1. 项目管理工具-Maven



今日学习目标：

（1）了解什么是Maven及Maven的作用

（2）掌握Maven安装及整合到Eclipse中

（3）掌握如何使用Maven构建Java项目

（4）掌握如何使用Maven构建JavaWeb项目

（5）掌握使用Eclipse导入现有的Maven项目

（6）掌握Maven如何管理依赖（即管理jar包）

# Maven介绍

## Maven是什么

### Maven是什么?



**Maven**: 翻译为"专家"、"内行"，是**Apache**下的一个纯Java开发的一个开源项目。

**Maven**是一个项目管理工具，使用Maven可以来管理企业级的Java项目开发及依赖的管理。

使用**Maven**开发，可以简化项目配置，统一项目结构。总之，Maven可以让开发者的工作变得更简单。

### 什么是依赖管理?

什么是依赖管理？要明白依赖管理，首先要知道什么是依赖？

一个Java项目中往往会依赖一些第三方的jar包。比如JDBC程序中要依赖数据库驱动包，或者在使用c3p0连接池时，要依赖c3p0的jar包，再或者解析xml文件时需要导入dom4j的jar包等。 这时我们称这些Java项目**依赖**第三方jar包。

何为依赖管理呢？其实就是对项目中所以依赖的jar包进行规范化管理。

## 为什么要使用Maven?

传统的项目（工程）中管理项目所依赖的jar包完全靠人工进行管理，而人工管理jar包可能会产生诸多问题。

**1、不使用Maven，采用传统方式管理jar包的弊端：**

(1)在一些大型项目中会使用一些框架，比如SSM或者SSH框架，而框架中所包含的jar包非常多（甚至还依赖其他第三方的jar包），如果这些jar包我们手动去网上寻找，有些jar包不容易找到，比较麻烦。

(2)传统方式会将jar包添加到工程中，比如Java工程中将jar包放在工程根目录或者放在自建的lib目录下；JavaWeb工程会将jar包放在:/WEB-INF/lib目录下，这样会导致项目文件的体积暴增（例如，有些项目代码本身体积可能仅仅几兆，而加入jar包后，工程的体积可能会达到几十兆甚至百兆）。

(3)在传统的Java项目中是将所有的jar包统一拷贝的同一目录中，可能会存在jar包文件名称冲突的问题！

(4)在进行项目整合时，可能会出现jar包版本冲突的问题。

(5)在传统java项目中通过编译（手动编译或者在eclipse保存自动编译）、测试（手动在main函数中测试、junit单元测试）、打包部署(手动打war包/手动发布)、运行(手动启动tomcat运行)，最终访问程序。

**2、使用Maven来管理jar包的优势：**

(1)Maven团队维护了一个非常全的Maven仓库(中央仓库)，其中几乎包含了所有的jar包，使用Maven创建的工程可以自动到Maven仓库中下载jar包，方便且不易出错。

另外, 在Maven构建的项目中，如果要使用到一些框架，我们只需要引入框架的核心jar包，框架所依赖的其他第三方jar包，Maven也会一并去下载。

(2)在Maven构建的项目中，不会将项目所依赖的jar包拷贝到每一个项目中，而是将jar包统一放在仓库中管理，在项目中只需要引入jar包的位置(坐标)即可。这样实现了jar包的复用。

(3)Maven采用坐标来管理仓库中的jar包，其中的目录结构为【公司名称+项目/产品名称+版本号】，可以根据坐标定位到具体的jar包。即使使用不同公司中同名的jar包，坐标不同（目录结构不同），文件名也不会冲突。

(4)Maven构建的项目中，通过pom文件对项目中所依赖的jar包及版本进行统一管理，可避免版本冲突。

(5)在Maven项目中，通过一个命令或者一键就可以实现项目的编译（mvn complie）、测试（mvn test）、打包部署（mvn deploy）、运行（mvn install）等。

还有发布到tomcat服务器中运行: mvn tomcat7:run。如果想实现上面的所有过程，只需要记住一个命令：mvn install

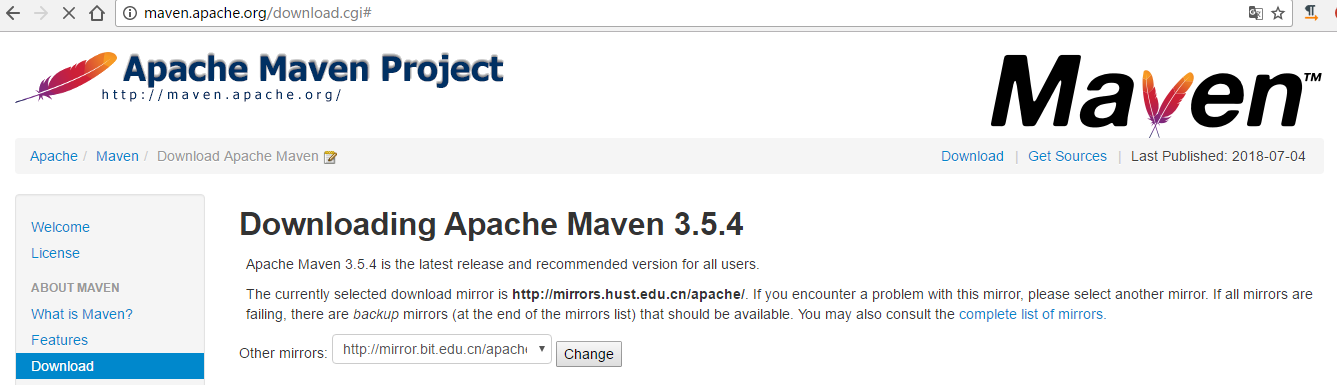
总之，使用Maven遵循规范开发有利于提高大型团队的开发效率，降低项目的维护成本，大公司都会优先使用Maven来构建项目

