## 一、背景

maven一般用来管理jar包依赖，构建项目，其他类似优秀的工具有Ant、Gradle  
  
mavne是基于项目对象模型（POM），可以通过一小段描述信息来管理项目的构建、报告和文档的软件项目管理工具  
目前很多开源的项目都使用maven来构建，如struts

maven主要解决如下问题

管理jar包的依赖范围

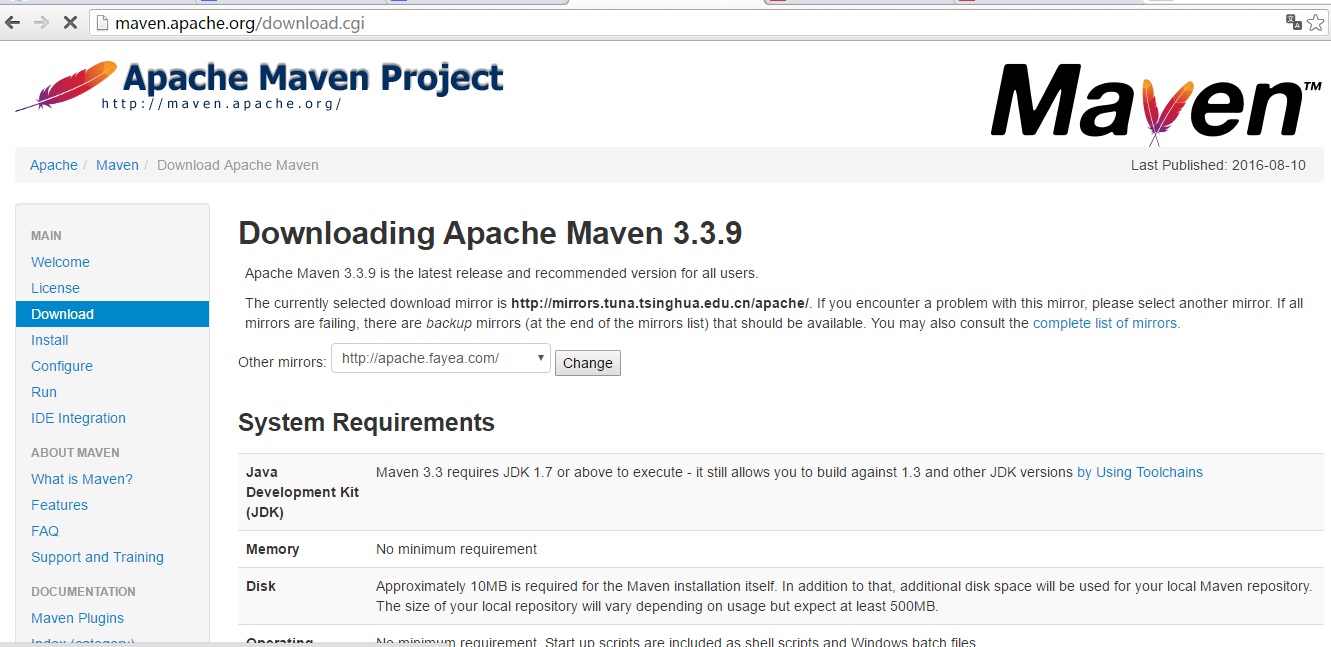
管理jar包的依赖传递  
管理jar包的依赖冲突

管理各个模块的聚合和继承

maven项目一般是多个模块的，利用pom.xml文件可以方便的管理各个项目中的关联问题

## 二、下载

http://maven.apache.org/



下载后，解压配置好相应的settings.xml和环境变量就可以使用了

## 三、maven配置

### 3.1 配置环境变量

M2\_HOME -->D:\maven\apache-maven-3.3.9  
path---> %M2\_HOME%\bin;  
cmd-[**测试**](http://lib.csdn.net/base/softwaretest)是否配置成功  
mvn -v

### 3.2生成Maven本地仓库

mvn help:system 命令然后回车，等其执行完后就可以在 C:\Users\Admin\.m2\repository 看到 Maven 下载的一些文件。  
注: 本步必须在联网情况下执行，该本地仓库的路径可以在settings.xml文件中配置

### 3. 3 用户自定义配置

修改maven的配置文件，使其符合我们的使用需要  
  
找到maven的安装目录 conf-->settings.xml  
  
用户自定义配置。  
Maven有一个全局配置文件为 Maven根目录/conf/settings.xml 文件（比如我的就是 C:\tools\apache-maven-3.2.5\conf\settings.xml），  
Maven默认是使用此配置文件，所有用户共享此配置。但是推荐每一个用户配置自己的配置文件，  
防止无意思影响系统中其他用户，只需要将全局的配置文件复制到用户目录下的 .m2 文件夹  
即可(我的当前用户是 Admin， 所以复制后为 C:\Users\Admin\.m2\settings.xml )。  
（如果没有 .m2 文件夹 请先执行上一步，maven会在当前用户的目录下生成 .m2 文件夹）。

