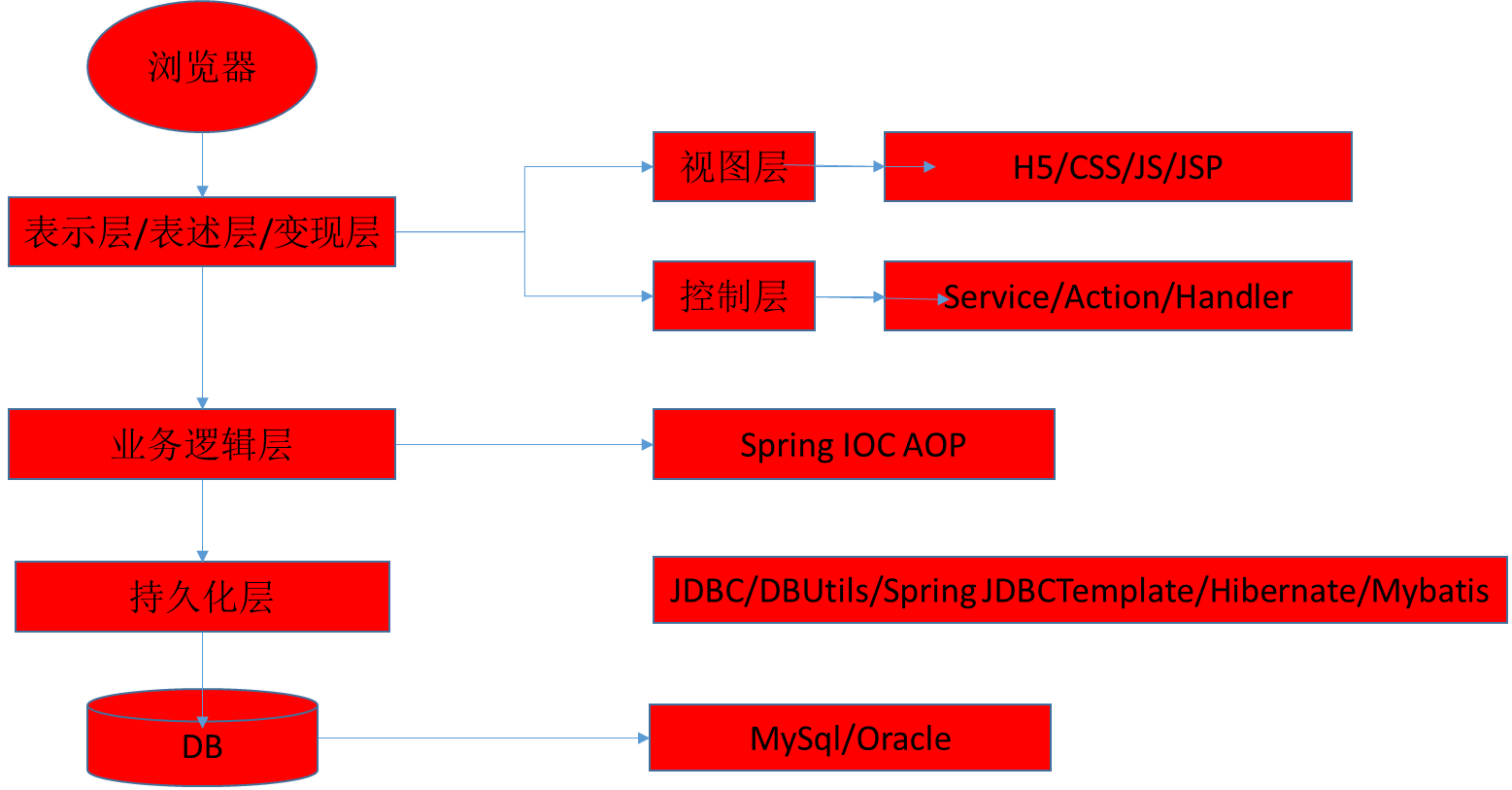
# Maven



## Maven的功能

### 1.一个项目一个工程

如果项目非常庞大,就不适合用package来划分模块,最好一个模块对应一个项目,利于分工合作,而借助Maven就可以将一个项目拆分成多个工程

### 2.手动导包

重复将同样的jar包拷贝到不同的项目工程中,一方面浪费空间,另一方面让工程结构看上去更加复杂,而借助Maven可以将jar包保存到 ”仓库”中,只需在需要的工程中 ”引用”jar包文件即可