# Maven安装及配置

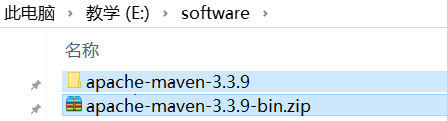
## 下载、安装Maven

### 下载、安装Maven

1、官方下载地址：<http://maven.apache.org/download.cgi>



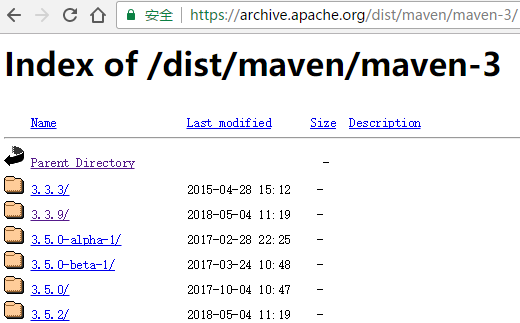
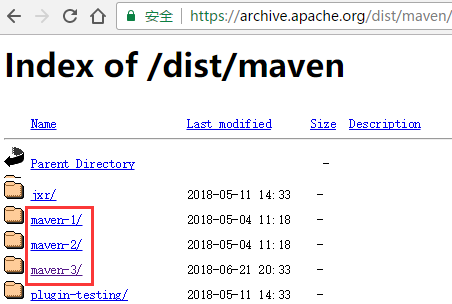
2、下载绿色版，解压之后就可以使用。





原则: 安装的路径中不要有中文和空格!!

3、若要下载旧版本Maven，可以访问：<https://archive.apache.org/dist/maven/maven-3/>



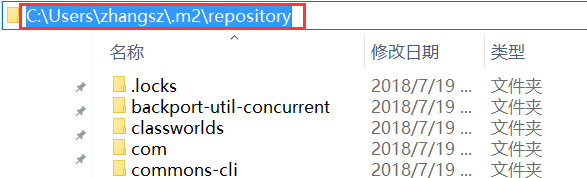
## 配置Maven到Eclipse中

在开发中更多是通过Eclipse+Maven来构建Maven项目，所以这里我们需要将Maven配置到Eclipse开发中。

在将安装好的Maven工具配置的Eclipse开发中之前，需要做一些配置。

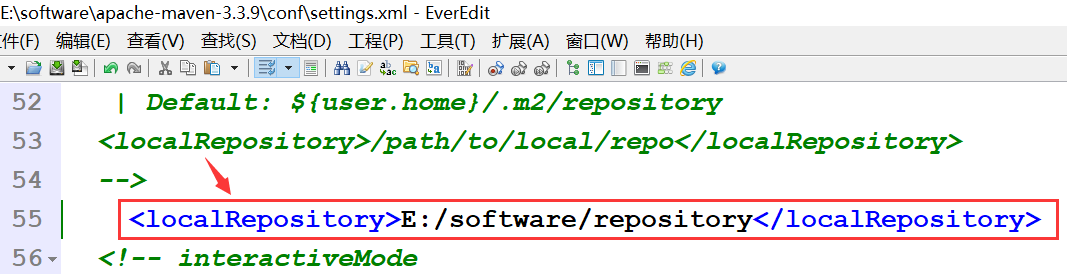
### 修改settings.xml文件中默认仓库的路径

maven指定的本地仓库的默认位置是在c盘，默认在：C:\Users\{当前用户}\.m2\repository，例如:



可以保持默认，当然也可以修改其位置到别的盘符路径。

修改方法：找到[MAVEN\_HOME]/conf/目录中的配置文件settings.xml，修改maven仓库的路径。



配置该目录后，以后通过maven下载的jar包将会保存在配置的目录下。

### 修改settings.xml文件，添加镜像服务器配置

在settings文件中配置私服服务器为达内私服服务器（连接达内内网）或者阿里云服务器（连接外网）。如果不配置默认连接的是中央仓库，由于中央仓库面对的是全球用户，所以在下载jar包时，效率可能会比较低。