### 3.4 maven项目的目录结构



mian目录下用于写项目代码，test下写测试代码，-resource下存放一些静态资源等

### 3.5 mavn常用命令

mvn -v :查看mvn版本

mvn complie :编译mvn项目  
mvn test :测试

mvn test-compile :编译测试程序  
mvn package :打包可以将一个项目快速打成jar包，war包，zip包，pom包等  
mvn clean :删除编译产生的target  
mvn install :安装jar包到本地仓库

mvn site :生成目标站点

mvn site-depoly:生成目标站点并发布

mvn install:install-file -DgroupId=<groupId> -DartifactId=<artifactId> -Dversion=1.0.0 -Dpackaging=jar -Dfile=<myfile.jar> :安装指定文件到本地仓库  
IMG_256mvn install:install-file -Dfile=jar包的位置 -DgroupId=上面的groupId -DartifactId=上面的artifactId -Dversion=上面的version -Dpackaging=jar

mvn install:install-file -Dfile=/home/caojx/下载/jodconverter包/jodconverter-core-3.0-beta-4/jodconverter-core-3.0-beta-4-sources.jar -DgroupId=org.artofsolving.jodconverter -DartifactId=jodconverter-core -Dversion=3.0-beta-4 -Dpackaging=jar

mvn help:effective-pom :查看实际pom信息

mvn dependency:analyze 或 mvn dependency:tree :分析项目的依赖信息

mvn eclipse:eclipse : 生成eclipse项目文件

mvn help:help 或 mvn help:help -Ddetail=true :查看帮助信息

mvn <plug-in>:help，比如：mvn dependency:help 或 mvn ant:help :查看插件的帮助信息

mvn archetype:create -DgroupId=com.mycompany.example -DartifactId=Example :创建一个简单的Java工程

mvn archetype:create -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-webapp -DgroupId=com.mycompany.app -DartifactId=my-webapp :创 建一个java的web工程

如在cmd中可以使用如下命令快速构建maven目录结构（首先建立一个工程文件夹）

1. mvn archetype:generate 按照提示进行  
建议命名风格按照如下方式进行

2. mvn archetype:generate -DgroupId=组织名，公司网址反写+项目名 -DartifactId=项目名+模块名 -Dversion=版本号 -Dpackage=代码所在的包

### 3.6 maven的生命周期

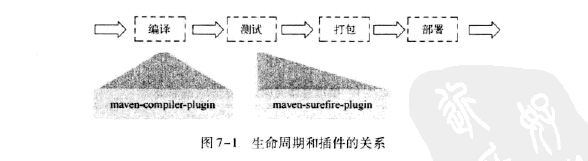
原文：<http://www.cnblogs.com/tenghoo/p/maven_life_cycle.html>

推荐文章：http://www.cnblogs.com/luotaoyeah/p/3819001.html

1.Maven的生命周期

Maven的生命周期就是对所有的构建过程进行抽象和统一。包含了项目的清理、初始化、编译、测试、打包、集成测试、验证、部署和站点生成等几乎所有的构建步骤。

Maven的生命周期是抽象的，即生命周期不做任何实际的工作，实际任务由插件完成，类似于设计模式中的模板方法。



2.三套生命周期  
Maven有三套相互独立的生命周期，分别是clean、default和site。每个生命周期包含一些阶段（phase），阶段是有顺序的，后面的阶段依赖于前面的阶段。

**1、clean生命周期**：清理项目，包含三个phase。

1）pre-clean：执行清理前需要完成的工作

2）clean：清理上一次构建生成的文件

3）post-clean：执行清理后需要完成的工作

**2、default生命周期**：构建项目，重要的phase如下。

1）validate：验证工程是否正确，所有需要的资源是否可用。  
2）compile：编译项目的源代码。    
3）test：使用合适的单元测试框架来测试已编译的源代码。这些测试不需要已打包和布署。  
4）Package：把已编译的代码打包成可发布的格式，比如jar。  
5）integration-test：如有需要，将包处理和发布到一个能够进行集成测试的环境。  
6）verify：运行所有检查，验证包是否有效且达到质量标准。  
7）install：把包安装到maven本地仓库，可以被其他工程作为依赖来使用。  
8）Deploy：在集成或者发布环境下执行，将最终版本的包拷贝到远程的repository，使得其他的开发者或者工程可以共享。

**3、site生命周期**：建立和发布项目站点，phase如下

1）pre-site：生成项目站点之前需要完成的工作

2）site：生成项目站点文档

3）post-site：生成项目站点之后需要完成的工作

4）site-deploy：将项目站点发布到服务器

#### 3.命令行和生命周期

各个生命周期相互独立，一个生命周期的阶段前后依赖。

举例如下：

1、mvn clean

调用clean生命周期的clean阶段，实际执行pre-clean和clean阶段

2、mvn test

调用default生命周期的test阶段，实际执行test以及之前所有阶段

3、mvn clean install

调用clean生命周期的clean阶段和default的install阶段，实际执行pre-clean和clean，install以及之前所有阶段

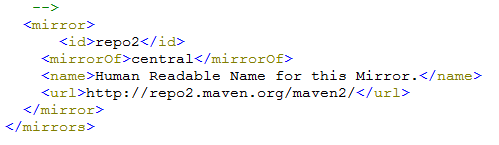
### 3.7settings.xml文件配置

设置本地仓库的路径（一般 使用自定义的）



不设置该路径，默认会在存放在.m2目录下如---》H:\Users\Mcaox\.m2\repository

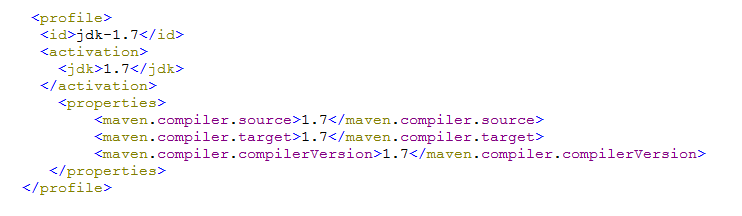
配置远程仓库



maven会去本地仓库中找需要的jar,如果不存在，则取远程仓库中找，将jar下载到本地然后供程序使用,上边使用的是公共的镜像仓库

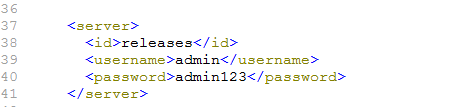
还有其他的一些，配置可以参考http://blog.csdn[.NET](http://lib.csdn.net/base/dotnet)/zuoluoboy/article/details/20007819