### 3.jar包来源

不同技术的官网jar包下载形式复杂,而有些技术的官网已经可以通过Maven或SVN等工具下载文件

如果文件是通过不正规的渠道下载的,那么文件内容可能也不正规,而Maven可以以一种规范的方式下载jar包,因为所有知名框架或第三方工具的jar包已经按照统一的规范存放在了Maven的中央仓库中

### 4.jar包依赖

程序员不需要非常清楚jar包间的依赖关系,这些可以交给Maven负责.它可以自动将jar包所依赖的其他jar包导入

## Maven

### 1.Maven是一款服务于java平台的自动化构建工具

构建工具的发展历程：Make –> Ant -> Maven -> Gradle

### 2.构建过程中的环节

1. 清理：将以前编译得到的.class文件删除
2. 编译：将.java文件编译成.class文件
3. 测试：自动测试，自动调用junit程序
4. 报告：测试程序执行的结果
5. 打包：将动态web工程打成war包，java工程打成jar包
6. 安装：Maven特定的概念 –> 将打包文件复制到仓库的指定位置
7. 部署：将war包复制到servlet容器的指定目录下使其运行

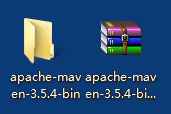
### 3.安装Maven核心程序

#### 1.检查JAVA\_HOME环境变量

C:\Users\Administrator>echo %java\_home%

F:\development\Java\jdk1.8.0\_171

2.解压Maven核心压缩包



3.配置Maven相关的环境变量

1. MAVEN\_HONE 或 M2\_HOME

F:\development\apache-maven-3.5.4-bin\apache-maven-3.5.4

#### 2.path（追加）

F:\development\apache-maven-3.5.4-bin\apache-maven-3.5.4\bin

#### 3.测试，出现以下信息表示成功

C:\Users\Administrator>mvn -v

Apache Maven 3.5.4 (1edded0938998edf8bf061f1ceb3cfdeccf443fe; 2018-06-18T02:33:14+08:00)

Maven home: F:\development\apache-maven-3.5.4-bin\apache-maven-3.5.4\bin\..

Java version: 1.8.0\_171, vendor: Oracle Corporation, runtime: F:\development\Java\jdk1.8.0\_171\jre

Default locale: zh\_CN, platform encoding: GBK

OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"

### 4.Maven核心概念

#### 1.约定目录结构

根目录：工程名

src：源码

main：主程序

java：java源文件

resources：存放框架或其它工具的配置文件

test：测试程序

java

resources

pom.xml：Maven工程的核心配置文件

如果想要自定义目录结果，需要进行配置，如：<param-value>calsspath:spring-context.xml</param-value>

#### 2.POM

Project Object Mode 项目对象模型，pom.xml对于Maven工程师核心配置文件，与构造过程 相关的一切设置都在这个文件中进行配置

3.坐标

使用三个向量在仓库中唯一定位一个Maven工程

1.groupid：公司或组织域名倒叙+项目名

2.artifactid：模块名

3.version：版本

例如打开：F:\development\RepMaven\org\junit\com.springsource.org.junit\4.7.0路径下的com.springsource.org.junit-4.7.0.pom文件会看到：

<groupId>org.junit</groupId>

<artifactId>com.springsource.org.junit</artifactId>

<version>4.7.0</version>

这说明jar包的位置在org/junit/ com.springsource.org.junit/4.7.0目录下，而文件名就是com.springsource.org.junit-4.7.0.jar