继续修改settings.xml文件，如果连接的是达内内网，添加达内镜像服务器配置信息: 在<settings>标签下的<mirrors>标签内部添加如下配置：

配置达内镜像仓库：

<mirror>

<id>nexus-tedu</id>

<name>Nexus tedu</name>

<mirrorOf>central</mirrorOf>

<url>http://maven.tedu.cn/nexus/content/groups/public/</url>

</mirror>

如果连接的外网，无法连接达内的私服服务器，则可以配置连接阿里云私服服务器，配置如下：

配置阿里云镜像仓库：

<mirror>

<id>nexus-tedu</id>

<name>Nexus tedu</name>

<mirrorOf>central</mirrorOf>

<url>http://maven.aliyun.com/nexus/content/groups/public/</url>

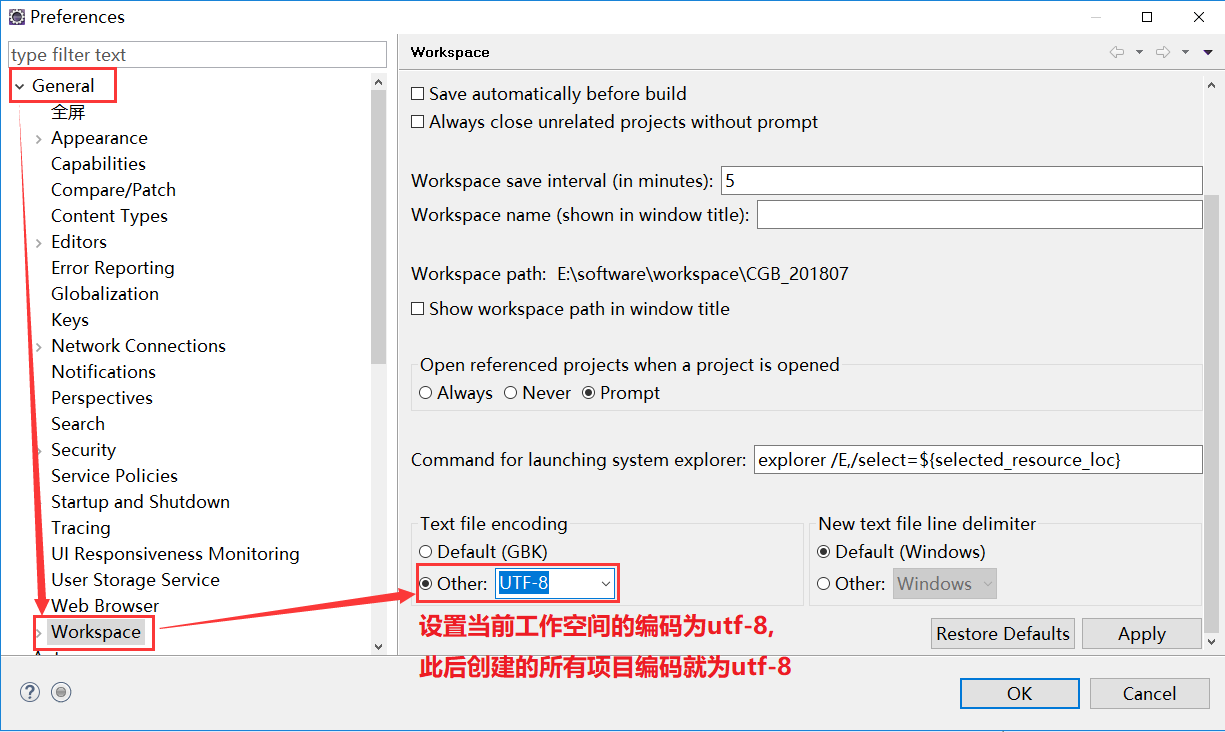
</mirror>

### 准备工作(设置编码及取消验证)

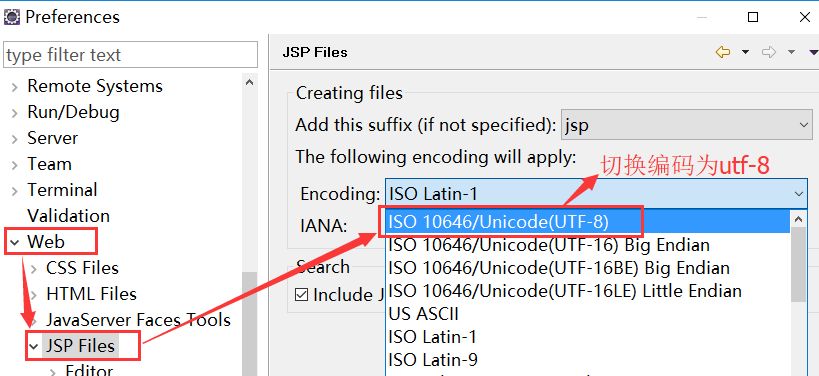
**1、切换新的工作空间，设置工作空间编码为utf-8**

(1)切换到新的工作空间目的: 若旧的工作空间内容过多，可能会导致eclipse不编译，甚至进入休眠状态。

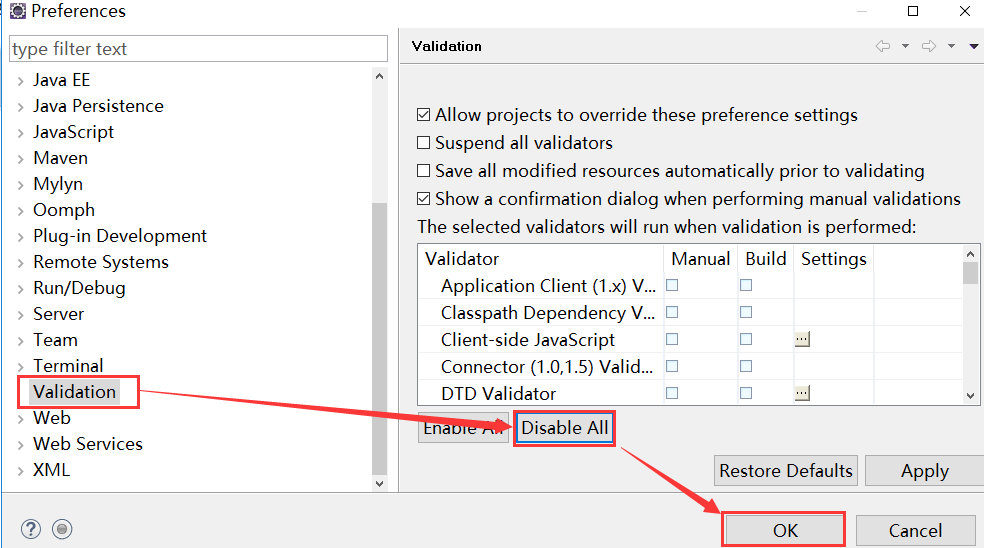
(2)设置工作空间编码为utf-8，此后在当前工作空间下创建的项目编码也默认是utf-8。



