# Maven中的dependencyManagement和dependencies区别

在Maven中dependencyManagement的作用其实相当于一个对所依赖jar包进行版本管理的管理器。

在dependencyManagement下申明的dependencies，Maven并不会去实际下载所依赖的jar包，而是

在dependencyManagement中用一个Map记录了jar的三维坐标。

而被仅仅是被dependencies包裹的元素，Maven是会去仓库实际下载所需要的jar包的，而至于需要下载什么

版本的jar包就有两种判断途径：

1：如果dependencies里的dependency自己没有声明version元素，那么maven就

会倒dependencyManagement里面去找有没有对该artifactId和groupId进行过版本声明，如果有，就继承它，如果

没有就会报错，告诉你必须为dependency声明一个version

2：如果dependencies中的dependency声明了version，那么无论dependencyManagement中有无对该jar的version声明，都以dependency里的version为准。

pom.xml

//只是对版本进行管理，不会实际引入jar

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>3.2.7</version>

</dependency>

</dependencies>

</dependencyManagement>

//会实际下载jar包

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

</dependency>

</dependencies>

可以看到dependecies元素下的dependency里并没有声明版本，这是因为在manager中已经将版本管理起来了。

比如我们在项目中需要引入一个框架A的jar包：a.jar。而A框架的作者为了实现A框架又在其pom里面引入了**[spring](http://lib.csdn.net/base/17" \o "Java EE知识库" \t "_blank)**框架，那么问题就来了，如果我们的项目也要用到spring框架，那spring框架的版本应该以谁的为准呢？

这可能就是dependencyManagement的作用了，你可以在dependencyManagement中对spring进行版本声明，然后你在dependencies中添加对A框架的引用，那么实际引入的spring版本就是你在dependencyManagement中声明的版本了。

# <distributionManagement>snapshot快照库和release发布库

    在使用maven过程中，我们在开发阶段经常性的会有很多公共库处于不稳定状态，随时需要修改并发布，可能一天就要发布一次，遇到bug时，甚至一天要发布N次。我们知道，maven的依赖管理是基于版本管理的，对于发布状态的artifact，如果版本号相同，即使我们内部的镜像服务器上的组件比本地新，maven也不会主动下载的。如果我们在开发阶段都是基于正式发布版本来做依赖管理，那么遇到这个问题，就需要升级组件的版本号，可这样就明显不符合要求和实际情况了。但是，如果是基于快照版本，那么问题就自热而然的解决了，而maven已经为我们准备好了这一切。

       maven中的仓库分为两种，snapshot快照仓库和release发布仓库。snapshot快照仓库用于保存开发过程中的不稳定版本，release正式仓库则是用来保存稳定的发行版本。定义一个组件/模块为快照版本，只需要在pom文件中在该模块的版本号后加上-SNAPSHOT即可(注意这里必须是大写)，如下：

<groupId>cc.mzone</groupId>

<artifactId>m1</artifactId>

<version>0.1-SNAPSHOT</version>

<packaging>jar</packaging>

   maven会根据模块的版本号(pom文件中的version)中是否带有-SNAPSHOT来判断是快照版本还是正式版本。如果是快照版本，那么在mvn deploy时会自动发布到快照版本库中，而使用快照版本的模块，在不更改版本号的情况下，直接编译打包时，maven会自动从镜像服务器上下载最新的快照版本。如果是正式发布版本，那么在mvn deploy时会自动发布到正式版本库中，而使用正式版本的模块，在不更改版本号的情况下，编译打包时如果本地已经存在该版本的模块则不会主动去镜像服务器上下载。

      所以，我们在开发阶段，可以将公用库的版本设置为快照版本，而被依赖组件则引用快照版本进行开发，在公用库的快照版本更新后，我们也不需要修改pom文件提示版本号来下载新的版本，直接mvn执行相关编译、打包命令即可重新下载最新的快照库了，从而也方便了我们进行开发。

接下来要介绍的是如何在项目中应用snapshot和release库，应用snapshot和release库达到不同环境下发布不同的版本的目的，首先看一个pom文件的定义：

<project >

<modelVersion>4.0.0</modelVersion>

<groupId>net.aty.mybatis</groupId>

<artifactId>mybatis-demo</artifactId>

<packaging>jar</packaging>

<version>${project.release.version}</version>

<name>mybatis-demo</name>

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

<properties>

<project.release.version>0.1-SNAPSHOT</project.release.version>

</properties>

<profiles>

<profile>

<id>release</id>

<properties>

<project.release.version>0.1</project.release.version>

</properties>

</profile>

</profiles>

<!--定义snapshots库和releases库的nexus地址-->

<distributionManagement>

<repository>

<id>nexus-releases</id>

<url>

http://172.17.103.59:8081/nexus/content/repositories/releases/

</url>

</repository>

<snapshotRepository>

<id>nexus-snapshots</id>

<url>

http://172.17.103.59:8081/nexus/content/repositories/snapshots/

</url>

</snapshotRepository>

</distributionManagement>

</project>

  首先我们看到pom文件中version的定义是采用占位符的形式，这样的好处是可以根据不同的profile来替换版本信息，比如maven默认是使用0.1-SNAPSHOT作为该模块的版本。