默认jdk配置



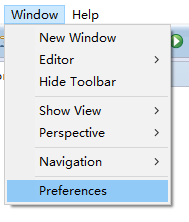
配置为jdk1.7

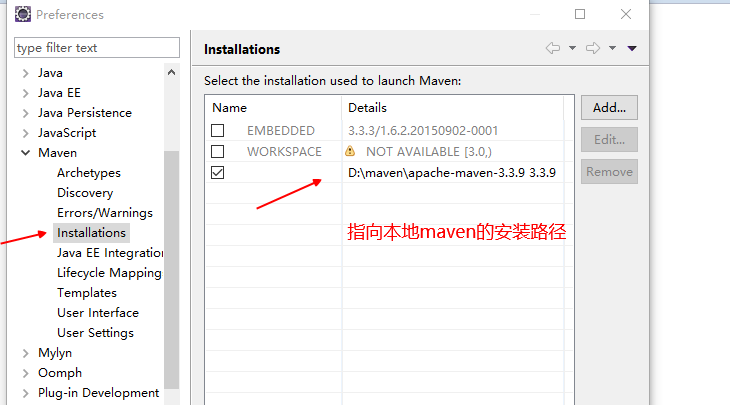
如果为私有的仓库可能还需要配置server

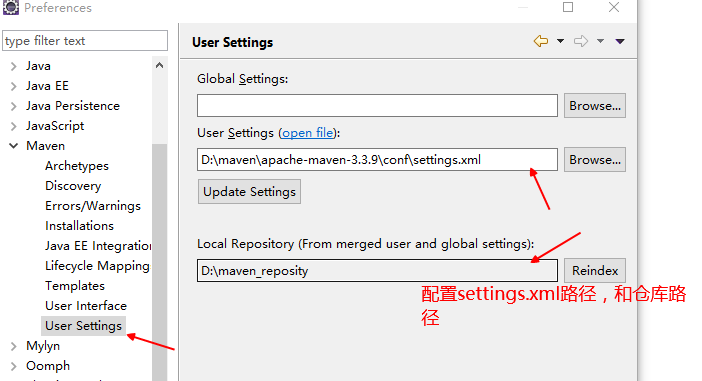


## 四、eclipse中配置maven

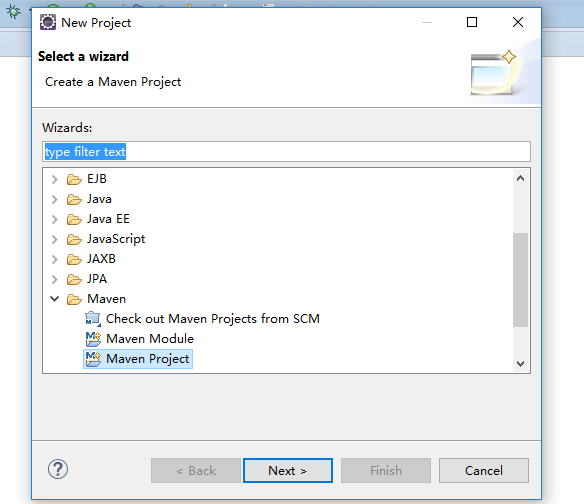
### 4.1配置安装到eclipse



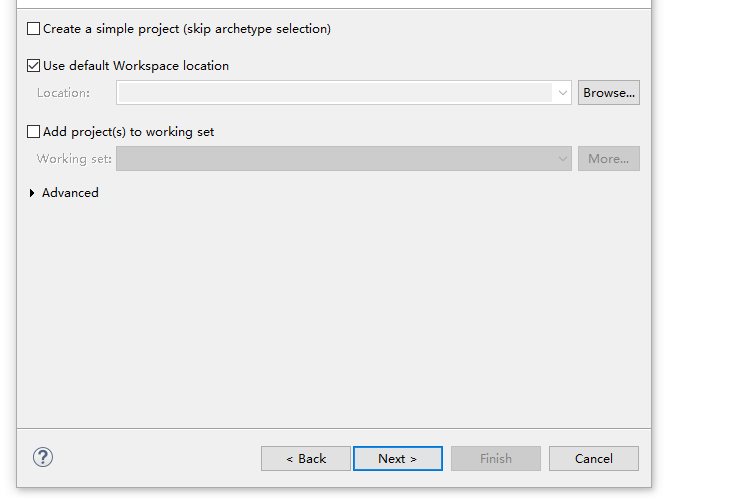




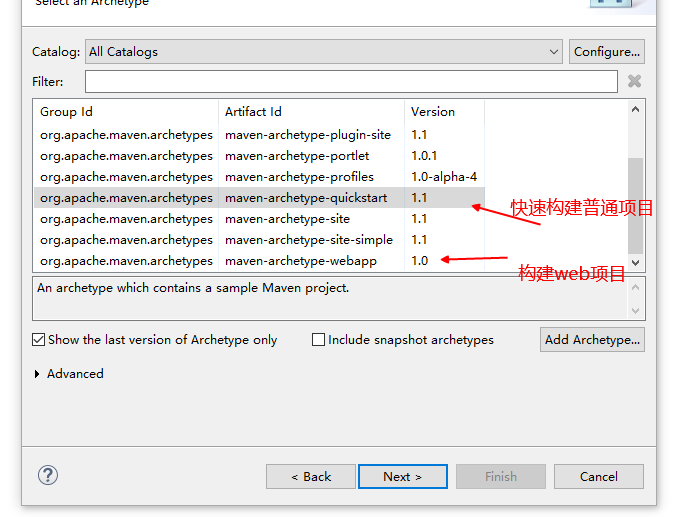
4.2创建maven工程



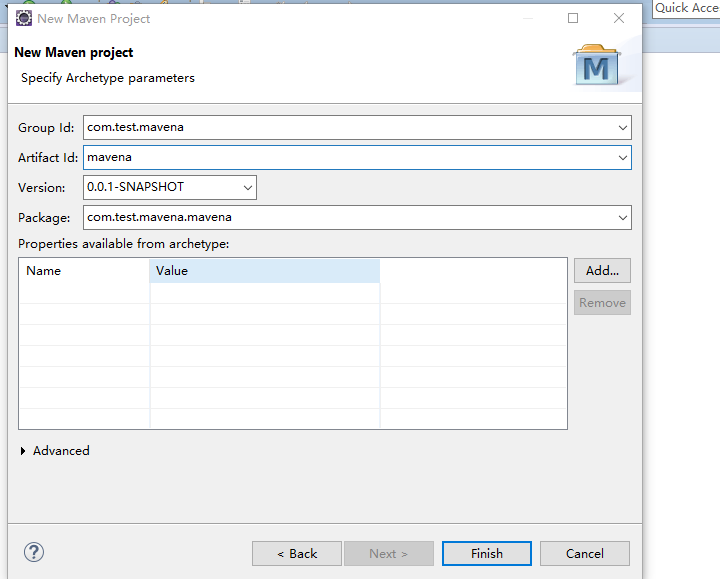
1选择 maven project 如果需要建立多模块工程选择maven module



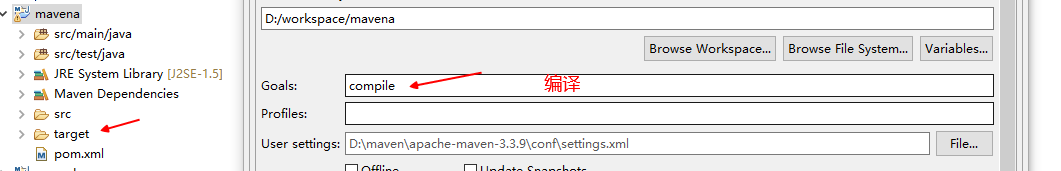
2 next



3.选择构建普通工程



4 .输入相应的坐标artifact id将会作为工程名



5. 选择工程右键run as--->maven build...  --> goals为命令 （compile,test,package,clean,install）等，如上图compile编译，就会生成target目录，里边存放生成的字节码文件，和测试报告文件等。

## pom.xml文件解析

每一个maven项目模块都会有一个pom.xml文件，里边可以配置项目的jar包依赖，模块依赖,比如添加jetty或tomcat插件等等

我们先看以下pom.xml的具体结构，之后再进行说明

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. < modelVersion**>**4.0.0**</modelVersion>**
4. **<groupId>**项目标识（说明该项目属于那个实际项目）**</groupId>**
5. **<artifactId>**项目名+模块名**</artifactId>**
6. <!--
7. 第一个0大版本号
8. 第二个0分支版本号
9. 第三个0小版本号
10. 0.0.1-SNAPSHOT 快照版
11. alpha内测版
12. beta公测版
13. release稳定版
14. GA正式版
15. --**>**
16. **<version>**0.0.1-SNAPSHOT**</version>**
18. <!--指定maven项目的打包方式   会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。
19. 默认是 jar
20. war zip pom
21. --**>**
22. **<packaging></packaging>**
24. <!--项目描述名-->
25. **<name></name>**
26. <!--项目地址-->
27. **<url></url>**  会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。
28. <!--描述信息-->
29. **<description></description>**
30. <!--开发人员-->
31. **<developers></developers>**
32. **<licenses></licenses>**
33. **<organization></organization>**
35. **<dependencies>**
36. **<dependency>**