**2、设置JSP文件的编码为utf-8**



**3、设置eclipse取消validation验证**

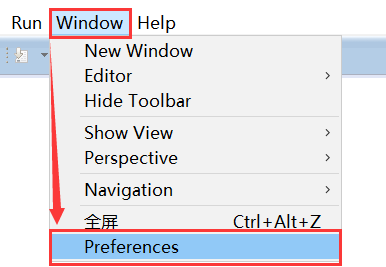


如果不关闭validation验证，在导入项目时，会因为校验项目代码中的语法耗费大量时间！！

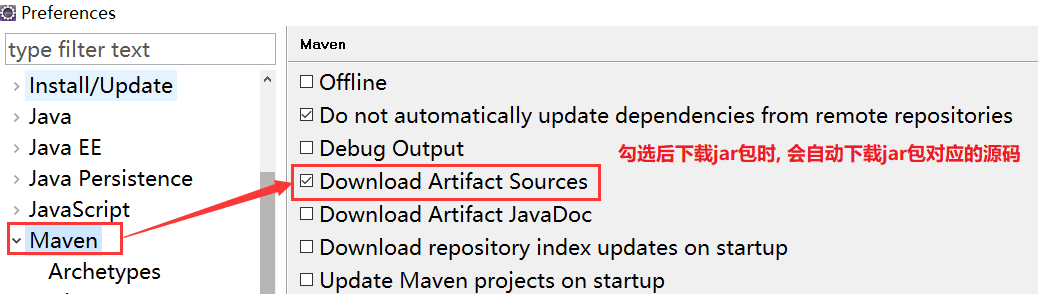
最后，在进行上面的准备工作之后，就可以利用Maven导入或者构建项目了。

### 将Maven配置到Eclipse开发工具中

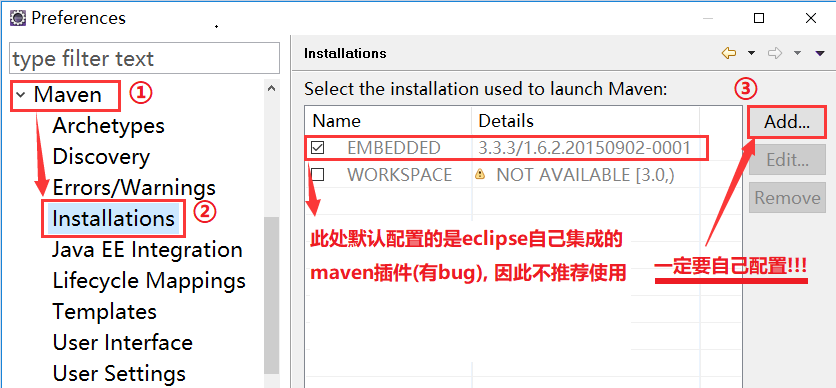
**1、window右键--> Preferences：**



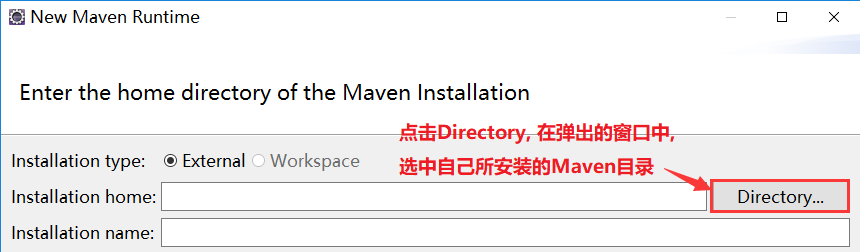
**2、点击Maven选项，在右侧选项中勾选 "Download Artifact Sources"：**

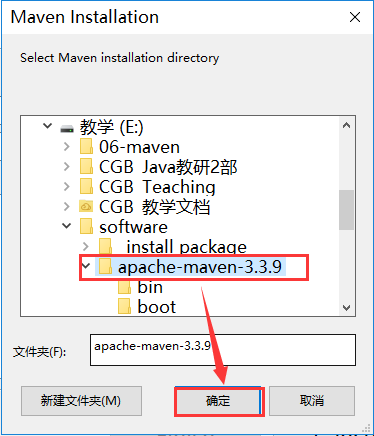


**3、点击add将自己安装的Maven添加进来：**

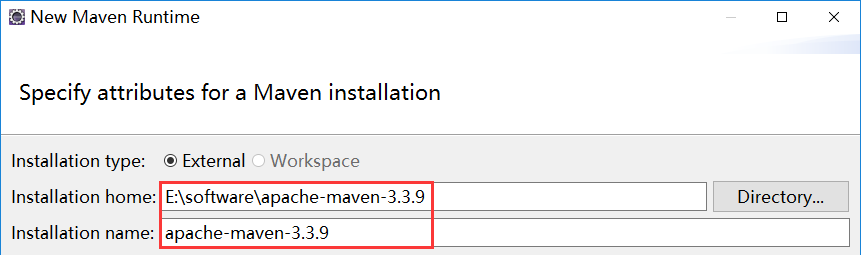


**4、添加自己安装的Maven：**

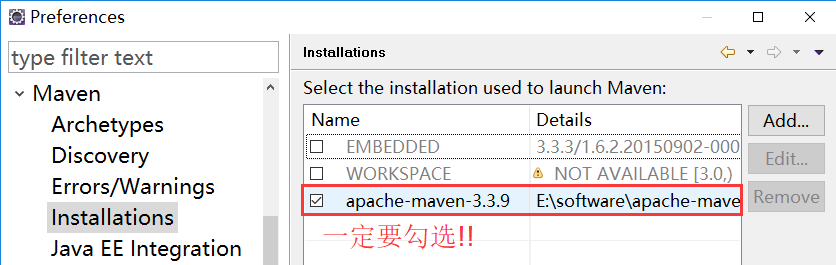




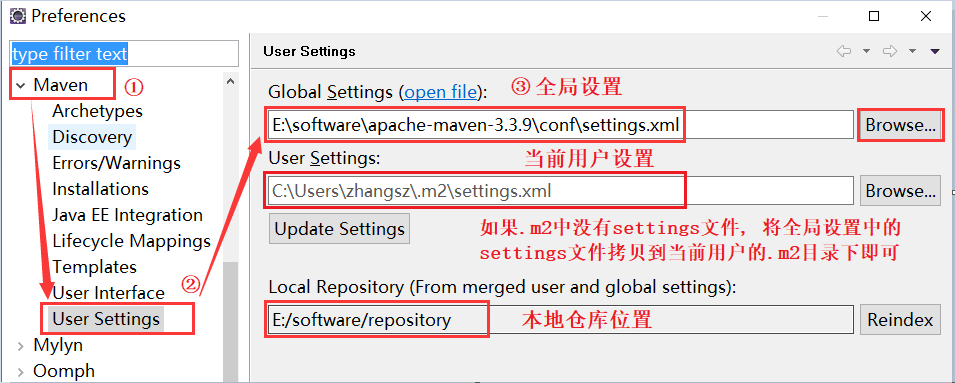
**一定要注意:** 自己安装的Maven不要放在桌面上(容易丢失，并且路径中往往有空格)，maven的安装路径中也不要包含中文和空格!!



