链滴

使用 maven 打包遇到的一些问题

作者：flyue

原文链接：https://ld246.com/article/1530361400194

来源网站：链滴

1. 源代码路径和资源文件路径

- **sourceDirectory**: 指定源代码所在路径
  - 默认为src/main/java
- **testSourceDirectory**:测试代码所在路径
  - 默认为src/test/java
- **outputDirectory** : 编译后的classes存放路径
  - 默认为target/classes
- **resources** : 所有资源文件的目录

* 配置多个resource

* 默认为src/main/resources,源代码目录下的资源文件会被过滤掉，如果资源文件和源代码放在了一，则需要配置

```xml
<resources>
   <resource>
      <directory>src/main/java</directory>
      <includes>
         <include>**/*.xml</include>
         <include>**/*.properties</include>
      </includes>
   </resource>
   <resource>
      <directory>src/main/resources</directory>
   </resource>
 </resources>
```

2. maven使用本地jar包

- 使用本地的jar包有两种方式:

1. 将jar包安装进maven的仓库

安装命令如下:

```
mvn install:install-file \
-DgroupId=org.hibernate \
-DartifactId=c3p0 \
-Dversion=4.2.4.Final-Dpackaging=jar \
-Dfile=hibernate-c3p0-4.2.4.Final.jar
```

  - 分别指定groupId，artifactId，version,和文件路径(当前路径下的该jar)

2. dependency的scope使用system,同时指定jar包位置

原文链接：使用 maven 打包遇到的一些问题

```
 <dependency>
    <groupId>com.flyue</groupId>  <!--自定义-->
    <artifactId>message</artifactId>    <!--自定义-->
    <version>1.0</version> <!--自定义-->
    <scope>system</scope> <!--system，类似provided，需要显式提供依赖的jar以后，Maven不会在Repository中查找它-->
    <systemPath>${basedir}/src/main/resources/lib/message.jar</systemPath>  <!--项目根目下的lib文件夹下-->
</dependency>
```

3. 插件的使用

3.1 项目编译时使用的jdk版本以及源代码使用的编码

* 使用 __maven-compiler-plugin__ 插件
  ```xml
  <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.7.0</version>
      <configuration>
          <source>1.8</source>
          <target>1.8</target>
          <encoding>utf-8</encoding>
      </configuration>
  </plugin>

  ```

3.2 打包成jar使用的插件 maven_jar_plugin

* 使用 __mvn jar:jar__ 命令打包
* 使用该插件打包时只会打包自身的class和资源文件，依赖的jar包与资源文件不会一起打包。

```xml
  <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.6</version>
        <configuration>
            <archive>
                <manifest>
        <!-- 主要影响jar包中的META-INF/MANIFEST.MF-->
        <!-- 是否追加classpath路径 -->
                    <addClasspath>true</addClasspath>
        <!-- 追加的classpath所在路径 -->
                    <classpathPrefix>lib/</classpathPrefix>
        <!-- 指定main方法所在类，直接使用java -jar命令运行jar包时依赖这些配置 -->
                    <mainClass>com.xxg.Main</mainClass>
                </manifest>
            </archive>
```

```
    <!-- 指定jar的分类器,用于依赖是区别不同的包 -->
    <classifier>classes</classifier>
      </configuration>
    </plugin
```

