Maven详解之仓库------本地仓库、远程仓库

在Maven中，任何一个依赖、插件或者项目构建的输出，都可以称之为构件。

Maven在某个统一的位置存储所有项目的共享的构件，这个统一的位置，我们就称之为仓库。（仓库就是存放依赖和插件的地方）

任何的构件都有唯一的坐标，Maven根据这个坐标定义了构件在仓库中的唯一存储路径，

解读Maven在仓库中的存储路径：

1.基于groupId准备路径，将句点分隔符转成路径分隔符，就是将 "." 转换成 "/" ; example： org.testng --->org/testng

2.基于artifactId准备路径，将artifactId连接到后面：org/testng/testng

3.使用version准备路径，将version连接到后面：org/testng/testng/5.8

4.将artifactId于version以分隔符连字号连接到后面：org/testng/testng/5.8/tesng-5.8

5.判断如果构件有classifier，就要在 第4项 后增加 分隔符连字号 再加上 classifier，org/testng/testng/5.8/tesng-5.8-jdk5

6.检查构件的extension，如果extension存在，则加上句点分隔符和extension，而extension是由packing决定的，org/testng/testng/5.8/tesng-5.8-jdk5.jar

到这里我们就明白了Maven 对于构件存储的细节。

Maven 仓库的分类：

maven的仓库只有两大类：1.本地仓库 2.远程仓库，在远程仓库中又分成了3种：2.1 中央仓库 2.2 私服 2.3 其它公共库

1.本地仓库，顾名思义，就是Maven在本地存储构件的地方。

注：maven的本地仓库，在安装maven后并不会创建，它是在第一次执行maven命令的时候才被创建

maven本地仓库的默认位置：无论是Windows还是Linux，在用户的目录下都有一个.m2/repository/的仓库目录，这就是Maven仓库的默认位置

如何更改maven默认的本地仓库的位置：这里要引入一个新的元素：localRepository，它是存在于maven的settings.xml文件中

1.1 更改配置用户范围的本地仓库：先在/.m2/目录下创建settings.xml文件，然后在~/.m2/settings.xml，设置localRepository元素的值为想要的仓库地址

[html] view plain copy

<settings>

<localRepository>D:\maven\_new\_repository</localRepository>

</settings>

这时候，maven的本地仓库地址就变成了 D:\maven\_new\_repository ，注：此时配置的maven的本地仓库是属于用户范围的。

1.2 更改配置全局范围的本地仓库：在M2\_HOME/conf/settings.xml中更改配置，更改配置的方法同上

注：此时更改后，所有的用户都会受到影响，而且如果maven进行升级，那么所有的配置都会被清除，所以要提前复制和备份M2\_HOME/conf/settings.xml文件

故：一般情况下不推荐配置全局的settings.xml

2. 远程仓库

2.1 说到远程仓库先从 最核心的中央仓库开始，中央仓库是默认的远程仓库，maven在安装的时候，自带的就是中央仓库的配置

在maven的聚合与继承中我们说过，所有的maven项目都会继承超级pom，具体的说，包含了下面配置的pom我们就称之为超级pom

[html] view plain copy

<repositories>

<repository>

<id>central</id>

<name>Central Repository</name>

<url>http://repo.maven.apache.org/maven2</url>

<layout>default</layout>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

中央仓库包含了绝大多数流行的开源Java构件，以及源码、作者信息、SCM、信息、许可证信息等。一般来说，简单的Java项目依赖的构件都可以在这里下载到

2.2 私服

私服是一种特殊的远程仓库，它是架设在局域网内的仓库服务，私服代理广域网上的远程仓库，供局域网内的Maven用户使用。当Maven需要下载构件的时候，它从私服请求，如果私服上不存在该构件，则从外部的远程仓库下载，缓存在私服上之后，再为Maven的下载请求提供服务。我们还可以把一些无法从外部仓库下载到的构件上传到私服上。