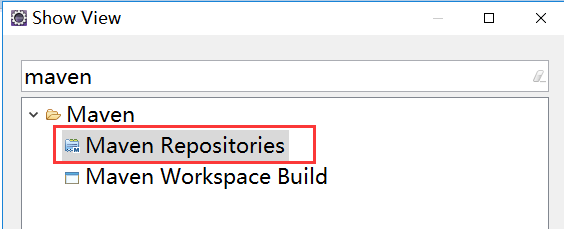
**5、将默认的maven切换为自己配置的maven：**



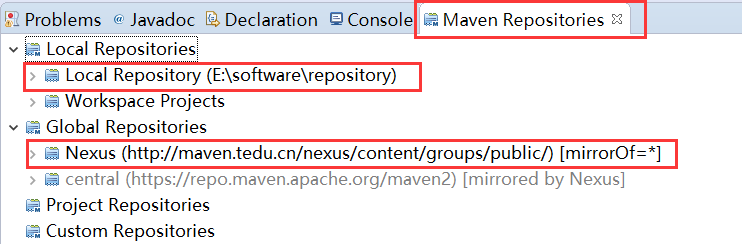
**6、设置maven的settings文件的位置：**



**7、测试是否配置成功：**window---> show view ---> other中搜索 "maven"，点击下面的选框中的选项



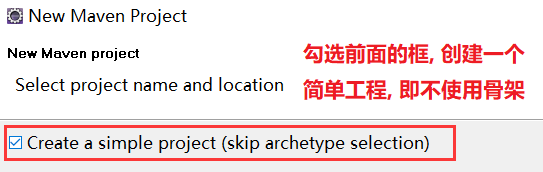
在弹出的窗口中，查看自己配置的本地仓库和远程仓库镜像:



# Maven项目构建

## 使用Maven构建项目1(simple Project)

在利用Maven构建项目时分两种，第一种是：创建**简单工程（Create a simple project）**，即在创建时**勾选前面的框**，不使用骨架。

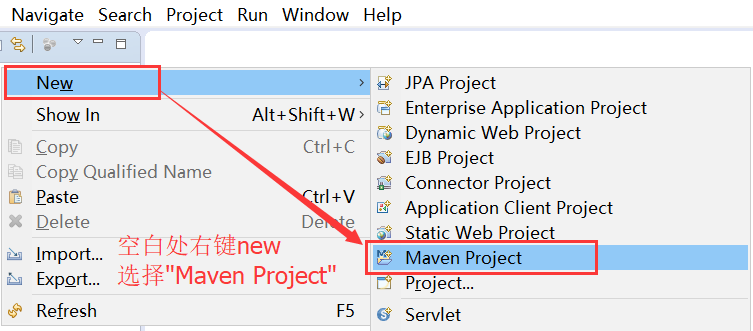


*(不勾选前面的框，即创建使用骨架（其实就是模版）创建Maven工程)*

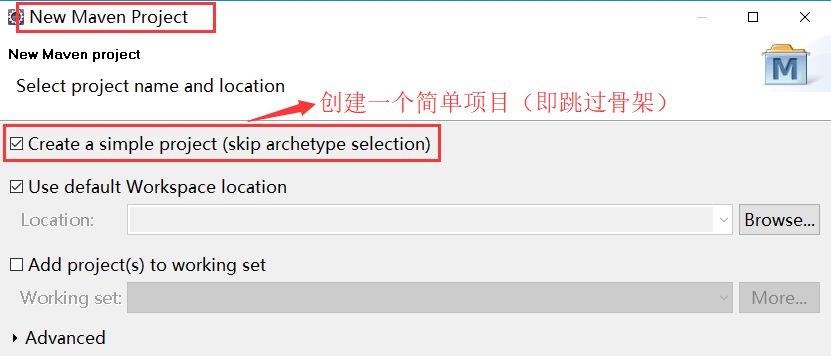
另，在创建简单工程时，还分为创建**Java工程**和**JavaWeb工程**。下面分别进行演示。

### 创建简单工程——Java工程

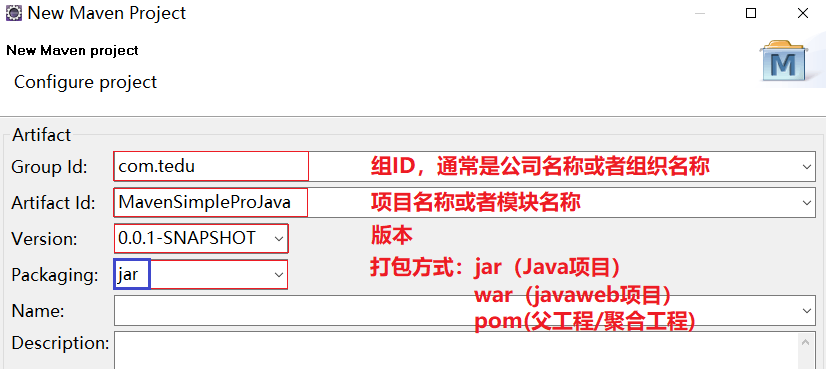
**1、空白处右键New ---> Maven Project：**



**2、在弹出的窗口中，勾选前面的框，创建一个简单工程（即不使用骨架），进入下一步。**



**3、在弹出的窗口中，填写内容(Package选择jar，即创建java工程)，点击完成即可。**



**在上述内容中，必填的内容有四项：**

(1)**Group Id** -- 组的名称，通常填写公司名称（比如 com.tedu）或者组织名称(org.apache..)