1、如果在发布时使用mvn deploy -P release 的命令，那么会自动使用0.1作为发布版本，那么根据maven处理snapshot和release的规则，由于版本号后不带-SNAPSHOT故当成是正式发布版本，会被发布到release仓库；  
  
2、如果发布时使用mvn deploy命令，那么就会使用默认的版本号0.1-SNAPSHOT，此时maven会认为是快照版本，会自动发布到快照版本库。

      在distributionManagement段中配置的是snapshot快照库和release发布库的地址，我这里是采用nexus作为镜像服务器。对于版本库主要是id和url的配置，配置完成后就可以通过mvn deploy进行发布了，当然了，如果你的镜像服务器需要用户名和密码，那么还需要在maven的settings.xml文件中做如下配置：

<server>

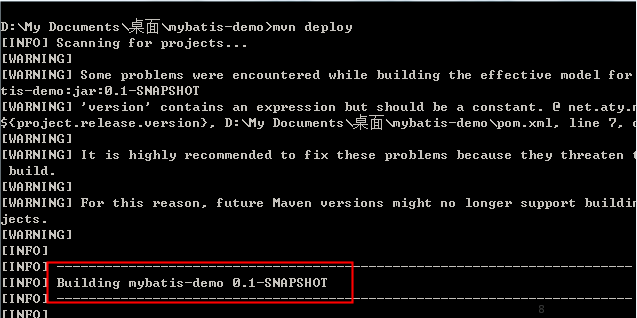
<id>nexus-snapshots</id>

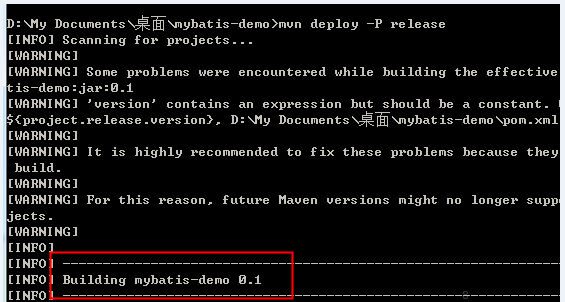
<username>admin</username>

<password>admin123</password>

</server>

注意这里配置的server的id必须和pom文件中的distributionManagement对应仓库的id保持一致，maven在处理发布时会根据id查找用户名称和密码进行登录和文件的上传发布。  
  
     我们这里通过profile的定义就可以在发布灵活切换snapshot快照版本和release正式版本了，在被依赖的组件中也可以使用profile来定义在开发阶段使用快照库，在发布阶段使用正式库的功能，只需要在不同的profile中覆盖默认的properties属性值即可。





# Maven的pom.xml文件结构之Build配置build

Build分两种，一种是在根下，一种是在profile标签下

<build>

<defaultGoal>install</defaultGoal>

<directory>${basedir}/target</directory>

<finalName>${artifactId}-${version}</finalName>

</build>

说明：

defaultGoal，执行构建时默认的goal或phase，如jar:jar或者package等

directory，构建的结果所在的路径，默认为${basedir}/target目录

finalName，构建的最终结果的名字，该名字可能在其他plugin中被改变

<resources>

资源往往不是代码，无需编译，而是一些properties或XML配置文件，构建过程中会往往会将资源文件从源路径复制到指定的目标路径。

<resource>

<targetPath>META-INF/plexus</targetPath>

<filtering>false</filtering>

<directory>${basedir}/src/main/plexus</directory>

<includes>

<include>configuration.xml</include>

</includes>

<excludes>

<exclude>\*\*/\*.properties</exclude>

</excludes>

</resource>

说明：

resources，build过程中涉及的资源文件

targetPath，资源文件的目标路径

filtering，构建过程中是否对资源进行过滤，默认false

directory，资源文件的路径，默认位于${basedir}/src/main/resources/目录下

includes，一组文件名的匹配模式，被匹配的资源文件将被构建过程处理

excludes，一组文件名的匹配模式，被匹配的资源文件将被构建过程忽略。同时被includes和excludes匹配的资源文件，将被忽略。

filters，给出对资源文件进行过滤的属性文件的路径，默认位于${basedir}/src/main/filters/目录下。属性文件中定义若干键值对。在构建过程中，对于资源文件中出现的变量（键），将使用属性文件中该键对应的值替换。

testResources，test过程中涉及的资源文件，默认位于${basedir}/src/test/resources/目录下。这里的资源文件不会被构建到目标构件中

如果想让filter中加载的properties替换src/main/resources/目录下配置文件中的${XX}占位符内容，那么就得配置以下内容：

<build>

<finalName>learn-framework</finalName>

<resources>

<resource>

<directory>src/main/resources</directory>

<filtering>true</filtering>

</resource>

</resources>

<filters>

<filter>${filters.base.path.main}/dev-filter.properties</filter>

</filters>

</build>