# Maven坐标与依赖

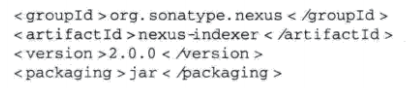
## 一 maven坐标

### 1.1 何为maven坐标

Maven定义了一组规则：世界上任何一个组件都可以使用maven坐标唯一标识,maven坐标的元素包括groupId,artifactId,version,packaging,classifier。只要我们提供正确的坐标元素，maven就能找到对应的构建。

### 1.2 坐标详情

#### 1.2.1 坐标定义



#### 1.2.2 坐标元素

##### (1) groupId

定义当前maven项目律属的实际项目,maven项目与实际项目不一定是一对一的关系，比如springframe-work框架下面就有spring-core,spring-bean等的项目构成。

##### artifactId

改元素定义实际项目中的一个maven模块,推荐的做法是以实际的项目名称作为作为这个maven模块的前缀。比如spring-core

##### Version

定义改maven项目模块所属的版本.包括版本的后缀snapshot标识正在更新，realease标识稳定的版本

##### Packaging

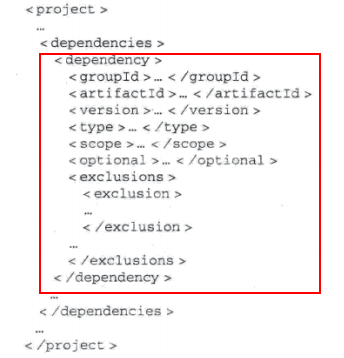
定义该maven项目的打包方式，如果是jar则打包成一个执行工具，如果是war包则打包成一个maven项目。

##### Classifier

该元素用来定义构建输出的附属构建。不能直接定义classfier，因为项目的附属构建不是项目直接生成的，而是有maven插件生成的。

## 二 maven依赖

### 2.1 依赖配置



#### 2.1.1 groupId,versionId,artifactId

依赖的基本坐标，对于任何一个依赖来说,基本坐标才是最重要的，maven根据坐标才能找到需要的依赖。

#### 2.1.2 type

依赖的类型，打部分情况都为jar,当然也不排除package类型的。

#### 2.1.3 scope

依赖范围

#### 2.1.4 optional

标记依赖是否可选

#### 2.1.5 exclusion

用来排除传递依赖

### 2.2 依赖范围

#### 2.2.1compile

编译依赖范围。如果没有指定，默认使用该依赖范围。典型的是spring-core在编译，测试，运行的时候都需要依赖。

#### 2.2.2test

测试依赖范围。对编译和测试classpath有效，典型的例子如Junit它只在编译测试代码及运行测试的时候需要。

#### 2.2.3provided

已提供依赖范围，使用此依赖范围的依赖对编译测试classpath有效。但在运行时无效，典型的例子是servlet-api,编译和测试项目的时候需要该依赖。但在运行项目的时候由于容器已经提供了，就不需要maven重复的引入了。

#### 2.2.4runtime

运行时依赖，使用此依赖范围对于测试运行classpath有效。在编译主代码时无效。典型的如jdbc驱动，只在测试执行或者运行项目的时候才需要具体的JDBC驱动。

#### 2.2.5system

依赖范围与其他的四种依赖基本一致，可以指定本地的依赖路径，可以通过systemPath来指定。不是通过maven仓库来解析的.往往需要与本机绑定，可能造成构建的不可移植。要谨慎使用。