#### 4.依赖

<dependency>

<groupId>com.atguigu.maven</groupId>

<artifactId>Hello</artifactId>

<version>0.0.1-SNAPSHOT</version>

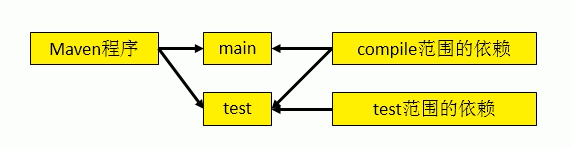
<scope>compile</scope>

</dependency>

1.Maven解析依赖信息时会先到本地仓库查找被依赖的jar包

对于自己开发的Maven工程，需要使用安装命令先将进入依赖工程使用安装命令

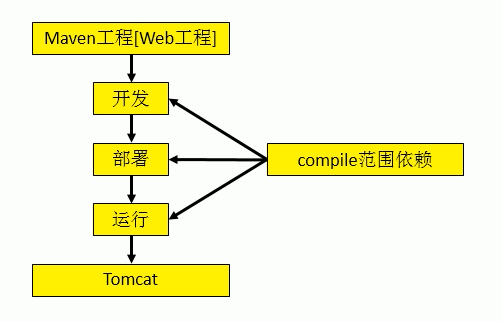
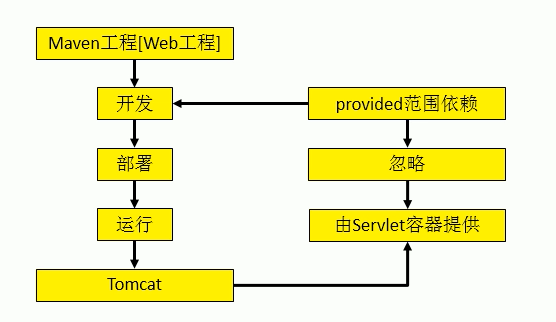
2.依赖的范围：由<scope>标签决定，其常用的有三种取值



1.compile：对主程序有效、对测试程序有效、参与打包，参与部署，默认值

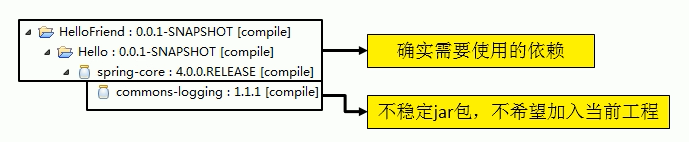
2.test：对主程序无效、对测试程序有效、不参与打包，不参与部署，例子：junit

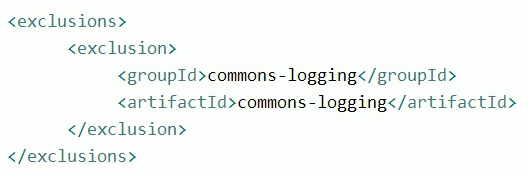
3.privoded：对主程序有效、对测试程序有效、不参与打包、不参与部署，例子：servlet-api.jar

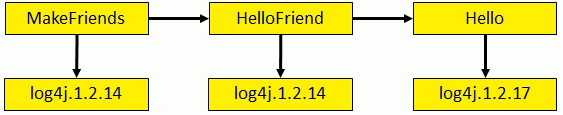
3.依赖是有传递性的，但非compile的依赖不能传递，如果有需要，就得重复声明依赖

