Maven学习笔记

## 一、Maven是什么

**1.定义**

Maven是一个用来将源代码构建成可发布的构件的工具，即将Java项目构建相关的过程自动化的工具（贯穿了整个项目生命周期，编译，测试，打包，发布...）。

**2.要解决什么问题**

Java项目构建工具要解决的问题有：这个项目需要构建什么？需要用到现在什么类库？把类库放哪里？构建中我该运行什么目标？

## 二、基本原理

**0. POM**

POM 代表项目对象模型。它是工作在Maven的基本单位。这是一个XML文件。它始终驻留在该项目 pom.xml 基本目录。POM文件描述的是构建“什么”，而不是“如何”构建。如何构建是取决于Maven的构建阶段和目标。当然，如果需要，你也可以向Maven构建阶段中添加自定义的目标。

POM 包含的项目使用Maven来构建该项目和各种配置的详细信息：用到的资源，包括源代码目录、测试代码目录等的位置，以及项目依赖的外部jar包。POM 也包含了目标和插件。在执行任务或目标时，Maven 会在当前目录中的POM。它读取POM，得到所需要的配置信息，然后执行的目标。

**1. 坐标**

Maven世界中有各种的jar包，每一个jar包都有他自己的坐标，Maven的坐标可以唯一的标识一个Java构件，Maven依赖管理是基于坐标进行的。

表格 1 Maven坐标

|  |
| --- |
| **groupId:**定义当前Maven项目隶属的实际项目，表示方式与Java包名的表示方式类似，通常与域名反向一一对应  **artifactId:**该元素定义实际项目中的一个Maven项目（模块），推荐的做法是使用实际项目的名称作为artifactId的前缀  **version:**该元素定义Maven项目当前所处的版本  **packaging :**该元素定义Maven项目的打包方式，打包方式通常与所生成构件的文件扩展名对应，默认为jar  **classifier:**该元素用来帮助定义构建输出的一些附属构件 |

**2.依赖管理**

一个复杂的项目将会包含很多依赖，也有可能包含依赖于其它构件的依赖，如果不使用依赖管理，则需要自己管理引用一大堆jar包，并且会遇到依赖冲突，无用依赖等问题。

通过Maven来进行依赖管理，只需要加上你直接依赖的那些库，Maven会隐式的把这些库间接依赖的库也加入到你的项目中（传递依赖发现）。Maven也会处理这些依赖中的冲突，同时能让你自定义默认行为，或者排除一些特定的传递性依赖。

**2.1指定依赖**

通过在POM文件中的根元素project下的dependencies可以包含一个或者多个dependency元素，以声明一个或者多个项目依赖：

表格 2 依赖元素

|  |
| --- |
| **基本坐标：**groupId，artifactId，version  **type：**依赖的类型。大部分情况下，该元素不必声明，默认是jar  **scope：**依赖的范围  **optional：**标记依赖是否可选  **exclusions：**用来排除传递性依赖 |

**2.2依赖范围**

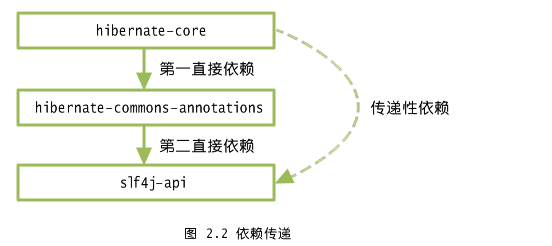
在Maven的世界中有很多种classpath，编译classpath，测试classpath，运行classpath

依赖范围就是用来控制依赖与这三种classpath的关系，scope 选项的值，决定了该依赖构件会被引入到哪一个 classpath 中。

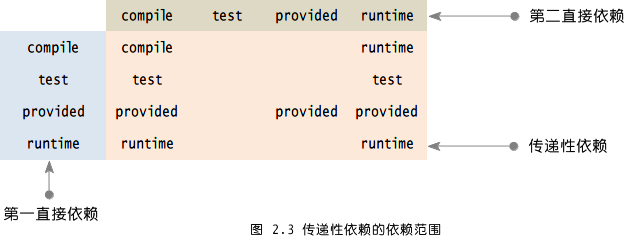
表格 3依赖范围

|  |
| --- |
| <scope>…</cope>指定范围控制哪些依赖在哪些classpath中可用，哪些依赖包含在一个应用中。  **Compile：**编译范围依赖在所有的classpath中可用，同时它们也会被打包  **Provided：**Provided依赖只有在当JDK或者一个容器已提供该依赖之后才使用；Provided依赖在编译classpath可用，它们不是传递性的，也不会被打包  **Runtime：**运行时依赖在运行和测试系统的时候需要，但在编译的时候不需要  **Test：**它们只有在测试编译和测试运行阶段可用 |

**2.3依赖范围和依赖传递**



第一直接依赖和第二直接依赖的依赖范围，决定了传递性依赖的依赖范围。



**2.4依赖冲突和排除**

通常我们不需要关心传递性依赖，当多个传递性依赖中有对同一构件不同版本的依赖，构成依赖冲突，这时按照以下原则选择依赖：

* 短路径优先：假如有以下依赖：A -> B -> C ->X(版本 1.0) 和 A -> D -> X(版本 2.0)，则优先解析较短路径的 X(版本 2.0)；
* 若相同路径，有严格区间限定的版本优先；
* 先声明优先：若路径长度相同，则谁先声明，谁被解析。

