TomEE Shading

Fat Jars

Shading the container and the application has some challenges like merging correctly resources (`META-INF/services/` typically).

Here is a maven shade plugin configuration working for most cases:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-shade-plugin</artifactId>
  <version>2.3</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>shade</goal>
      </goals>
      <configuration>
        <dependencyReducedPomLocation>${project.build.directory}/reduced-
pom.xml</dependencyReducedPomLocation>
        <transformers>
          <transformer implementation=
"org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
            <mainClass>org.apache.tomee.embedded.Main</mainClass>
          </transformer>
          <transformer implementation=
"org.apache.maven.plugins.shade.resource.AppendingTransformer">
            <resource>META-INF/cxf/bus-extensions.txt</resource>
          </transformer>
          <transformer implementation=
"org.apache.maven.plugins.shade.resource.AppendingTransformer">
            <resource>META-INF/openwebbeans/openwebbeans.properties</resource>
          </transformer>
        </transformers>
        <filters>
          <filter> <!-- we don't want JSF to be activated -->
            <artifact>*:*</artifact>
            <excludes>
              <exclude>META-INF/faces-config.xml</exclude>
            </excludes>
          </filter>
        </filters>
      </configuration>
    </execution>
  </executions>
</plugin>
```

NOTE see TomEE Embedded page for more information about tomee embedded options.

IMPORTANT	this shade uses TomEE Embedded but you can do the same with an Application Composer application.

TIP	if you have `META-INF/web-fragment.xml` in your application you will need to merge them in a single one in the shade. Note that tomcat provides one which can be skipped in this operation since it is there only as a marker for jasper detection.

Then just build the jar:

```
mvn clean package
```

And you can run it:

```
java -jar myapp-1.0-SNAPSHOT.jar
```

Fat Wars

Fat Wars are executable wars. Note they can be fancy for demos but they have the drawback to put the server in web resources at packaging time (to ensure the war is actually an executable jar) so adding a filter preventing these files to be read can be needed if you don't already use a web technology doing it (a servlet bound to /*).

Here how to do a fat war:

```
<properties>
  <!-- can be uber (for all), jaxrs, jaxws for lighter ones -->
  <tomee.flavor>uber</tomee.flavor>
</properties>

<dependencies>
  <!-- ...your dependencies as usual... -->
  <dependency>
    <groupId>org.apache.tomee</groupId>
    <artifactId>tomee-embedded</artifactId>
    <classifier>${tomee.flavor}</classifier>
    <version>7.0.0</version>
    <scope>provided</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>2.6</version>
      <configuration>
        <failOnMissingWebXml>false</failOnMissingWebXml>
        <archive>
          <manifest>
            <mainClass>org.apache.tomee.embedded.Main</mainClass>
          </manifest>
        </archive>
        <dependentWarExcludes />
        <overlays>
          <overlay>
            <groupId>org.apache.tomee</groupId>
            <artifactId>tomee-embedded</artifactId>
            <classifier>${tomee.flavor}</classifier>
            <type>jar</type>
            <excludes />
          </overlay>
        </overlays>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Then just build the war:

```
mvn clean package
```

And you can run it:

```
java -jar myapp-1.0-SNAPSHOT.war
```