4.依赖的排除



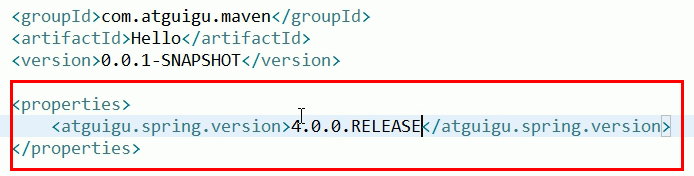


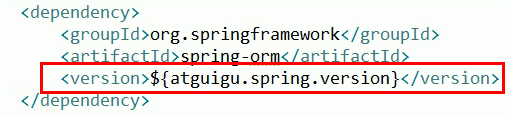
5.依赖重复时的选择原则

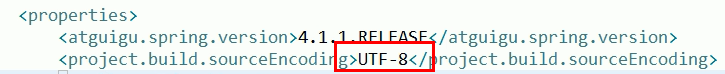
如果出现这种情况，谁离得近谁优先

 这种同级情况，先声明者优先

6. 统一管理依赖的版本





还可以配置编码



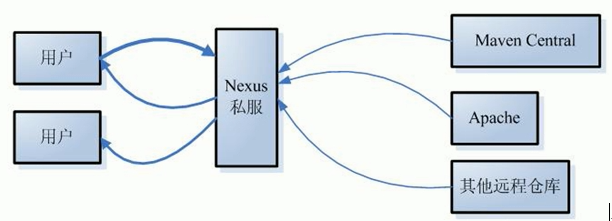
#### 5.仓库

仓库的分类：

1.本地仓库：当前电脑上部署的仓库目录，为当前电脑上所有Maven工程服务

2.远程仓库：

1. 私服：架设在当前局域网环境下，为当前局域网范围内所有的Maven工程服务



2.中央仓库：架设在Internet上，为全世界所有Maven工程服务

3.中央仓库镜像：分担中央仓库的压力，提示访问速度

仓库中保存的内容：Maven工程

1.Maven自身所需的插件

2.第三方框架或工具的jar包

3.自己开发的Maven工程

#### 6.生命周期/插件/目标

1.各个构建环节执行的顺序不能打乱，必须按照既定的正确的顺序执行

2.Maven的核心程序中定义了抽象的生命周期，生命周期中各个阶段的具体任务由插件完成

3.Maven核心程序为了更好地实现自动化构建，按照这一特点执行生命周期的各个阶段：无论现在要执行的是生命周期的那 一阶段，都是从Default生命周期最初的位置开始执行

