[**Maven生成可以直接运行的jar包的多种方式**](http://blog.csdn.net/xiao__gui/article/details/47341385)

标签： [jar](http://www.csdn.net/tag/jar)[maven](http://www.csdn.net/tag/maven)[shade](http://www.csdn.net/tag/shade)[assembly](http://www.csdn.net/tag/assembly)[spring](http://www.csdn.net/tag/spring)

2015-08-07 18:56 44346人阅读 [评论](http://blog.csdn.net/xiao__gui/article/details/47341385#comments)(5) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/xiao__gui/article/details/47341385#report)

 分类：

Java语言（54）

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/xiao__gui/article/details/47341385)

Maven可以使用mvn package指令对项目进行打包，如果使用[**Java**](http://lib.csdn.net/base/java) -jar xxx.jar执行运行jar文件，会出现"no main manifest attribute, in xxx.jar"（没有设置Main-Class）、ClassNotFoundException（找不到依赖包）等错误。

要想jar包能直接通过java -jar xxx.jar运行，需要满足：

1、在jar包中的META-INF/MANIFEST.MF中指定Main-Class，这样才能确定程序的入口在哪里；

2、要能加载到依赖包。

使用Maven有以下几种方法可以生成能直接运行的jar包，可以根据需要选择一种合适的方法。

**方法一：使用maven-jar-plugin和maven-dependency-plugin插件打包**

在pom.xml中配置：

**[html]** [view plain](http://blog.csdn.net/xiao__gui/article/details/47341385) [copy](http://blog.csdn.net/xiao__gui/article/details/47341385)

1. **<build>**
2. **<plugins>**
4. **<plugin>**
5. **<groupId>**org.apache.maven.plugins**</groupId>**
6. **<artifactId>**maven-jar-plugin**</artifactId>**
7. **<version>**2.6**</version>**
8. **<configuration>**
9. **<archive>**
10. **<manifest>**
11. **<addClasspath>**true**</addClasspath>**
12. **<classpathPrefix>**lib/**</classpathPrefix>**
13. **<mainClass>**com.xxg.Main**</mainClass>**
14. **</manifest>**
15. **</archive>**
16. **</configuration>**
17. **</plugin>**
18. **<plugin>**
19. **<groupId>**org.apache.maven.plugins**</groupId>**
20. **<artifactId>**maven-dependency-plugin**</artifactId>**
21. **<version>**2.10**</version>**
22. **<executions>**
23. **<execution>**
24. **<id>**copy-dependencies**</id>**
25. **<phase>**package**</phase>**
26. **<goals>**
27. **<goal>**copy-dependencies**</goal>**
28. **</goals>**
29. **<configuration>**
30. **<outputDirectory>**${project.build.directory}/lib**</outputDirectory>**
31. **</configuration>**
32. **</execution>**
33. **</executions>**
34. **</plugin>**
36. **</plugins>**
37. **</build>**

maven-jar-plugin用于生成META-INF/MANIFEST.MF文件的部分内容，<mainClass>com.xxg.Main</mainClass>指定MANIFEST.MF中的Main-Class，<addClasspath>true</addClasspath>会在MANIFEST.MF加上Class-Path项并配置依赖包，<classpathPrefix>lib/</classpathPrefix>指定依赖包所在目录。

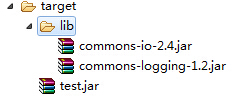
例如下面是一个通过maven-jar-plugin插件生成的MANIFEST.MF文件片段：

**[plain]** [view plain](http://blog.csdn.net/xiao__gui/article/details/47341385) [copy](http://blog.csdn.net/xiao__gui/article/details/47341385)

1. Class-Path: lib/commons-logging-1.2.jar lib/commons-io-2.4.jar
2. Main-Class: com.xxg.Main

只是生成MANIFEST.MF文件还不够，maven-dependency-plugin插件用于将依赖包拷贝到<outputDirectory>${project.build.directory}/lib</outputDirectory>指定的位置，即lib目录下。

配置完成后，通过mvn package指令打包，会在target目录下生成jar包，并将依赖包拷贝到target/lib目录下，目录结构如下：



指定了Main-Class，有了依赖包，那么就可以直接通过java -jar xxx.jar运行jar包。

这种方式生成jar包有个缺点，就是生成的jar包太多不便于管理，下面两种方式只生成一个jar文件，包含项目本身的代码、资源以及所有的依赖包。

**方法二：使用maven-assembly-plugin插件打包**

在pom.xml中配置：

**[html]** [view plain](http://blog.csdn.net/xiao__gui/article/details/47341385) [copy](http://blog.csdn.net/xiao__gui/article/details/47341385)

1. **<build>**
2. **<plugins>**
4. **<plugin>**
5. **<groupId>**org.apache.maven.plugins**</groupId>**
6. **<artifactId>**maven-assembly-plugin**</artifactId>**
7. **<version>**2.5.5**</version>**
8. **<configuration>**
9. **<archive>**
10. **<manifest>**
11. **<mainClass>**com.xxg.Main**</mainClass>**
12. **</manifest>**
13. **</archive>**
14. **<descriptorRefs>**
15. **<descriptorRef>**jar-with-dependencies**</descriptorRef>**
16. **</descriptorRefs>**
17. **</configuration>**
18. **</plugin>**
20. **</plugins>**
21. **</build>**