**<groupId>**junit**</groupId>**

1. **<artifactId>**junit**</artifactId>**
2. **<version>**4.10**</version>**
3. **<type>**jar**</type>**
4. <!-- 指定依赖范围，说明该依赖在哪里有用 -->
5. **<scope>**test**</scope>**
6. <!--设置依赖是否可选默认是false,
7. 如果是false子项目默认是继承的，如果是true子项目必须显示
8. 引入改依赖  会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。
9. <dependency>
10. <groupId>junit</groupId>
11. <artifactId>junit</artifactId>
12. <version>3.8.1</version>
13. <scope>test</scope>
14. </dependency>  --**>**
15. **<optional></optional>**
16. <!-- 排除依赖传递
17. A-》B-》C
18. A依赖于B，B依赖于C
19. A传递依赖于C
20. 如果A不想依赖于C，只依赖于B则使用该标签排除对C的传递依赖
21. --**>**
22. <dependency>
23. <groupId>junit</groupId>
24. <artifactId>junit</artifactId>
25. <version>3.8.1</version>
26. <scope>test</scope>
27. </dependency>  **<exclusions>**
28. **<exclusion>**
29. **</exclusion>**
30. **<exclusions>**
31. **</dependency>**
33. **</dependencies>**
35. <!--依赖管理-->
36. **<de <dependency>**
37. **<groupId>junit</groupId>**
38. **<artifactId>junit</artifactId>**
39. **<version>3.8.1</version>**
40. **<scope>test</scope>**
41. **</dependency>pendencyManagement>**
42. <!--
43. 依赖来不会引入到实际的依赖中
44. 主要定义在父模块中，供子模块继承用
45. --**>**
46. **<dependencies>**
47. **<dependency>**
48. **</dependency>**
49. **</dependencies>**
50. **</dependencyManagement>**
51. <!--插件列表-->
52. **<build>**
53. **<plugins>**
54. **<plugin>**
55. **<groupId></groupId>**
56. **<artifactId></artifactId>**
57. **<version></version>**
58. **</plugin>**
59. **</plugins>**
60. **</build>**
61. <!--用于子模块中对父模块中的pom的继承-->
62. **<parent></parent>**
63. <!--用于聚合多个模块的运行，比如之前需要一个一个模块的编译，用这个标签可以实现一起编译-->
64. **<modules>**
65. **<module></module>**
66. **</modules>**
67. **</project>**

