plugin-2.0/tomcat7-maven-plugin/exec-war-mojo.html] that does just that. It takes your WAR artifact and packages it together with all Tomcat dependencies. At the end it also includes `Tomcat7RunnerCli` Main-class [http://tomcat.apache.org/maven-plugin-2.0/apidocs/org/apache/tomcat/maven/runner/Tomcat7RunnerCli.html] to manifest.

Curious to try it? Take your existing WAR project and add the following to your `pom.xml`:

```
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.0</version>
    <executions>
        <execution>
            <id>tomcat-run</id>
            <goals>
                <goal>exec-war-only</goal>
            </goals>
            <phase>package</phase>
            <configuration>
                <path>/standalone</path>
                <enableNaming>false</enableNaming>
                <finalName>standalone.jar</finalName>
                <charset>utf-8</charset>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Run `mvn package` and few seconds later you'll find shiny `standalone.jar` in your `target` directory. Running your web application was never that simple:
[]

```
$ java -jar target/standalone.jar
```

...and you can browse `localhost:8080/standalone`. Although the documentation of `path` parameter [http://tomcat.apache.org/maven-plugin-2.0/tomcat7-maven-plugin/exec-war-mojo.html#path] says (emphasis mine):

> The webapp context path to use for the web application being run. The name to store webapp in exec jar.
> **Do not use /**

just between the two of us, `<path>/</path>` seems to work after all. It turns out that built in `main` class is actually a little bit more flexible. For example you can say (I hope it's self-explanatory):

```
$ java -jar standalone.jar -httpPort=7070
```

What this runnable JAR does is it first unpacks WAR file inside of it to some directory (`.extract` by default[1]) and deploys it to Tomcat - all required Tomcat JARs are also included. Empty `standalone.jar` (with few KiB WAR inside) weights around 8.5 MiB - not that much if you claim that pushing whole Tomcat with every release alongside your application is wasteful.

Talking about Tomcat JARs, you should wonder how to choose Tomcat version included in this runnable? Unfortunately I couldn't find any simple option, so we must fall back to explicitly redefining **plugin** dependencies (version 2.0 has hardcoded 7.0.30 Tomcat). It's quite boring, but not that complicated and might be useful for future reference:

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <tomcat7Version>7.0.33</tomcat7Version>
</properties>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.0</version>
            <executions>
                <execution>
                    <id>tomcat-run</id>
                    <goals>
                        <goal>exec-war-only</goal>
                    </goals>
                    <phase>package</phase>
                    <configuration>
                        <path>/standalone</path>
                        <enableNaming>false</enableNaming>
                        <finalName>standalone.jar</finalName>
                        <charset>utf-8</charset>
                    </configuration>
                </execution>
            </executions>
            <dependencies>
                <dependency>
                  <groupId>org.apache.tomcat.embed</groupId>
                  <artifactId>tomcat-embed-core</artifactId>
                  <version>${tomcat7Version}</version>
                </dependency>
                <dependency>
                  <groupId>org.apache.tomcat</groupId>
                  <artifactId>tomcat-util</artifactId>
                  <version>${tomcat7Version}</version>
                </dependency>
                <dependency>
                  <groupId>org.apache.tomcat</groupId>
```

```
  <artifactId>tomcat-coyote</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-api</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jdbc</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-dbcp</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-servlet-api</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jsp-api</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jasper</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-jasper-el</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-el-api</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-catalina</artifactId>
  <version>${tomcat7Version}</version>
</dependency>
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-tribes</artifactId>
```

```
                <version>${tomcat7Version}</version>
              </dependency>
              <dependency>
                <groupId>org.apache.tomcat</groupId>
                <artifactId>tomcat-catalina-ha</artifactId>
                <version>${tomcat7Version}</version>
              </dependency>
              <dependency>
                <groupId>org.apache.tomcat</groupId>
                <artifactId>tomcat-annotations-api</artifactId>
                <version>${tomcat7Version}</version>
              </dependency>
            </dependencies>
        </plugin>
    </plugins>
</build>
```

In the next article we will learn how to tame these pesky Tomcat internal logs appearing in the terminal (`java.util.logging`...) In the meantime I discovered and reported *MTOMCAT-186 Closing executable JAR does not call ServletContextListener.contextDestroyed()* [https://issues.apache.org/jira/browse/MTOMCAT-186] - have look if this is a deal breaker for you.

[1] - it might be a good idea to specify different directory using `-extractDirectory` and clean it before every restart with `-resetExtract`.