Maven私服的 个特性：

1.节省自己的外网带宽：减少重复请求造成的外网带宽消耗

2.加速Maven构件：如果项目配置了很多外部远程仓库的时候，构建速度就会大大降低

3.部署第三方构件：有些构件无法从外部仓库获得的时候，我们可以把这些构件部署到内部仓库(私服)中，供内部maven项目使用

4.提高稳定性，增强控制：Internet不稳定的时候，maven构建也会变的不稳定，一些私服软件还提供了其他的功能

5.降低中央仓库的负荷：maven中央仓库被请求的数量是巨大的，配置私服也可以大大降低中央仓库的压力

当前主流的maven私服：

1.Apache的Archiva

2.JFrog的Artifactory

3.Sonatype的Nexus

三、远程仓库配置

配置远程仓库将引入新的配置元素：<repositories> <repository>

在<repositories>元素下，可以使用 <repository>子元素声明一个或者多个远程仓库。

例子：

[html] view plain copy

<repositories>

<repository>

<id>jboss</id>

<name>JBoss Repository</name>

<url>http://repository.jboss.com/maven2/</url>

<releases>

<updatePolicy>daily</updatePolicy><!-- never,always,interval n -->

<enabled>true</enabled>

<checksumPolicy>warn</checksumPolicy><!-- fail,ignore -->

</releases>

<snapshots>

<enabled>false</enabled>

</snapshots>

<layout>default</layout>

</repository>

</repositories>

<updatePolicy>元素：表示更新的频率，值有：never, always,interval,daily, daily 为默认值

<checksumPolicy>元素：表示maven检查和检验文件的策略，warn为默认值

出于安全方面的考虑，有时我们要对远程仓库的访问进行认证，一般将认证信息配置在settings.xml中：

[html] view plain copy

<span style="white-space:pre"> </span><servers>

<server>

<id>same with repository id in pom</id>

<username>username</username>

<password>pwd</password>

</server>

</servers>

注：这里的id必须与POM中需要认证的repository元素的Id一致。

如何将生成的项目部署到远程仓库

完成这项工作，也需要在POM中进行配置，这里有新引入了一个元素：<distributionManagement>

distributionManagement包含了2个子元素：repository和snapshotRepository, 前者表示发布版本构件的仓库，后者表示快照版本的仓库

这两个元素都需要配置 id(该远程仓库的唯一标识)，name，url(表示该仓库的地址)

向远程仓库中部署构件，需要进行认证。配置同上

配置正确后运行： mvn clean deploy

正确的看待快照

之前我们在配置pom的时候，对于快照的配置都很谨慎，或者说很少用快照的版本，原因是它还很不稳定，极容易给我们的系统带来未知的错误，让我们很难查找。其实快照版本也并不是一无是处，快照最大的用途是用在开发的过程中，尤其是有模块依赖的时候，比如说AB两个模块同时开发，A依赖于B，开发过程中AB都是持续集成的开发，不断的修改POM文件和构建工程，这时候版本同步就成了一个很大的问题。使用快照就可以达到这一目的。

其实在快照版本在发布的过程中，Maven会自动为构件以当前时间戳做标记，有了这个时间戳，我们就可以随时找到最新的快照版本，这样也就解决刚才说的 协作开发的问题。

至于A如何检查B的更新，刚刚在讲配置的时候说过，快照配置中有一个元素可以控制检查更新的频率------updatePolicy

我们也可以使用命令行加参数的形式强制执行让maven检查更新：

mvn clean install-U

maven到底是如何从仓库中解析构件的呢？----maven从仓库解析依赖的机制

1. 当依赖的范围是system的时候，Maven直接从本地文件系统解析构件

2. 根据依赖坐标计算仓库路径后，尝试直接从本地仓库寻找构件，如果发现相应构件，则解析成功

3. 在本地仓库不存在相应的构件情况下，如果依赖的版本是显示的发布版本构件，则遍历所有的远程仓库，发现后下载使用

4. 如果依赖的版本是RELEASE或LATEST, 则基于更新策略读取所有远程仓库的元数据，将其于本地仓库的对应元数据合并后，计算出RELEASE或者LATEST的真实值，然后基于这个真实值检查本地仓库

5. 如果依赖的版本是SNAPSHOT， 则基于更新策略读取所有远程仓库的元数据， 将其与本地仓库的对应元数据合并后，得到最新快照版本的值，然后基于该值检查本地仓库或从远程仓库下载

6. 如果最后解析到的构件版本是时间戳格式的快照，则复制其时间戳格式的文件 至 非时间戳格式，并使用该非时间戳格式的构件

注：一定要记得<release> <enabled> & <snapshot> <enabled> ，对于快照也是一样

在POM的依赖声明的时候不推荐使用LATEST & RELEASE, 在Maven3中也不再支持在插件配置中使用LATEST & RELEASE， 如果不设置插件版本，那么最终版本和release一样，

maven只会解析最新的发布版本构建。

镜像