3.3 打成war包 maven_war_plugin

1. 遇到需要将两个webapp项目A,B同时打成一个war包的需求,同时B依赖A

- 目前的主要步骤是：
  - A需要打成jar与war包同时安装到maven仓库中，打jar包需要指定分类器classifier
  - B项目依赖A项目的jar包，使得B能引用A项目的类，scope设置为provided,打包时不需要这个ar
  - B项目依赖A项目的war包，使用 **maven_war_plugin** 插件合并两个war包
  - **maven_war_plugin** 的合并两war包配置如下：

```
 <!-- 合并多个war -->
  <plugin>
     <groupId>org.apache.maven.plugins</groupId>
     <artifactId>maven-war-plugin</artifactId>
     <version>3.2.0</version>
     <executions>
        <execution>
           <!--绑定到maven 的生命周期package上 -->
           <!-- 当进行package时,会调用war-plugin进行打包 -->
           <phase>package</phase>
        </execution>
     </executions>
     <configuration>
        <!-- web.xml不合并 -->
        <packagingExcludes>WEB-INF/web.xml</packagingExcludes>
        <overlays>
           <!-- 合并的war包地址,overlays下可以写多个 -->
           <overlay>
              <groupId>com.xxx</groupId>
              <artifactId>xxx</artifactId>
           </overlay>
           <!--<overlay>
              <groupId></groupId>
              <artifactId></artifactId>
           </overlay>-->
        </overlays>
     </configuration>
  </plugin>
```

3.4 自定义打包 maven_assembly_plugin：

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
```

```
 <artifactId>maven-assembly-plugin</artifactId>
 <version>3.1.0</version>
 <executions>  <!--执行器 mvn assembly:assembly-->
    <execution>
       <id>make-war</id><!--名字任意 -->
       <phase>package</phase><!-- 绑定到package生命周期阶段上 -->
       <goals>
          <goal>single</goal><!-- 只运行一次 -->
       </goals>
       <configuration>
          <descriptors>  <!--打包配置的描述文件路径-->
             <descriptor>src/main/assembly/src.xml</descriptor>
          </descriptors>
       </configuration>
    </execution>
 </executions>
</plugin>
```

- **src/main/assembly/src.xml配置**
  - 将依赖的指定的jar包解压和项目编译后的代码一同放到classes下示例

```
<assembly xmlns="http://maven.apache.org/ASSEMBLY/2.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/ASSEMBLY/2.0.0 http://maven.apache.org
xsd/assembly-2.0.0.xsd">
 <id>release</id>
 <formats>
    <!-- 将项目打成war包 -->
    <format>war</format>
 </formats>

 <!-- 设置自身项目文件目录 -->
 <fileSets>
    <fileSet>
       <!-- 将编译后的classes放到WEB-INF/classes下 -->
       <directory>target/classes</directory>
       <outputDirectory>/WEB-INF/classes/</outputDirectory>
    </fileSet>
    <fileSet>
       <!-- webapp 下的文件到根目录 -->
       <directory>src/main/webapp</directory>
       <outputDirectory>/</outputDirectory>
    </fileSet>
 </fileSets>

 <!-- 设置依赖的放置目录 -->
 <dependencySets>
    <dependencySet>
       <!-- 项目自身不考虑进去 -->
       <useProjectArtifact>false</useProjectArtifact>
       <outputDirectory>/WEB-INF/lib/</outputDirectory><!-- 将scope为runtime的依赖包打
到lib目录下。 -->
       <excludes>
```

```
            <!-- 排除掉一些jar包,用groudId:artifactId方式 -->
            <exclude>
                com.taotao
            </exclude>
        </excludes>
    </dependencySet>
    <dependencySet>
        <!-- 项目自身不考虑进去 -->
        <useProjectArtifact>false</useProjectArtifact>
        <outputDirectory>/WEB-INF/classes/</outputDirectory>
        <!-- 将依赖解压 -->
        <unpack>true</unpack>
        <unpackOptions>
            <includes>
                <include>com/taotao/**</include>
            </includes>
        </unpackOptions>
        <includes>
            <include>
                com.taotao
            </include>
        </includes>
    </dependencySet>
 </dependencySets>
</assembly>
```

3.5 maven 的jetty插件:

```
  <plugin>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>9.4.5.v20170502</version>
    <configuration>
        <scanIntervalSeconds>10</scanIntervalSeconds>
        <httpConnector>
            <port>8080</port>
        </httpConnector>
        <webApp>
            <contextPath>/</contextPath>
        </webApp>
    </configuration>
  </plugin>
```