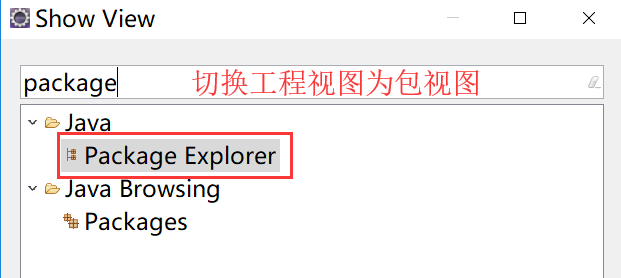
(2)**Artifact Id** -- 项目名称或者模块名称

(3)**Version** -- 项目的版本，创建的项目默认是0.0.1-SNAPSHOT 快照，也叫非正式版，正式版是RELEASE)

(4)**Package** -- 项目的类型: jar表示创建的是Java工程，war表示创建的是web工程，pom表示创建的是父工程（当然相对的还有子工程）或者聚合工程，pom目前我们不讨论。

填写完毕后，点击完成即可完成创建简单Java工程

**4、切换工程视图为包视图：window --> show view，在弹出的窗口中搜索：**

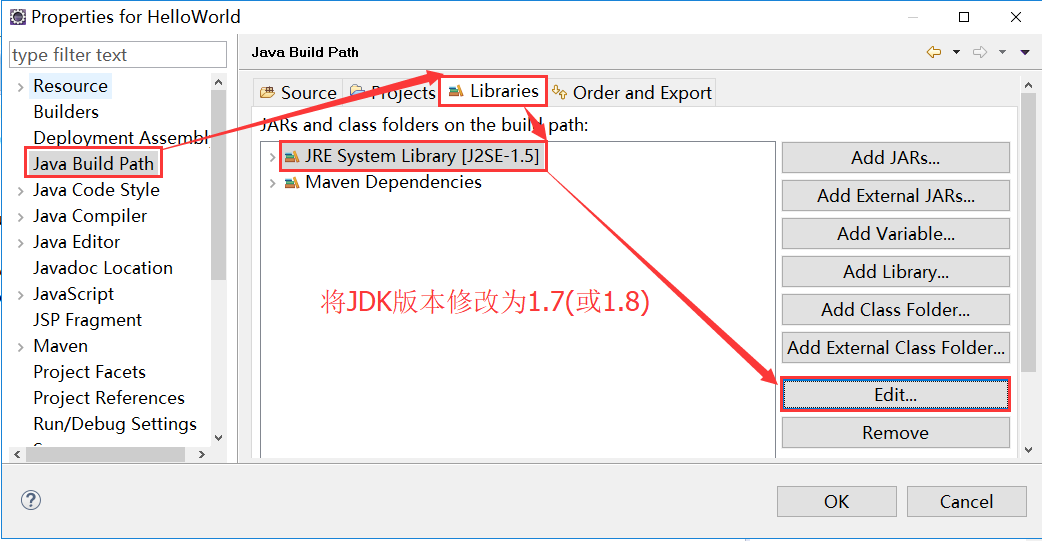


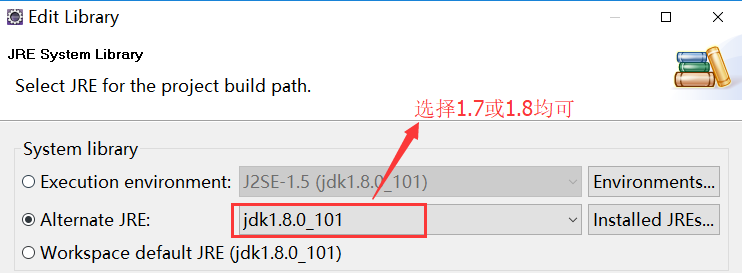
**5、修改项目JDK的版本**

在创建完从Java工程中，默认的JDK版本较低，我们可以将JDK版本（运行和编译）修改为1.7或者1.8版本，修改方式有两种：

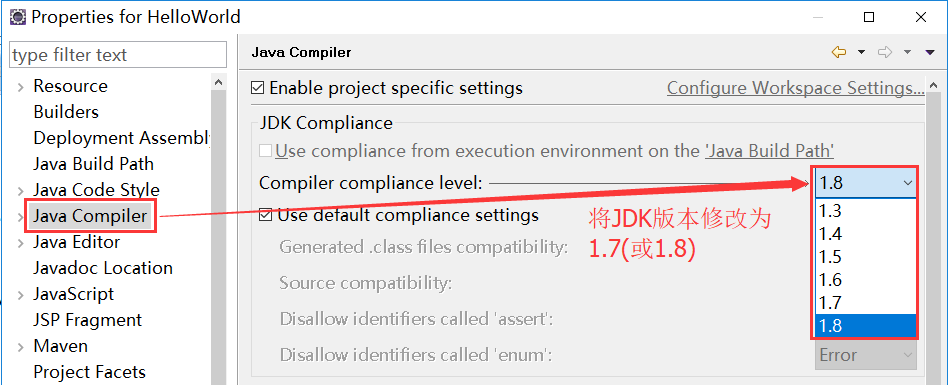
**方式一：**

(1)修改JDK的运行版本：在项目上右键选择 "Build Path" ---> "Configure Build Path…"，再参照下图修改JDK运行版本 (1.7或者1.8)





(2)修改JDK的编译版本：在上面的窗口中切换到 "Java Compiler"，选择JDK版本(1.7或者1.8)



