Maven3 实战笔记 05 仓库依赖解析与插件解析

刘岩

Email:suhuanzheng7784877@163.com

1. Maven 仓库依赖解析机制

本节复习前文背景是：http://suhuanzheng7784877.iteye.com/blog/1069252

当本地仓库没有依赖组件的时候，Maven 会从远程的中心仓库或者私服下载依赖包，当依赖的版本是快照版本的时候，则自动先找到快照的最新版本。

1.1：当依赖范围是 system 的时候，Maven 直接从本地库解析

1.2：根据咱们之前提到的 Maven 坐标解析路径后，开始查找工作，如果根据坐标发现了该组件，那么认为此次解析依赖成功

2.1：当本地仓库不存在相应组件的情况下，如果在 pom.xml 写着以来的版本是显示的发布版本，例如

```
<dependency>
    <groupId>javax.mail</groupId>
    <artifactId>mail</artifactId>
    <version>1.4.1</version>
</dependency>
```

就是要 javax.mail.jar.1.4.1 版本，遍历远程仓库，如果在远程仓库发现了精确版本，下载。

2.2.1：如果在 pom.xml 文件中依赖写着的版本是基于更新策略（RELEASE 或者 LATEST）读取远程仓库元数据，将远程元数据的版本与本地仓库元数据仓库对应合并后，计算出版本真实值，然后基于这个真实的值检查本地和远程仓库。重复 1.1 和 1.2 步骤。

2.2.2：如果 pom 文件中依赖的版本是快照版本 SNAPSHOP 版本，则读取远程仓库的元数据，将其与本地仓库元数据进行对比，合并，得到一个快照版本值，之后看哪个比较

新，从相应的库下载（一般都是远程仓库比本地的新，基本不可能本地的比远程的仓库版本还新吧，除非是该组件的原作者）

2.2.3：如果最后解析出来的版本是时间戳，会将其替换成快照格式。

从而可以看出，当依赖版本很“暧昧”的时候——RELEASE、LATEST、SNAPSHOP，Maven 需要将本地仓库和远程仓库进行对比，本地库与远程库要进行合并、更新。与此有关的配置是 releases、snapshots，只有开启了相关配置，才能访问仓库的发布版本信息。

```
    <repositories>
        <repository>
            <id>jboss</id>
            <url>http://repository.jboss.com/maven2/</url>
            <releases>
                <enabled>true</enabled>
            </releases>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
            <layout>default</layout>
        </repository>
    </repositories>
```

RELEASE：代表最新发布版本（发布的意思就是较为稳定，经过测试）

LATEST：代表最新版本（包含快照版本）

```
<?xml version="1.0" encoding="UTF-8"?><metadata>
  <groupId>org.icefaces</groupId>
  <artifactId>icefaces-comps</artifactId>
  <version>1.6.1</version>
  <versioning>
    <versions>
```

```
        <version>1.6.1</version>
        <version>1.6.2</version>
        <version>1.7.0</version>
        <version>1.7.1</version>
        <version>1.7.2</version>
        <version>1.8.0</version>
      </versions>
      <lastUpdated>20110602022501</lastUpdated>
   </versioning>
</metadata>
```

```
<?xml version="1.0" encoding="UTF-8"?><metadata>
   <groupId>org.eclipse.persistence</groupId>
   <artifactId>eclipselink</artifactId>
   <version>1.0.2</version>
   <versioning>
      <versions>
         <version>1.0.2</version>
      </versions>
      <lastUpdated>20110602022437</lastUpdated>
   </versioning>
</metadata>
```

需要注意的是，这种暧昧的版本在 Maven 中不太推荐，有可能今天构建成功了，明天第三方组织发布了一个快照版本，接口全变了，那就构建失败了。所以别搞暧昧，需要什么版本，大大方方的提出来即可。

2. Maven 插件解析机制

本节复习前文背景是：http://suhuanzheng7784877.iteye.com/blog/1069257

我们使用 Maven 插件的目标的时候都是利用他的前缀（简写），一旦执行命令出问题了，比较难定位具体是哪个插件运行出错的。在此咱们一起来看看它的插件插件机制。为何将仓库的依赖解析和插件解析放在一起呢，因为他们确实有相似的地方。插件的组件也是基于坐标存在于 Maven 库中，需要的时候，从本地仓库需要相关插件，不存在，从远程仓库去找，找到后下载到本地。当然了区别于依赖的是插件的远程库必须显示的在 pom 文件中配置，不配置，不会去远程下载。

```
<pluginRepositories>
    <pluginRepository>
        <id>central</id>
        <name>Maven plugin</name>
        <url>htpp://repo1.maven.org/maven2</url>
        <layout>default</layout>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
        <releases>
            <enabled>false</enabled>
        </releases>
    </pluginRepository>
</pluginRepositories>
```

这个和依赖库的配置意思差不多打开快照版本插件，如果自己写了 Maven 插件，可以参考上面的配置。使用插件时，默认的 groupId 的值是 org.apache.maven.plugins，是官方 apache 标准插件。

```
<build>
    <plugins>
```

```
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers

    implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">

    <mainClass>com.liuyan.maven.helloword.HelloWorld</mainClass>

                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
```

<groupId>org.apache.maven.plugins</groupId>其实可以省略。

使用插件的版本解析和仓库依赖组件的解析一样，先从本地解析，寻找。没有了从配置仓库查找。同样建议别使用“暧昧”版本插件。

至于插件的前缀解析机制，可以参看插件元数据

Windows7 平台：C:\Users\用户名\.m2\repository\org\apache\maven\plugins 下面的 maven-metadata-central.xml 文件描述了大多数常用的插件信息、以及前缀简写信息。

使用——简写插件：目标后 Maven 会根据简写找到具体的插件全名，之后根据插件的当前元数据 groupId，之后在找到版本号信息，最后就得到了完整的该插件的完整坐标。之后按照坐标去找具体实现、下载、使用之。

3. 总结

这次补充了仓库依赖和插件的解析机制，让我们更了解一些 Maven 的内核机制。