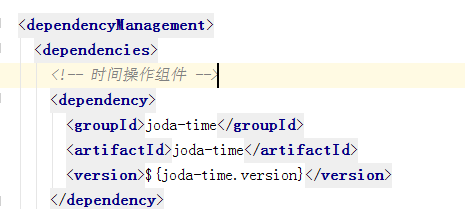
# 依赖管理dependencyManagement

只声明不引用

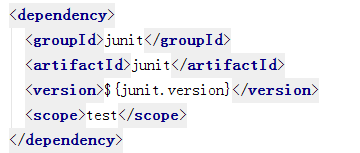


# 依赖

## 概述

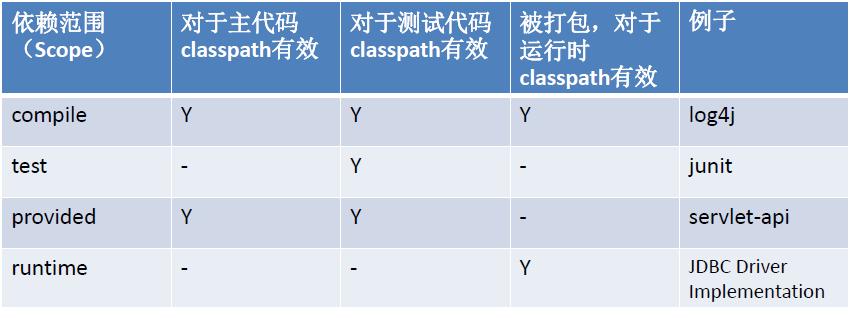
就是对项目中jar 包的管理。可以在pom文件中定义jar包的GAV坐标，管理依赖。

依赖声明主要包含如下元素：



|  |
| --- |
| <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.10</version>  <scope>test</scope>  </dependency>    </dependencies> |

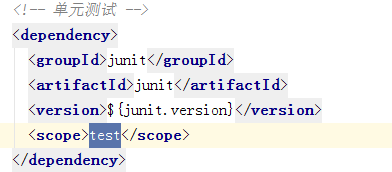
## 范围scope（依赖范围）



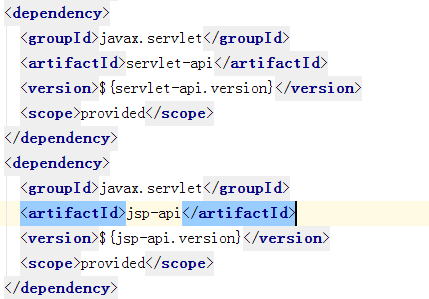
其中依赖范围**scope** 用来控制依赖和编译，测试，运行的classpath的关系. 主要的是三种依赖关系如下：

1.compile： 默认编译依赖范围（可省略）。对于编译，测试，运行三种classpath都有效

2.test：测试依赖范围。只对于测试classpath有效 junit



3.provided：已提供依赖范围。对于编译，测试的classpath都有效，但对于运行无效。因为由容器已经提供，例如servlet-api socket-api