打包方式：

**[plain]** [view plain](http://blog.csdn.net/xiao__gui/article/details/47341385) [copy](http://blog.csdn.net/xiao__gui/article/details/47341385)

1. mvn package assembly:single

打包后会在target目录下生成一个xxx-jar-with-dependencies.jar文件，这个文件不但包含了自己项目中的代码和资源，还包含了所有依赖包的内容。所以可以直接通过java -jar来运行。

此外还可以直接通过mvn package来打包，无需assembly:single，不过需要加上一些配置：

**[html]** [view plain](http://blog.csdn.net/xiao__gui/article/details/47341385) [copy](http://blog.csdn.net/xiao__gui/article/details/47341385)

1. **<build>**
2. **<plugins>**
4. **<plugin>**
5. **<groupId>**org.apache.maven.plugins**</groupId>**
6. **<artifactId>**maven-assembly-plugin**</artifactId>**
7. **<version>**2.5.5**</version>**
8. **<configuration>**
9. **<archive>**
10. **<manifest>**
11. **<mainClass>**com.xxg.Main**</mainClass>**
12. **</manifest>**
13. **</archive>**
14. **<descriptorRefs>**
15. **<descriptorRef>**jar-with-dependencies**</descriptorRef>**
16. **</descriptorRefs>**
17. **</configuration>**
18. **<executions>**
19. **<execution>**
20. **<id>**make-assembly**</id>**
21. **<phase>**package**</phase>**
22. **<goals>**
23. **<goal>**single**</goal>**
24. **</goals>**
25. **</execution>**
26. **</executions>**
27. **</plugin>**
29. **</plugins>**
30. **</build>**

其中<phase>package</phase>、<goal>single</goal>即表示在执行package打包时，执行assembly:single，所以可以直接使用mvn package打包。

不过，如果项目中用到[**spring**](http://lib.csdn.net/base/javaee) Framework，用这种方式打出来的包运行时会出错，使用下面的方法三可以处理。

**方法三：使用maven-shade-plugin插件打包**

在pom.xml中配置：

**[html]** [view plain](http://blog.csdn.net/xiao__gui/article/details/47341385) [copy](http://blog.csdn.net/xiao__gui/article/details/47341385)

1. **<build>**
2. **<plugins>**
4. **<plugin>**
5. **<groupId>**org.apache.maven.plugins**</groupId>**
6. **<artifactId>**maven-shade-plugin**</artifactId>**
7. **<version>**2.4.1**</version>**
8. **<executions>**
9. **<execution>**
10. **<phase>**package**</phase>**
11. **<goals>**
12. **<goal>**shade**</goal>**
13. **</goals>**
14. **<configuration>**
15. **<transformers>**
16. **<transformer** implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer"**>**
17. **<mainClass>**com.xxg.Main**</mainClass>**
18. **</transformer>**
19. **</transformers>**
20. **</configuration>**
21. **</execution>**
22. **</executions>**
23. **</plugin>**
25. **</plugins>**
26. **</build>**

配置完成后，执行mvn package即可打包。在target目录下会生成两个jar包，注意不是original-xxx.jar文件，而是另外一个。和maven-assembly-plugin一样，生成的jar文件包含了所有依赖，所以可以直接运行。

如果项目中用到了Spring Framework，将依赖打到一个jar包中，运行时会出现读取XML schema文件出错。原因是Spring Framework的多个jar包中包含相同的文件spring.handlers和spring.schemas，如果生成一个jar包会互相覆盖。为了避免互相影响，可以使用AppendingTransformer来对文件内容追加合并：

**[html]** [view plain](http://blog.csdn.net/xiao__gui/article/details/47341385) [copy](http://blog.csdn.net/xiao__gui/article/details/47341385)

1. **<build>**
2. **<plugins>**
4. **<plugin>**
5. **<groupId>**org.apache.maven.plugins**</groupId>**
6. **<artifactId>**maven-shade-plugin**</artifactId>**
7. **<version>**2.4.1**</version>**
8. **<executions>**
9. **<execution>**
10. **<phase>**package**</phase>**
11. **<goals>**
12. **<goal>**shade**</goal>**
13. **</goals>**
14. **<configuration>**
15. **<transformers>**
16. **<transformer** implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer"**>**
17. **<mainClass>**com.xxg.Main**</mainClass>**
18. **</transformer>**
19. **<transformer** implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer"**>**
20. **<resource>**META-INF/spring.handlers**</resource>**
21. **</transformer>**
22. **<transformer** implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer"**>**
23. **<resource>**META-INF/spring.schemas**</resource>**
24. **</transformer>**
25. **</transformers>**
26. **</configuration>**
27. **</execution>**
28. **</executions>**
29. **</plugin>**
31. **</plugins>**
32. **</build>**