提示:maven中涉及到一个坐标的概念--》对需要的jar或模块进行定义，或插件进行定位引入

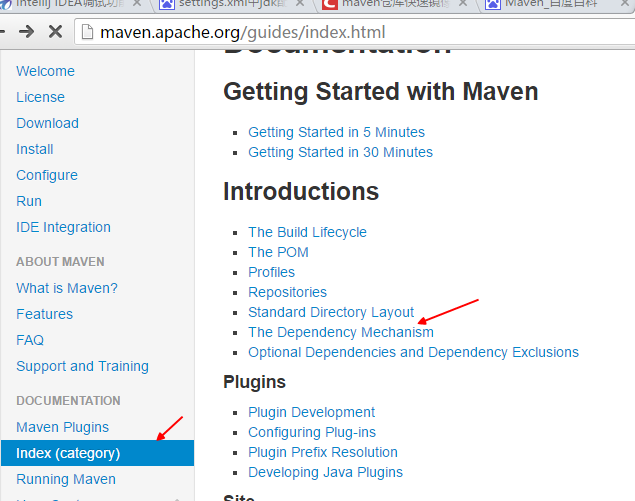
<groupId></groupId>  
<artifactId></artifactId>  
<version></version>

这三个一般是必须的

Maven需要引入那个坐标，写法可以在http://mvnrepository.com/中找到

### 5.1 依赖范围 scope

在maven管网中有详细说明



compile

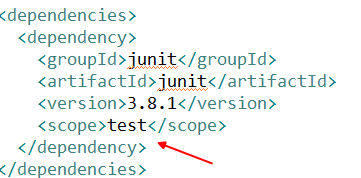
默认就是compile，什么都不配置也就是意味着compile。compile表示被依赖项目需要参与当前项目的编译，当然后续的测试，运行周期也参与其中，是一个比较强的依赖。打包的时候通常需要包含进去。

provided

测试和编译有效，意味着打包的时候可以不用包进去，别的设施(Web Container)会提供。事实上该依赖理论上可以参与编译，测试，运行等周期。相当于compile，但是在打包阶段做了exclude的动作。

runtime

表示被依赖项目无需参与项目的编译，不过后期的测试和运行周期需要其参与。与compile相比，跳过编译而已，说实话在终端的项目（非开源，企业内部系统）中，和compile区别不是很大。比较常见的如JSR×××的实现，对应的API jar是compile的，具体实现是runtime的，compile只需要知道接口就足够了。oracle jdbc驱动架包就是一个很好的例子，一般scope为runntime。另外runntime的依赖通常和optional搭配使用，optional为true。我可以用A实现，也可以用B实现。  
test 表示依赖项目仅仅参与测试相关的工作，包括测试代码的编译，执行。比较典型的如junit。



有效会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。

提示：optionnal

<https://www.cnblogs.com/chenweichu/p/6349478.html>

应用场景：projectA 依赖projectB,  projectB 依赖projectC时

<dependency>

<groupId>com.itear.projectC</groupId>

<artifactId>projectC</artifactId>

<version>0.0.1-SNAPSHOT</version>

<optional>true</optional>

</dependency>

当projectB的<optional>true</optional>时, projectA中如果没有显式的引入projectC, 则projectA不依赖projectC, 即projectA可以自己选择是否依赖projectC

默认<optional>的值为false, 及子项目必须依赖

system

测试和编译有效，可移植性差，有可能会需要依赖本机系统，从参与度来说，也provided相同，不过被依赖项不会从maven仓库抓，而是从本地文件系统拿，一定需**要配合systemPath属性使用**。

import   它只使用在dependencyManagement中标识从其他的pom中导入的dependecy配置

提示：scope=import只能用在dependencyManagement里面,且仅用于type=pom的dependency

参考：

maven中import scope依赖方式解决单继承问题的理解

<https://www.cnblogs.com/huahua035/p/7680607.html>

### 5.2依赖传递

建立3个maven工程 mavena、mavenb、mavenc,我们让mavenb依赖mavena,   mavenc依赖于mavenb

mavana中的pom.xml文件

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
5. **<groupId>**com.test.mavena**</groupId>**
6. **<artifactId>**mavena**</artifactId>**
7. **<version>**0.0.1-SNAPSHOT**</version>**
8. **<packaging>**jar**</packaging>**
10. **<name>**mavena**</name>**
11. **<url>**http://maven.apache.org**</url>**
13. **<properties>**
14. **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**
15. **</properties>**
17. **<dependencies>**
18. **<dependency>**
19. **<groupId>**junit**</groupId>**
20. **<artifactId>**junit**</artifactId>**
21. **<version>**3.8.1**</version>**
22. **<scope>**test**</scope>**
23. **</dependency>**
24. **</dependencies>**
25. **</project>**