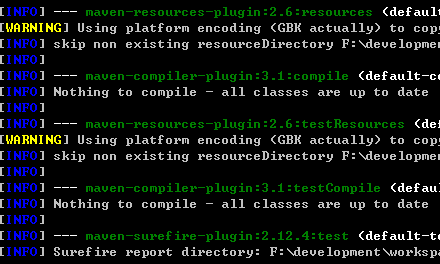
4.插件和目标

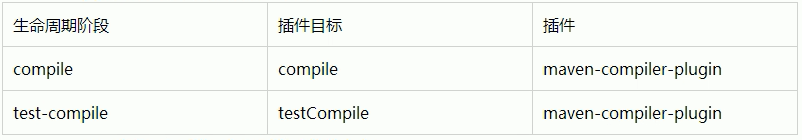
1.生命周期各个阶段仅仅定义了要执行的任务是什么

2.各个阶段和插件的目标是对应的

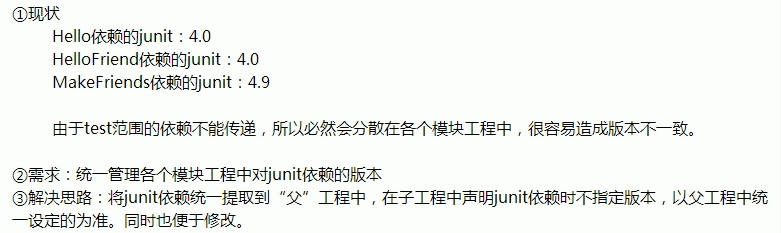
3.相似的目标由特定的插功能的命令件来完成

4.可以将目标看作调用插件



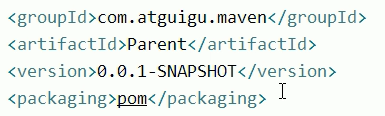


#### 7.继承



④解决步骤：

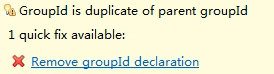
1.创建一个Maven工程作为父工程，打包方式为pom



2.子工程中声明父工程



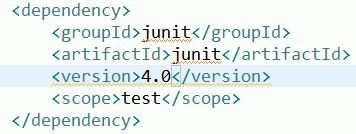
3.将子工程的坐标与父工程坐标重复部分删除

4.在父工程中统一junit的依赖



5.在子工程中删除junit依赖的版本部分内容



#### 8.聚合







### 5.Maven常用命令

1.执行与构建过程相关的Maven命令时必须进入pox.xml所在目录

2.常用命令

1. mavn clean：清理
2. mvn compile：编译主程序
3. mvn test-compile：编译测试程序
4. mvn test：执行测试
5. mvn package：打包
6. mvn install：安装
7. mvn site：生成站点
8. mvn deploy：将最终的包部署到服务器上运行

6.关于联网问题

1.Maven的核心程序仅仅定义了抽象的生命周期，但是具体的工作必须由特定的插件来完成，而插件本身并不包含在Maven的核心程序中

2.当我们执行的Maven命令需要用到某些插件时，Maven核心程序会首先到本地仓库中查找

3.本地仓库的默认位置：系统中当前用户的家目录\m2\repository，即C:\Users\Administrator\.m2

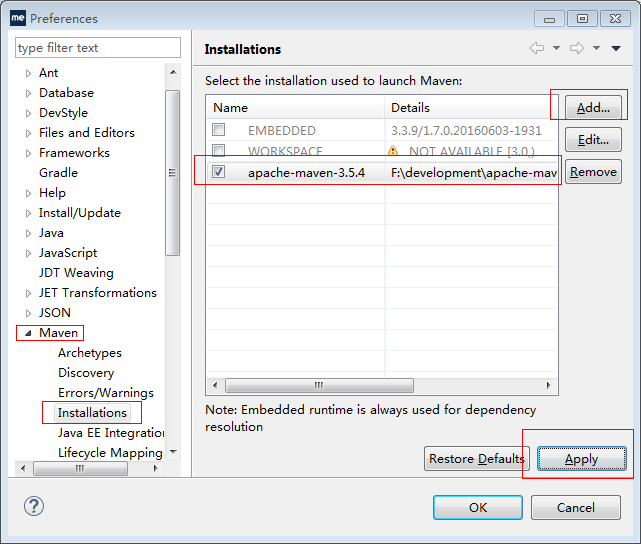
4.Maven核心程序如果在本地仓库找不到需要的插件，那么它会自动连接外网，到中央仓库去下载，如果此时无法连接外网，则构建失败

5.修改本地仓库默认位置

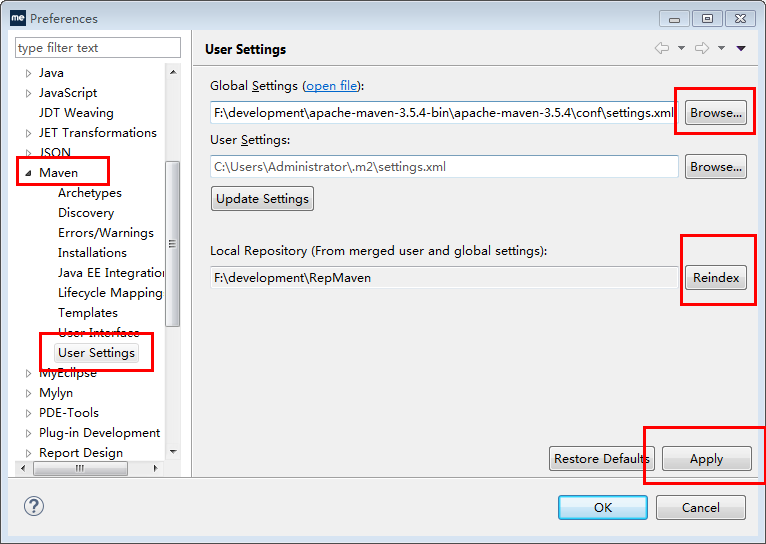
在apache-maven-3.5.4\conf\setting.xml中将<localRepository>标签内容从注释中拷贝出来，然后修改标签中的路径即可