**方式二（推荐）:**

(1) 下面通过配置插件的方式配置JDK的版本。打开当前项目中的pom.xml文件，在根标签内部直接添加如下配置：

配置JDK插件：

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

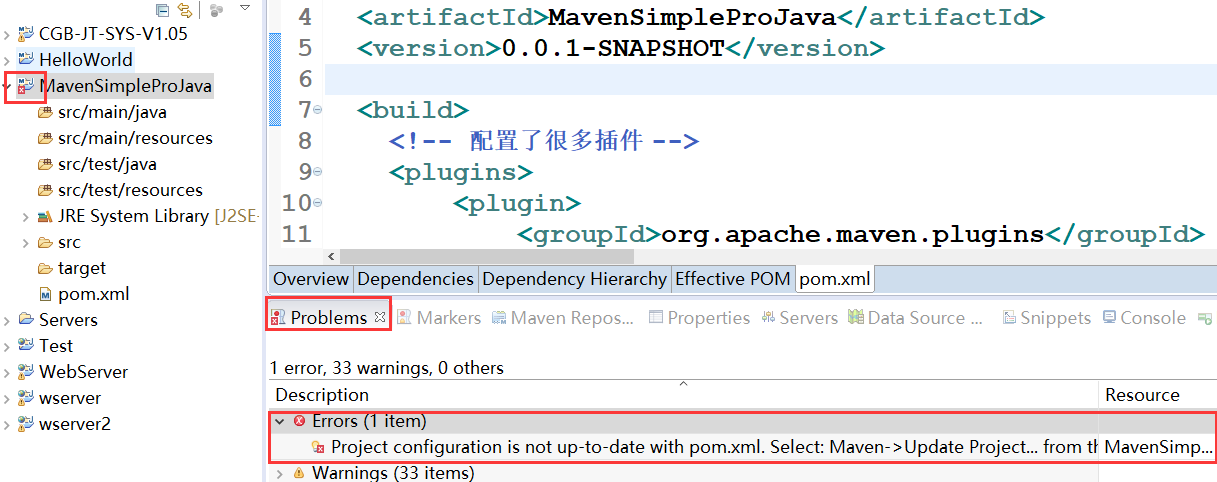
</configuration>

</plugin>

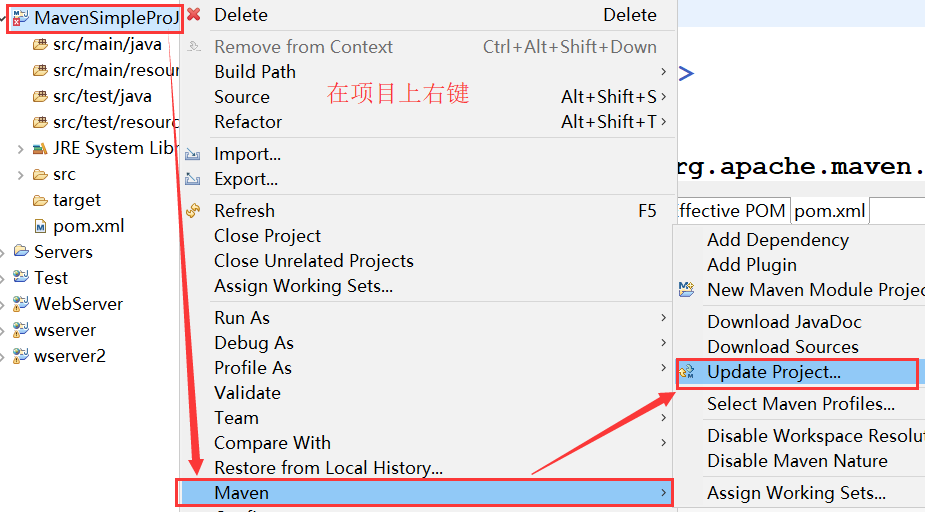
</plugins>

</build>

(2)添加之后会提示如下错误：

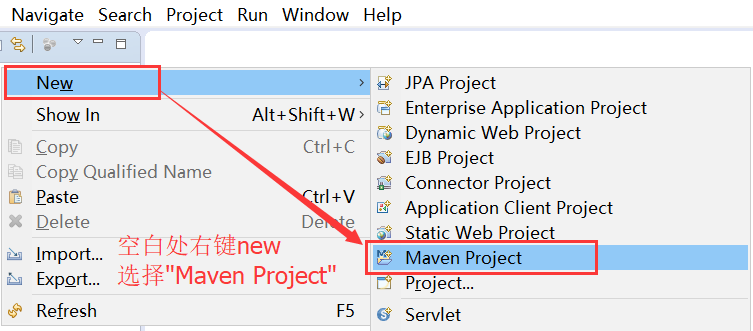


解决办法：在项目上右键选择 **"Maven"** ---> **"Update Project…"**，在弹出的窗口中直接点击OK即可！

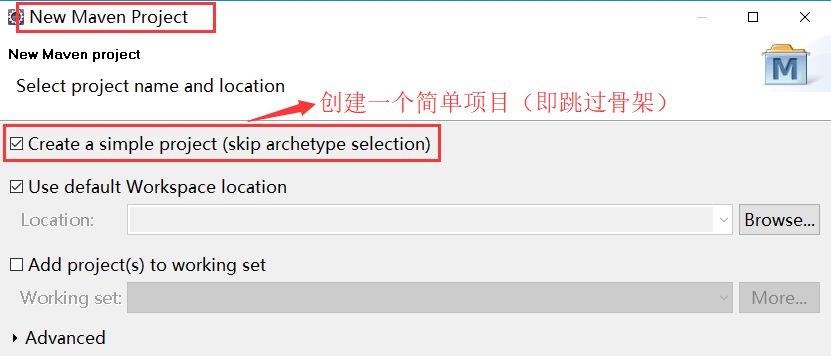


### 创建简单工程——JavaWeb工程

**1、空白处右键New ---> Maven Project：**



**2、在弹出的窗口中，勾选前面的框，创建一个简单工程（即不使用骨架），进入下一步。**



**3、在弹出的窗口中，填写内容(Package选择war，即创建web工程)，点击完成即可。**



**4、修改项目JDK的版本**

下面通过配置插件的方式配置JDK的版本。打开当前项目中的pom.xml文件，在根标签内部直接添加如下配置：

配置JDK插件：

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