mavenb中的pom.xml文件，引入mavena的依赖

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

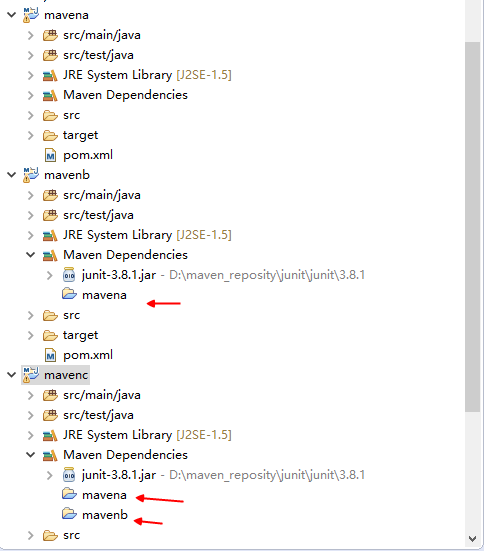
1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
5. **<groupId>**com.test.mavenb**</groupId>**
6. **<artifactId>**mavenb**</artifactId>**
7. **<version>**0.0.1-SNAPSHOT**</version>**
8. **<packaging>**jar**</packaging>**
10. **<name>**mavenb**</name>**
11. **<url>**http://maven.apache.org**</url>**
13. **<properties>**
14. **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**
15. **</properties>**
17. **<dependencies>**
18. **<dependency>**
19. **<groupId>**junit**</groupId>**
20. **<artifactId>**junit**</artifactId>**
21. **<version>**3.8.1**</version>**
22. **<scope>**test**</scope>**
23. **</dependency>**
24. <!-- 引入mavena的依赖
25. 需要将mavena先进行打包，并将其安装到本地仓库中,不然会报mavena-0.0.1-SNAPSHOT.jar找不到
26. 清除上次编译的内容后编译打包
27. run as -maven build.. - goals  clean package
28. 将生成的jar包安装到本地仓库
29. run as -maven build.. - goals  install
30. --**>**
31. **<dependency>**
32. **<groupId>**com.test.mavena**</groupId>**
33. **<artifactId>**mavena**</artifactId>**
34. **<version>**0.0.1-SNAPSHOT**</version>**
35. **</dependency>**
36. **</dependencies>**
37. **</project>**

mavenc中的pom.xml文件引入对mavenb的依赖

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
5. **<groupId>**com.test.mavenc**</groupId>**
6. **<artifactId>**mavenc**</artifactId>**
7. **<version>**0.0.1-SNAPSHOT**</version>**
8. **<packaging>**jar**</packaging>**
10. **<name>**mavenc**</name>**
11. **<url>**http://maven.apache.org**</url>**
13. **<properties>**
14. **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**
15. **</properties>**
17. **<dependencies>**
18. **<dependency>**
19. **<groupId>**junit**</groupId>**
20. **<artifactId>**junit**</artifactId>**
21. **<version>**3.8.1**</version>**
22. **<scope>**test**</scope>**
23. **</dependency>**
24. <!--
25. run as -maven build.. - goals  clean package
26. 将生成的jar包安装到本地仓库
27. run as -maven build.. - goals  install
28. --**>**
29. **<dependency>**
30. **<groupId>**com.test.mavenb**</groupId>**
31. **<artifactId>**mavenb**</artifactId>**
32. **<version>**0.0.1-SNAPSHOT**</version>**
33. **</dependency>**
34. **</dependencies>**
35. **</project>**

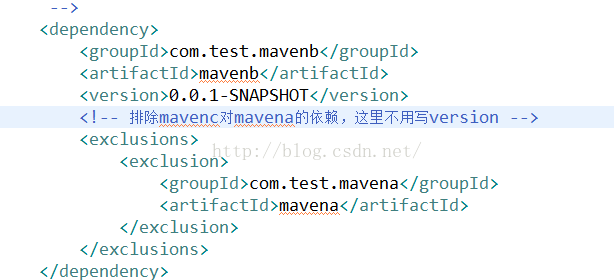
工程中jar情况如下图，mavenb中引入了mavena的jar, mavenc引入了mavenb的依赖后同时又mavena的jar和mavenb的jar



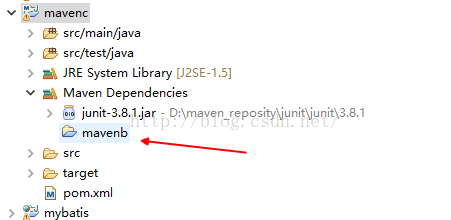
会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。

### 5.3依赖冲突，排除依赖

假如我们只想让mavenc依赖于mavenb,将mavenc中的mavena排除,可以使用如下



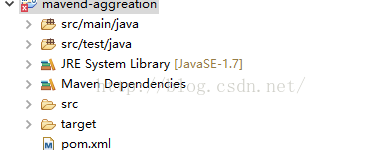
可以看到mavenc中没有了mavena的依赖



### 5.4聚合

如上边的做法，分别建立mavena,mavenb,mavenc如果想要将他们打包且安装到本地仓库或编译时，需要分别进行。如果将所有的工程聚合在一起操作就会很方便

这里我们再建立一个maven-aggeraation工程用于聚合mavena,mavenb,mavenc



**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
5. **<groupId>**com.test.mavend**</groupId>**
6. **<artifactId>**mavend-aggreation**</artifactId>**
7. **<version>**0.0.1-SNAPSHOT**</version>**
9. **<packaging>**pom**</packaging>**
11. **<name>**mavend-aggreation**</name>**
12. **<url>**http://maven.apache.org**</url>**
14. **<properties>**
15. **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**
16. **</properties>**
18. **<dependencies>**
19. **<dependency>**
20. **<groupId>**junit**</groupId>**
21. **<artifactId>**junit**</artifactId>**
22. **<version>**3.8.1**</version>**
23. **<scope>**test**</scope>**
24. **</dependency>**
25. **</dependencies>**
26. <!-- 由于mavend-aggreation工程和其他三个工程是处于一级别
27. 所以需要../回到上级木，不然会找不到其他工程
28. 不过一般我们是建立多模块工程的，有一个父工程，然后又几个子模块
29. --**>**
30. **<modules>**
31. **<module>**../mavena**</module>**
32. **<module>**../mavenb**</module>**
33. **<module>**../mavenc**</module>**
34. **</modules>**
35. **</project>**