## 在Eclipse中使用Maven插件

### 1.指定Maven核心文件的位置

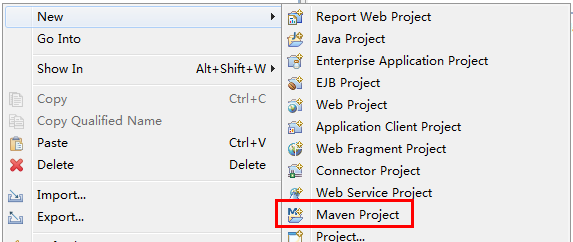
最上面是Eclipse自带的，不建议使用

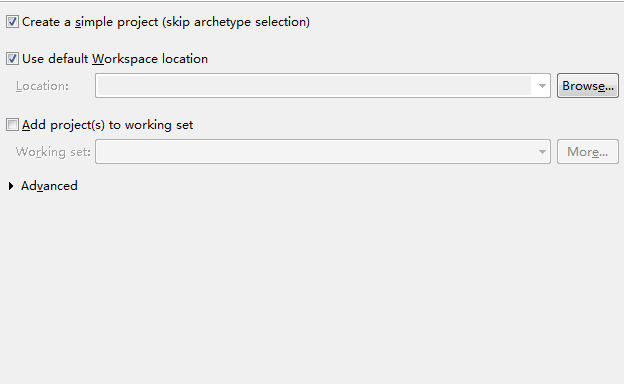
### 2.指定conf/settings.xml的位置，进而获取本地仓库的位置

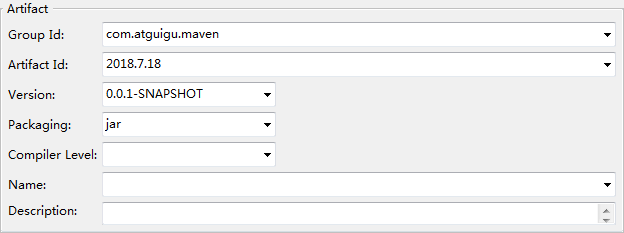


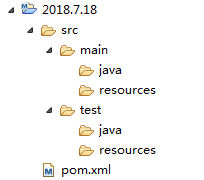
### 3.基本操作

#### 1.创建Maven版的java工程









需要在settings.xml中指定jdk环境，该方法一劳永逸

<profile>

<id>jdk-1.8</id>

<activation>

<activeByDefault>true</activeByDefault>

<jdk>1.8</jdk>

</activation>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

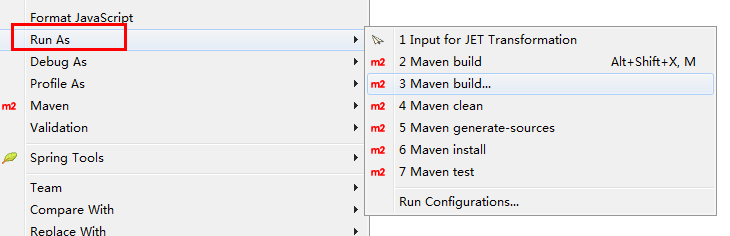
<maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>