依赖排除：

针对依赖冲突中的“短路径优先”，如果我们想使用长路径的依赖怎么办呢？这时可以使用依赖排除 <exclusions> 元素，显示排除短路径依赖。在非冲突的情况下，这种方法同样有效。

**3.**[**聚合与继承**](http://blog.csdn.net/yuguiyang1990/article/details/9973795)

聚合是为了协同项目构建；继承是为了消除重复；

**聚合**：分模块开发的各个子项目要分别编译、测试、打包会很麻烦，可以建立一个单独的项目parent，这个项目中只有一个pom文件，将分开的子项目作为模块（在pom文件中使用module配置）配置到parent项目中，运行parent中的pom文件，就能自动同时一并构建各个模块。

**聚合**pom.xml的配置关键在两点：（1）packaging类型为pom；（2）<modules>模块的引入。

**继承**：分模块开发的各个子项目的pom文件中会有很大一部分的配置元素是重复的（如：作为一个大项目的各个模块，groupId应该是相同的；依赖的重复配置），可以建立一个单独的项目parent，这个项目的pom文件会抽取那些公有的重复配置供各个子模块继承（子项目的pom文件中使用parent配置需要继承的pom项目信息）。

**区别** ：

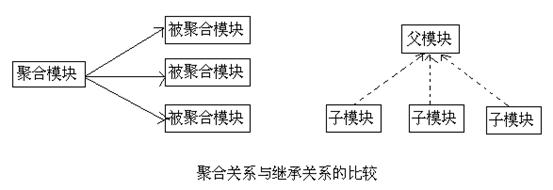
**1**.对于聚合模块来说，它知道有哪些被聚合的模块，但那些被聚合的模块不知道这个聚合模块的存在。

2.对于继承关系的父 POM来说，它不知道有哪些子模块继承与它，但那些子模块都必须知道自己的父 POM是什么。

**共同点** ：

1.聚合 POM与继承关系中的父POM的 packaging都是pom；

2.聚合模块与继承关系中的父模块除了 POM之外都没有实际的内容。



在现有的实际项目中一个 POM既是聚合POM，又是父 POM，这么做主要是为了方便

## 4. DependencyManagement

父pom中的dependencies 元素影响项目的依赖项，所有声明在dependencies里的依赖都会自动引入，并默认被所有的子项目继承。

而dependencyManagement 中的 dependencies 元素只表明依赖项版本的优先选择，并不影响项目的依赖项；

在整个项目中，对于某个构件（如mysql）的依赖配置只有一种，这样就能避免引入不同版本的依赖，避免依赖冲突。Maven提供的dependencyManagement就是用来消除重复和避免冲突。dependencyManagement实际上不会真正引入任何依赖，dependencies才会。但是，当父模块中配置了某个依赖之后，子模块只需使用简单groupId和artifactId就能自动继承相应的父模块依赖配置。另外如果子项目中指定了版本号，那么会使用子项目中指定的jar版本。这样的好处是子模块可以有选择行的继承，而不需要全部继承。

正确的做法是：

1. 在父模块中使用dependencyManagement配置依赖；

2. 在子模块中使用dependencies添加groupId和artifactId依赖。

 建议依赖都放在父POM中的dependencyManagement、一个是减少配置、二个是方便管理、比如版本冲突就是很常见的问题。通过dependencyManagement+变量的方式统一管理、更安全高效。

## 问题：

1. 父项目的Pom的dependencies依赖，如果子项目重写的版本号，那会引入哪一个？
2. Maven中冲突消解的单位是项目还是模块？

## 三、基本使用

**1、maven生命周期**

**maven有三套生命周期：**

1.clean       清理项目

2.default     构建项目

3.site           建立项目站点

每套生命周期都包含了一些阶段，这些阶段是有序的，后面的阶段依赖前面的阶段。

**2、maven archetype**

可用项目骨架有：

maven-archetype-archetype

maven-archetype-j2ee-simple

maven-archetype-mojo

maven-archetype-portlet

maven-archetype-profiles

maven-archetype-quickstart

maven-archetype-simple

maven-archetype-site

maven-archetype-site-simple

maven-archetype-webapp

***待学习：自定义脚手架***

**3、常用命令**

**3.1 Maven环境变量配置：**

变量名：M2\_HOME 变量值：D:\bin\apache-maven-2.2.1 (安装路径)

在path值后面追加 %M2\_HOME%\bin; (注意要加分号)

