[maven生成war包的两种方式](http://www.iteye.com/topic/545708)

 war包即对WEB应用程序进行打包，用于应用容器的部署。如在jboss中只要把war包丢入deploy目录下即可发布自己的应用了。打包方式有很多中，很多工具本身就支持此功能。下面主要介绍通过maven来操作的两种方式：

**方法一：**

   即第一反应的那种啦，就是利用maven的maven-war-plugin。

**Xml代码  [收藏代码](javascript:void())**

1. **<build>**
2. **<plugins>**
3. ......
4. **<plugin>**
5. **<groupId>**org.apache.maven.plugins**</groupId>**
6. **<artifactId>**maven-war-plugin**</artifactId>**
7. **<version>**2.0.2**</version>**
8. **<configuration>**
9. **<warSourceExcludes>**WEB-INF/lib/log4j-${log4j.version}.jar**</warSourceExcludes>**
10. **</configuration>**
11. **</plugin>**
12. **</plugins>**
13. **</build>**

当然你可以指定war包的名称，输出的目录，webapp目录等等。

可以在需要打包的工程中直接设定：<pom>war</pom>

这样在install的时候就会对此工程生成相应配置的war包，原来此web工程依赖的其他工程全部会以jar包的形式出现在lib目录下，所以不需要再去关心war包中集成的jar包是否完整，maven已经帮你做完了。

顺便提一句，在jboss中发布war包，如果包中包含log4j.jar，会报错，所以在本例中将此jar包排除在外。

**方法二：**

不用war插件，而是用到另外三个插件：maven-dependency-plugin 、maven-antrun-plugin 、 build-helper-maven-plugin

下面分别说明下着三个插件在此过程扮演的角色：

maven-dependency-plugin ：相当于数据收集器，该依赖那些jar包，该加载那些resource信息。

maven-antrun-plugin：完成收集完文件夹的压缩和包的重命名。

build-helper-maven-plugin：让压缩包拥有男人本色，oh sorry 是war本色。

在写pom.xml配置前，先说明下目录结构

root

|

|---template

|

|---wardir

其中这两个目录都作为maven的独立工程，下面要写配置在wardir下的pom.xml中

**Xml代码  [收藏代码](javascript:void())**

1. **<build>**
2. **<plugins>**
3. **<plugin>**
4. **<groupId>**org.apache.maven.plugins**</groupId>**
5. **<artifactId>**maven-dependency-plugin**</artifactId>**
6. **<version>**2.0**</version>**
7. **<executions>**
8. **<execution>**
9. **<id>**create-structure**</id>**
10. **<phase>**process-resources**</phase>**
11. **<goals>**
12. **<goal>**unpack**</goal>**
13. **</goals>**
14. **<configuration>**
15. **<artifactItems>**
16. **<artifactItem>**
17. **<groupId>**GROUPID**</groupId>**
18. **<artifactId>**template**</artifactId>**
19. **</artifactItem>**
20. **</artifactItems>**
21. **<outputDirectory>**
22. ${project.build.directory}/${assembly.name}.war
23. **</outputDirectory>**
24. **</configuration>**
25. **</execution>**
26. **<execution>**
27. **<id>**copy-libs**</id>**
28. **<phase>**process-resources**</phase>**
29. **<goals>**
30. **<goal>**copy**</goal>**
31. **</goals>**
32. **<configuration>**
33. **<artifactItems>**
34. **<artifactItem>**
35. **<groupId>**jdom**</groupId>**
36. **<artifactId>**jdom**</artifactId>**
37. **</artifactItem>**
38. ......
39. **</artifactItems>**
41. **<outputDirectory>**${project.build.directory}/${assembly.name}.war/WEB-INF/lib**</outputDirectory>**
42. **</configuration>**
43. **</execution>**
44. **</executions>**
45. **</plugin>**
46. **<plugin>**
47. **<groupId>**org.apache.maven.plugins**</groupId>**
48. **<artifactId>**maven-antrun-plugin**</artifactId>**
49. **<version>**1.3**</version>**
50. **<executions>**
51. **<execution>**
52. **<id>**copy-config**</id>**
53. **<phase>**process-resources**</phase>**
54. **<goals>**
55. **<goal>**run**</goal>**
56. **</goals>**
57. **<configuration>**
58. **<tasks>**
59. **<copy** todir="${project.build.directory}/${assembly.name}.war"**>**
60. **<fileset** dir="src/main/resources" **/>**
61. **</copy>**
62. **<zip** destfile="${project.build.directory}/${assembly.name}-${project.version}-war.zip"**>**
63. **<zipfileset** dir="${project.build.directory}/${assembly.name}.war" includes="\*\*\\*.\*"  excludes="\*.zip" **/>**
64. **</zip>**
65. **<copy** file="${project.build.directory}/${assembly.name}-${project.version}-war.zip"
66. tofile="${project.build.directory}/${assembly.name}-${project.version}.war" **/>**
67. **</tasks>**
68. **</configuration>**
69. **</execution>**
70. **</executions>**
71. **</plugin>**
72. **<plugin>**
73. **<groupId>**org.codehaus.mojo**</groupId>**
74. **<artifactId>**build-helper-maven-plugin**</artifactId>**
75. **<version>**1.4**</version>**
76. **<executions>**
77. **<execution>**
78. **<id>**attach-artifacts**</id>**
79. **<phase>**package**</phase>**
80. **<goals>**
81. **<goal>**attach-artifact**</goal>**
82. **</goals>**
83. **<configuration>**
84. **<artifacts>**
85. **<artifact>**
86. **<file>**${project.build.directory}/${assembly.name}-${project.version}.war**</file>**
87. **<type>**war**</type>**
88. **</artifact>**
89. **</artifacts>**
90. **</configuration>**
91. **</execution>**
92. **</executions>**
93. **</plugin>**
94. **</plugins>**
95. **</build>**

 当然这种方式也可以用来打ear包。对于打无页面操作系统的war包，个人认为用后者更灵活，当然建立假web工程，再依赖工程，通过第一种方式同样可以达到最终的结果。看个人喜好了。

结束语：maven真的很强大！