注意：

<packaging>pom</packaging>这个父项目不像之前的项目那样创建一个JAR 或者一个WAR，它仅仅是一个引用其它Maven 项目的POM。pom.xml 中下一部分列出了项目的子模块。在多模块maven项目中这些模块在modules元素中定义，每个modules 元素对应了一个simple-parent/目录下的子目录。Maven知道去这些子目录寻找pom.xml 文件，并且，在构建的simp-parent 的时候，它会将这些子模块包含到要构建的项目中。  
 当然，仅仅在父项目，配置子项目是不能够真正实现关联的，因为，这毕竟需要子项目的同意，故！子项目中需要配置：（http://blog.csdn[.Net](http://lib.csdn.net/base/dotnet)/eclipser1987/article/details/5739177）

如下

上边我们不是建立的多模块的项目,举一个多模块的例子：父模块中声明聚合的子模块

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

1. **<modules>**
2. **<module>**newchnl-common**</module>**
3. **<module>**newchnl-service**</module>**
4. **<module>**newchnl-backend**</module>**
5. **<module>**newchnl-front**</module>**
6. **</modules>**

真正引入的时候需要子模块中同意，所以个子模块中需要声明父模块的坐标

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

1. **<parent>**
2. **<groupId>**com.ailk**</groupId>**
3. **<artifactId>**newchnl**</artifactId>**
4. **<version>**0.0.1-SNAPSHOT**</version>**
5. **</parent>**

这样我们编译的时候编译父模块就可以同时编译所有的模块

### 5.5继承

假如新建一个工程作为付工程maven-parent

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
5. **<groupId>**com.test.mavene**</groupId>**
6. **<artifactId>**mavene-parent**</artifactId>**
7. **<version>**0.0.1-SNAPSHOT**</version>**
8. **<packaging>**pom**</packaging>**
10. **<name>**mavene-parent**</name>**
11. **<url>**http://maven.apache.org**</url>**
12. 会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。
13. **<properties>**
14. **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**
15. **<junit.versioin>**3.8.1**</junit.versioin>**
16. **</properties>**
17. <!--  dependencyManagement 中的依赖不会真的引入该模块，
18. 子类中可以使用parent标签定位父模块中的坐标继承该依赖，然后子模块中的
19. dependency标签中引入该（如junit）坐标时，可以不用写version，因为父模块中声明了
20. --**>**
21. **<dependencyManagement>**
22. **<dependencies>**
23. **<dependency>**
24. **<groupId>**junit**</groupId>**
25. **<artifactId>**junit**</artifactId>**
26. **<version>**${junit.versioin}**</version>**
27. **<scope>**test**</scope>**
28. **</dependency>**
29. **</dependencies>**
30. **</dependencyManagement>**
32. **</project>**

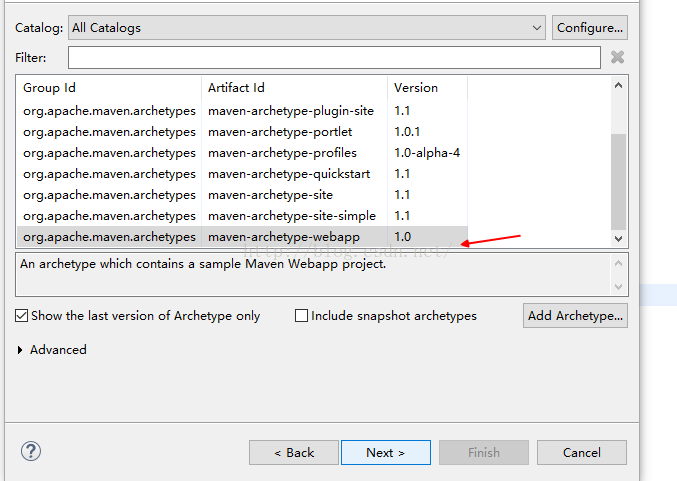
如在mavenc中，继承父模块中的东西

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。](http://blog.csdn.net/u014292162/article/details/52183040" \o "copy)

1. **<project** xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2. xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>**
3. **<modelVersion>**4.0.0**</modelVersion>**
5. **<groupId>**com.test.mavenb**</groupId>**
6. **<artifactId>**mavenc**</artifactId>**
7. **<version>**0.0.1-SNAPSHOT**</version>**
8. **<packaging>**jar**</packaging>**
10. **<name>**mavenc**</name>**
11. **<url>**http://maven.apache.org**</url>**
12. <!--
13. 直接继承父模块中的依赖
14. -->
15. **<parent>**
16. **<groupId>**com.test.mavene**</groupId>**
17. **<artifactId>**mavene-parent**</artifactId>**
18. **<version>**0.0.1-SNAPSHOT**</version>**
19. **</parent>**
20. **<properties>**
21. **<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>**
22. **</properties>**
24. **<dependencies>**
25. **<dependency>**
26. **<groupId>**junit**</groupId>**
27. **<artifactId>**junit**</artifactId>**
28. **<scope>**test**</scope>**
29. **</dependency>**
31. **</dependencies>**
32. **</project>**