测试： mvn –version

**3.2 maven常用命令为**

mvn archetype:create ：创建 Maven 项目

mvn compile ：编译源代码

mvn test-compile ：编译测试代码

mvn test ： 运行应用程序中的单元测试

mvn site ： 生成项目相关信息的网站

mvn clean ：清除目标目录中的生成结果

mvn package ： 依据项目生成 jar 文件

mvn install ：在本地 Repository 中安装 jar

mvn eclipse:eclipse ：生成 Eclipse 项目文件

**3.3生成项目**

建一个 JAVA 项目 ： mvn archetype:create -DgroupId=com.demo -DartifactId=App

建一个 web 项目: mvn archetype:create -DgroupId=com.demo -DartifactId=web-app -DarchetypeArtifactId=maven-archetype-webapp

archetype 是一个内建插件，他的create任务将建立项目骨架，archetypeArtifactId项目骨架的类型，DartifactId 项目名称。

**3.4 Setting.xml 配置文件**

localRepository：本地库存位置，jar包保存位置，默认${user.dir}/.m2/repository

offline: true/false,如不想每次编译都去查找远程中心库，就设置为true，前提是已下载好包

Servers：配置特定的用户

<servers>

<server>

<id/><username/><password/><privatekey/><pssphrase/>

<filePermissions/><directoryPermissions/><configuration/>

</server>

</servers>

id：用于匹配distributionManagement库id

username,password：用于登陆此服务器的用户名和密码

privateKey，passphrase：设置private Key，及passphrase

filePermissions,directoryPermissions：访问权限664/775

mirrosrs：镜像库，用于添加其他库

<mirrors>

<miror>

<id/><name/><url/><mirroOf/>

</mirror>

</mirrors>

id,name：唯一的标志，用来区别镜像

url：镜像的url

mirrorOf：此镜像指向的服务id

Proxies：代理设置，用于无法直接访问中心库的配置

<proxies>

<proxy>

<id/><active/><protocol/><host/><port/><username/><password/><nonProxyHosts/>

</proxy>

</proxies>

id：代理的标志 active：是否激活代理

protocol，host，port：protocol：//host：port代理

username，password：用户名密码 nonProxyHosts：不需要代理的host

repositories和pluginRepositonries定义本地开发库 用于release发布

<repositories>

<repository>

<id/><name/><url/>

<releases>

<enabled/><updatePolicy/><checksumPolicy/>

</releases>

<snapshots>

<enabled/>

</snapshots>

<layout/>

</repository>

</repositories>

-----------------------------------------------------------------------------------------------

**3.5 Pom.xml 配置文件**

在pom.xml中定义jar包版本还有依赖，管理jar文件。

包括配置文件；开发需要遵循的规则，缺陷管理系统，组织和licenses,项目的依赖性

groupId：项目或组织唯一标志，并生成项目路径

artifactId：项目名称 version：项目版本

packaging：打包的机制，如jar，pom，war....

maven中继承的定义

<project>

<modelVersion/><groupId/><version/><packaging/>

</project>

父项目packaging类型需要为pom，用于parent和合成多个项目，子项目为

<parent>

<groupId/><artifctId/><version/>

</parent>

Modules合成

<modules>

<module>tc-dient</moduble>

<module>tc-server</module>

</modules>

build设置：用于编译设置，包括两个主要元素，build,report

<project>

<project>

<build/>

<profies>

<profile>

<build/>

</profile>

</profies>

</profiles>

</project>

build-plugin插件

<project>

<build>

.....

<plugins>

<plugin>

<groupId/><artifactId/><version/>

<extensions/><executions/>

</plugin>

</plugins>

</build>

</project>

extensions：true/false 是否装载插件扩展，默认为false

inherited：true/false 此插件配置是否应用于poms(继承)

configuration：指定插件配置

dependencies：插件需要依赖的包

executions：用于配置execution目标，一个插件可以有多个目标

build-resources资源，项目中需要指定的资源如：log4j.properties

<project>

<build>

<resources>

<resource>

<targetPath/><filtering/><directory/>

<includes>

<include/>

</includes>

<excludes>

<exclude/>

</excludes>

</resource>

</resources>

<testResources>

...

</testResources>

....

</build>

</project>

resources：列表，包括所有的资源

targetPath：指定目标路径，用于放置资源，用于build

filtering：是否替换资源中属性placehold

directory：资源所在的位置

includes：样式，包括资源

excludes：排除的资源

testResources：测试资源列表

dependencies：依赖关系

<dependencies>

<dependency>

<groupId/><artifactId/><version/>

<type/><scope/><optional/>

</dependency>

</dependencies>

groupId,artifactId,version：描述了依赖的项目唯一标志

使用以下的命令安装：

mvn install:install-file –Dfile=non-maven-proj.jar –DgroupId=some.group –DartifactId=non-maven-proj –Dversion=1

创建自己的库,并配置，使用deploy:deploy-file 设置此依赖范围为system，定义一个系统路径。不提倡。

type：相应的依赖产品包形式，如jar，war

scope：用于限制相应的依赖范围，包括以下的几种变量：

compile：默认范围，用于编译

provided：类似于编译，但支持你期待jdk或者容器提供，类似于classpath

runtime：在执行时，需要使用

test：用于test任务时使用

system：需要外在提供相应得元素。通过systemPath来取得

systemPath: 仅用于范围为system。提供相应的路径

optional: 标注可选，当项目自身也是依赖时。用于连续依赖时使用