</properties>

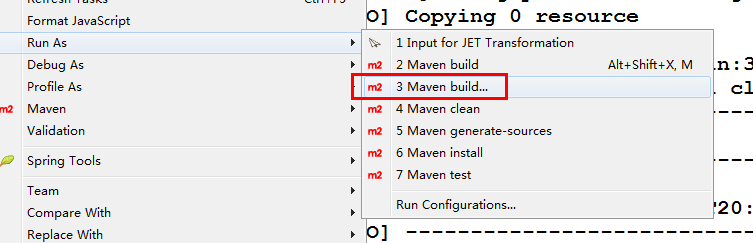
</profile>

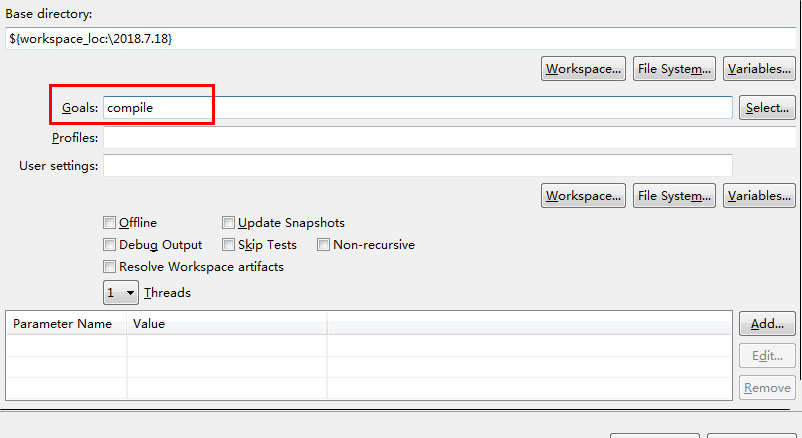
</profiles>

#### 2.执行Maven命令

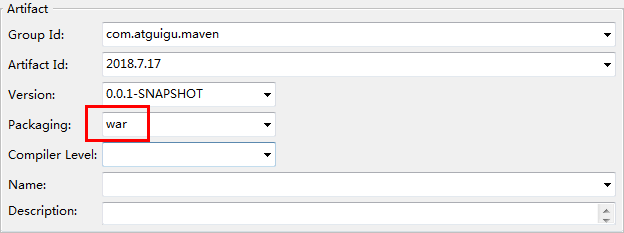


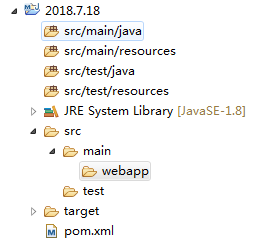
没有的命令可以手动输入

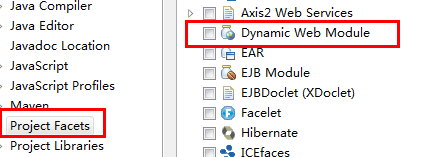


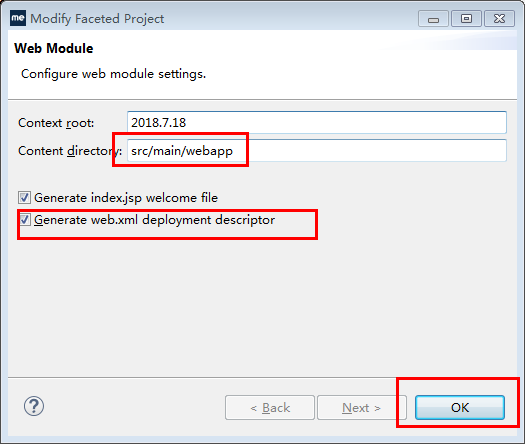


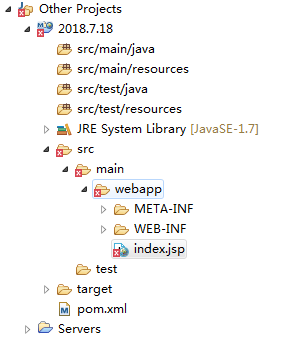
#### 1.创建Maven版的web工程



这里没有WEB-INF(web.xml)、META-INF，需要调整

先去掉勾点Apple，再勾上后点击



添加成功，但页面报错，是因为没有servlet依赖，添加即可

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

</dependencies>

解决jar包无法部署问题