## 六、建立web项目

webdemo项目

  
新建项目的时候，会报错需要引入servlet的jar包

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

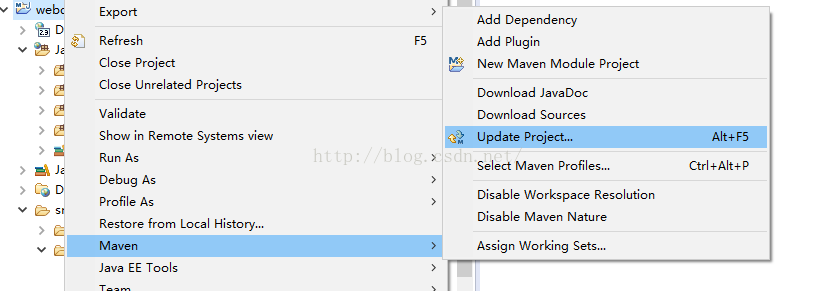
1. **<dependency>**
2. **<groupId>**javax.servlet**</groupId>**
3. **<artifactId>**javax.servlet-api**</artifactId>**
4. **<version>**3.0.1**</version>**
5. **</dependency>**

web项目如果目录结构没有，可以手动创建，如果在Project Explorer视图中无法创建，则可以切换到Navigator视图中创建



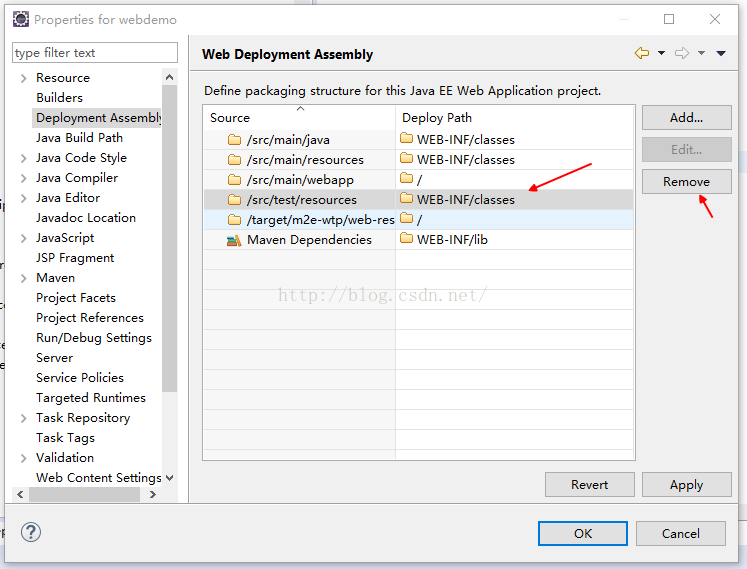
切换到Project Explorer刷新工程

会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。



如果发布代码的时候，测试目录test是不需要的，我们可以将其移除

会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。



### 6.1配置jetty做为web容器

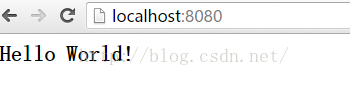
**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

1. **<build>**
2. **<finalName>**webdemo**</finalName>**
3. **<plugins>**
4. **<plugin>**
5. **<groupId>**org.eclipse.jetty**</groupId>**
6. **<artifactId>**jetty-maven-plugin**</artifactId>**
7. **<version>**9.2.10.v20150310**</version>**
8. **<executions>**  <!--配置execultions 表示做完成某件事后需要干嘛-->
9. **<execution>**
10. <!-- 打包完成后使用jetty:run运行 默认端口8080-->
11. **<phase>**package**</phase>**
12. **<goals>**  <!--完成打包后执行run 运行-->
13. **<goal>**run**</goal>**
14. **</goals>**  会分析编译主代码和测试代码需要用到的依赖，一些执行测试和运行时需要的依赖它无法发现。
15. **</execution>**
16. **</executions>**
17. **</plugin>**
18. **</plugins>**
19. **</build>**

上边运行打包命令就可以运行

可以在cmd中使用命令：mvn -Djetty.port=8989  jetty:run 指定端口运行

也可以在eclipse中 Run as --> Maven build..-->goals--> -Djetty.port=8989  jetty:run--->run



### 6.2 配置tomcat插件做web容器

**[html]** [view plain](http://blog.csdn.net/u014292162/article/details/52183040) [copy](http://blog.csdn.net/u014292162/article/details/52183040)

1. **<groupId>**org.apache.tomcat.maven**</groupId>**
2. **<artifactId>**tomcat7-maven-plugin**</artifactId>**
3. **<version>**2.2**</version>**

运行：在eclipse中  
Run as --> Maven build..-->goals--> -Dmaven.tomcat.port=8989

tomcat:run--->run

## 七、坐标查询官网

http://mvnrepository.com/

两个关于镜像的的文章：

http://blog.csdn.net/zuoluoboy/article/details/20007819

<http://my.oschina.net/heweipo/blog/480244>

参考文章：<http://www.cnblogs.com/adolfmc/archive/2012/07/31/2616908.html>

搞懂这7个Maven问题，带你吊打面试官：

<https://mp.weixin.qq.com/s/OZSnGuz2cWiJpy9wu9_ijQ>

Maven精选系列--依赖范围、传递、排除：

<https://mp.weixin.qq.com/s/2bfeo-oiI2_C3KBepPH5Gw>