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-api</artifactId>

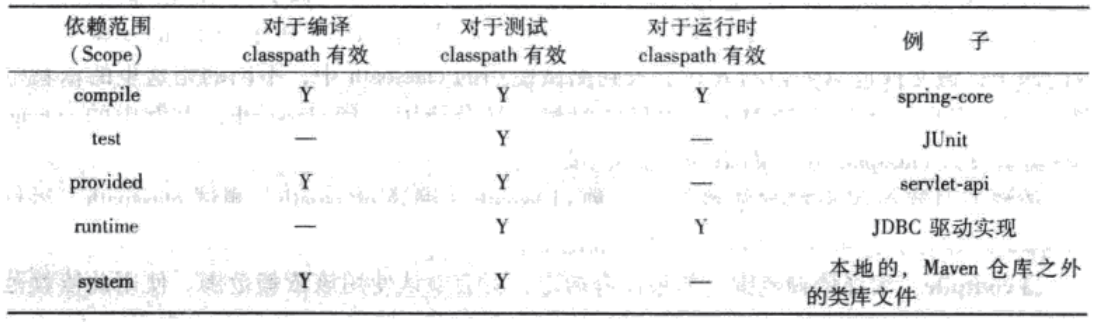
<version>2.10.0</version>

<scope>system</scope>

<systemPath>../../..//</systemPath>

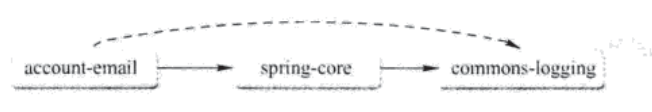
</dependency>

#### 2.2.6 依赖范围与ClassPath的关系



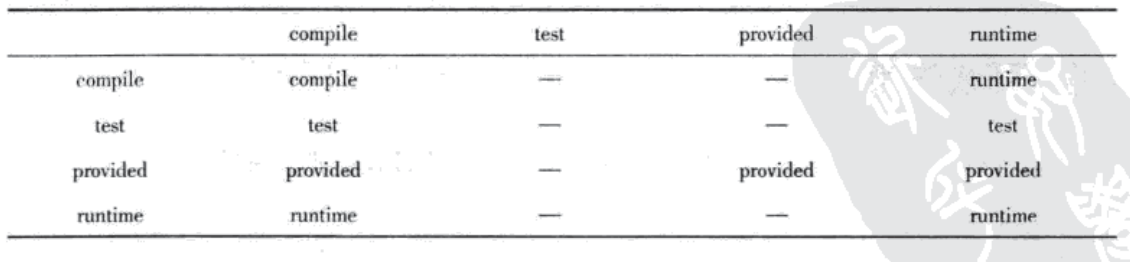
### 2.3 传递依赖

#### 2.3.1什么是传递依赖



如上图所示spring-core依赖于commons-logging所以account-email依赖于spring-core所以就产生了传递依赖account-email依赖于commons-logging，这就是传递依赖。

#### 2.3.2 传递依赖关系



规律：当第二直接依赖范围是Compile的时候，传递依赖范围与第一直接依赖范围一致。当第二直接依赖范围是test的时候，依赖不会传递。当第二直接依赖范围为provided的时候，只传递第一直接依赖范围为provided的依赖。当第二直接依赖范围为runtime的时候除了compile的依赖跟随第二直接依赖，其他的依赖不变。

#### 2.3.3 依赖调解

第一原则:路径最近者优秀

A->B->C->X(1.0) A->B->X(2.0)

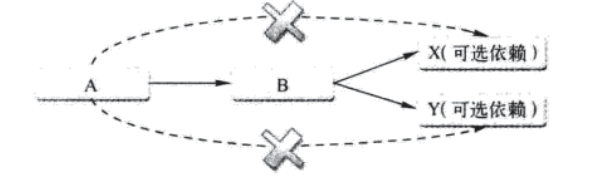
如上所示选择依赖X(2.0)

第二原则:第一申明者优先

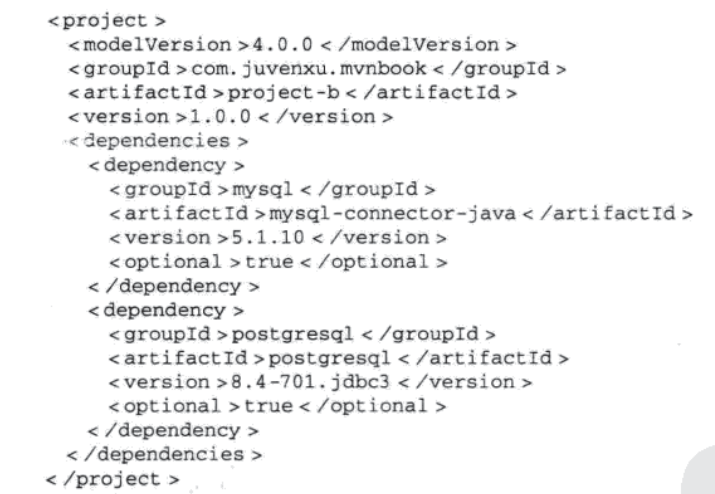
1. >B->C(1.0) A->B->C(2.0)