4.runtime:运行时提供。例如:jdbc驱动

## 依赖传递

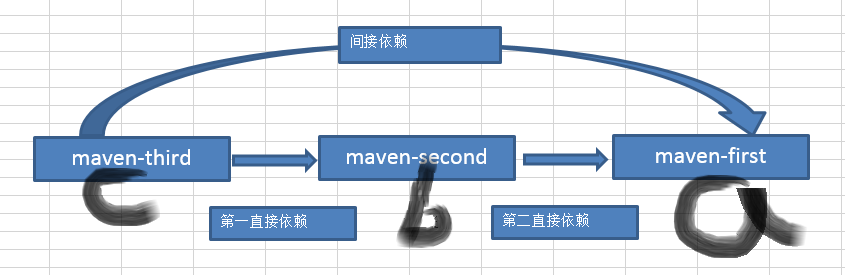
### 直接依赖和间接依赖

如果B中使用A，C中使用B，则称B是C的**直接依赖，**而称A是C的**间接依赖**。

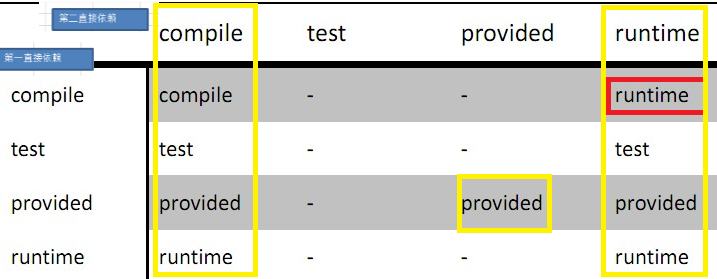
C->B B->A

C直接依赖B

C间接依赖A



### 依赖范围对传递依赖的影响



中间的交叉单元格表示传递性依赖范围。

总结：

* 当第二依赖的范围是compile的时候，传递性依赖的范围与第一直接依赖的范围一致。
* 当第二直接依赖的范围是test的时候，依赖不会得以传递。
* 当第二依赖的范围是provided的时候，只传递第一直接依赖范围也为provided的依赖，且传递性依赖的范围同样为 provided；
* 当第二直接依赖的范围是runtime的时候，传递性依赖的范围与第一直接依赖的范围一致，但compile例外，此时传递的依赖范围为runtime；

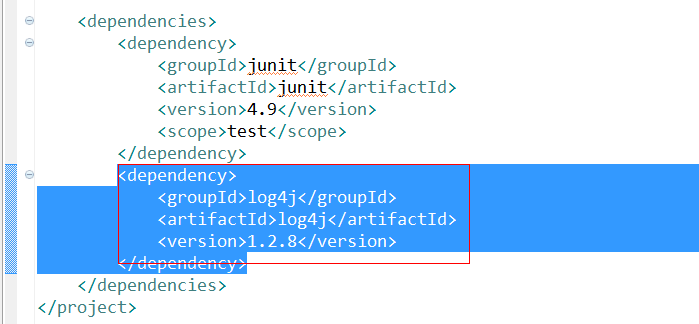
## 依赖冲突

在maven中存在两种冲突方式：一种是跨pom文件的冲突，一致是同一个pom文件中的冲突。

### 跨pom文件：直接与间接依赖情况

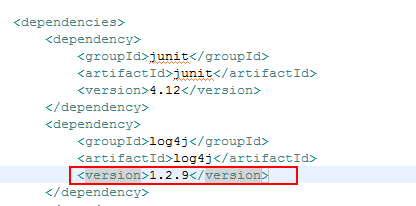
如果直接与间接依赖中包含有同一个坐标不同版本的资源依赖，以直接依赖的版本为准（就近原则）

1、Maven-first工程中依赖log4j-1.2.8版本



**那么maven-third中依赖的就是log4j-1.2.8**

2、maven-second工程中依赖log4j-1.2.9版本

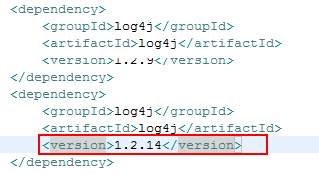


**那么maven-third中依赖的就是log4j-1.2.9，因为它直接依赖的maven-second项目中依赖的就是1.2.9版本**

### 同一个pom文件

如果直接依赖中包含有同一个坐标不同版本的资源依赖，以配置顺序下方的版本为准（就近原则）

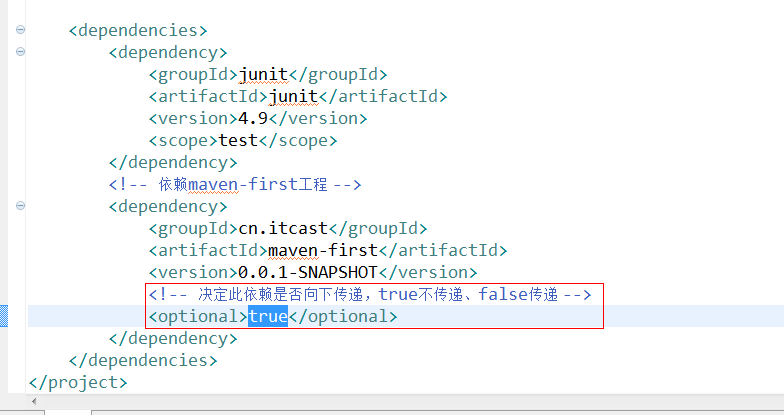
Maven-second中依赖log4j-1.2.9和log4j-1.2.14，此时log4j-1.2.14版本生效。



## 可选依赖 -父 optional

<optional> true/false 是否可选，也可以理解为是否向下传递。

在依赖中添加optional选项决定此依赖是否向下传递，如果是true则不传递，如果是false就传递，默认为false。



## 排除依赖 -子

|  |
| --- |
| <exclusions>  <exclusion>  <groupId>cn.itcast.maven</groupId>  <artifactId>maven-first</artifactId>  </exclusion>  </exclusions> |

排除依赖包中所包含的依赖关系，**不需要添加版本号**。

如果在本次依赖中有一些多余的jar包也被传递依赖过来，如果想把这些jar包排除的话可以配置exclusions进行排除。

