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

**1.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 包含的项目使用Maven来构建该项目和各种配置的详细信息。  
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**

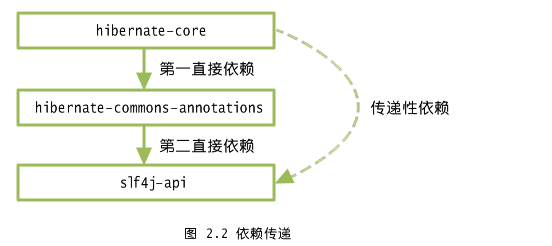
在Maven的世界中有很多种classpath，编译classpath，测试classpath，运行classpath

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。

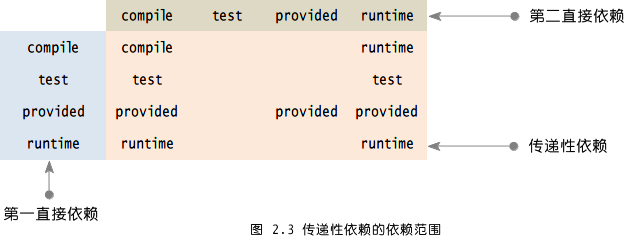
表格 3依赖范围

|  |
| --- |
| <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。



**2.3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；

**区别** ：

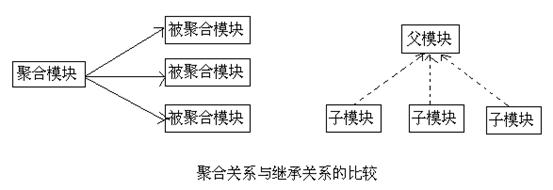
**1**.对于聚合模块来说，它知道有哪些被聚合的模块，但那些被聚合的模块不知道这个聚合模块的存在。

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

**3.1 dependencyManagement**

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。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添加依赖。

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。

## 使用方法