如上图所示取C(1.0)

#### 2.3.4 可选依赖



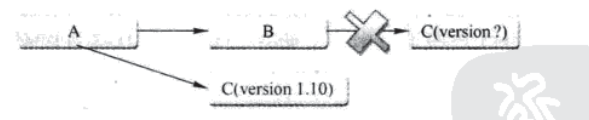
比如支持两种数据库X,Y依赖时只选择一种数据库。



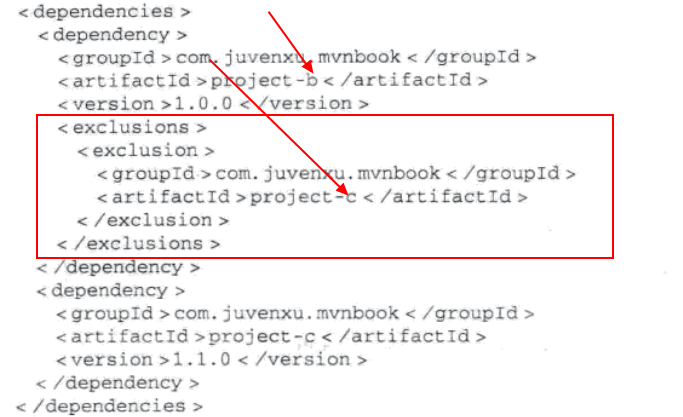
例如上面的配置

假如B对于X,Y的依赖是可选的。那么这个依赖中加上了<option>字段标识可选的。此时依赖是不会传递的。如果A依赖B，那么就得在A中显示声明这种依赖。

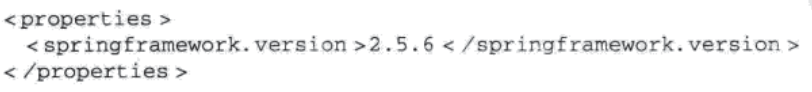
#### 2.3.5 排除依赖



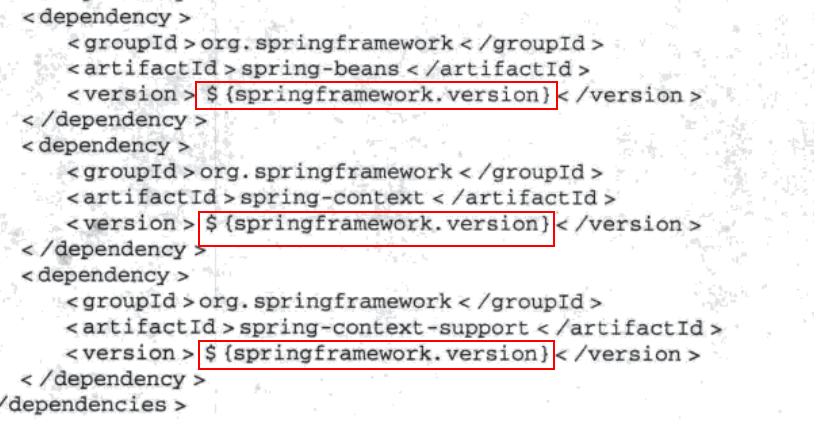
如上图所示假设A依赖B不想引入C的依赖就可以使用排除依赖



#### 2.3.6归类依赖



如上图所示可以配置公共的版本信息后面只要是同一个项目的版本问题都可以使用同一的依赖。



### 2.4 依赖优化

Mvn dependency:list 查看当前项目已解析的依赖。

Mvn dependency:tree 查看依赖树。

Mvn dependency:analyze 分析当前项目的依赖。