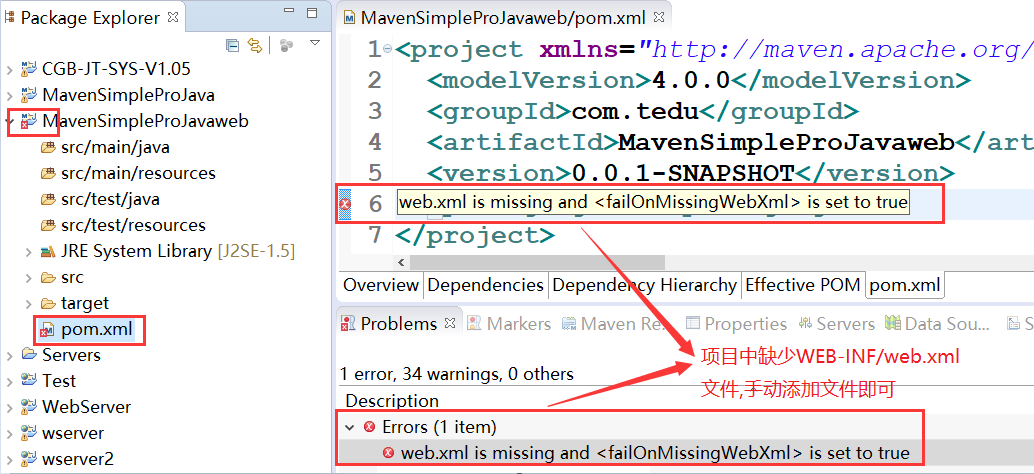
</plugin>

</plugins>

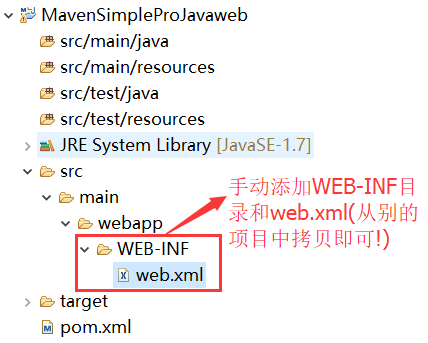
</build>

添加之后若还提示错误，解决办法：在项目上右键选择 **"Maven"** ---> **"Update Project…"**，在弹出的窗口中直接点击OK即可！

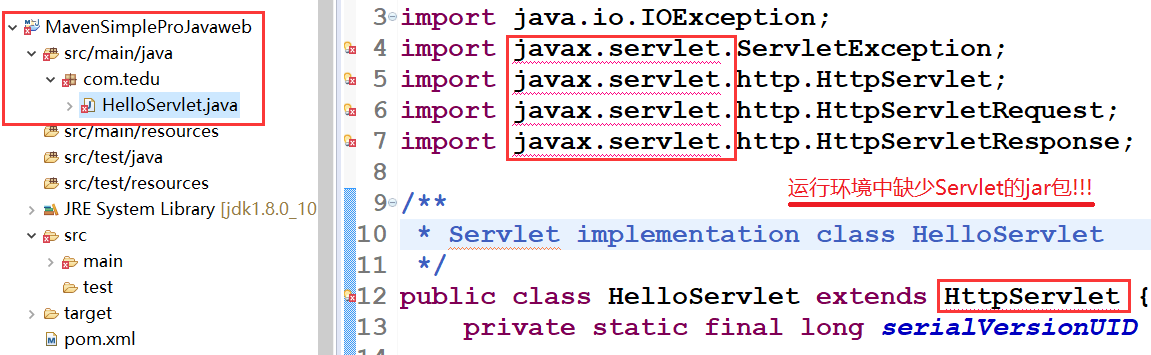
**5、创建完成后pom.xml文件会报错，说找不到web.xml文件，例如：**



**手动添加(拷贝)即可，例如:**

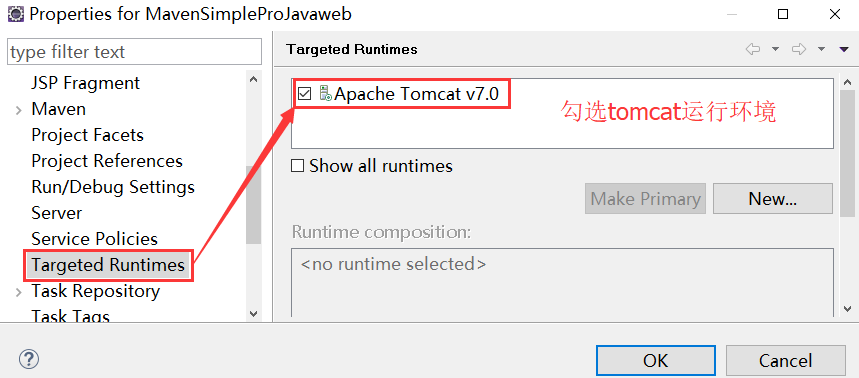


**6、创建Servlet程序，测试运行环境。**



上面的错误是因为运行环境中缺少Servlet的jar包，将tomcat运行环境添加过来即可！

在项目上点击鼠标右键，选择 "Properties" ---> "Targeted Runtimes":



**或者，**如果当前没有配置tomcat运行环境，在项目中的pom.xml文件中的根标签下添加Servlet的jar包的坐标，引入Servlet，如下：

添加依赖：

<dependencies>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

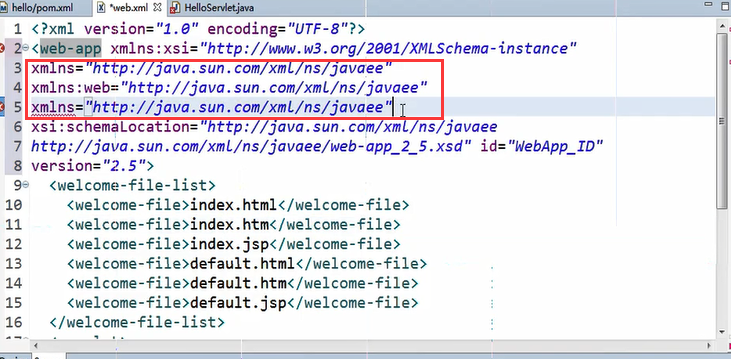
<scope>provided</scope>

</dependency>

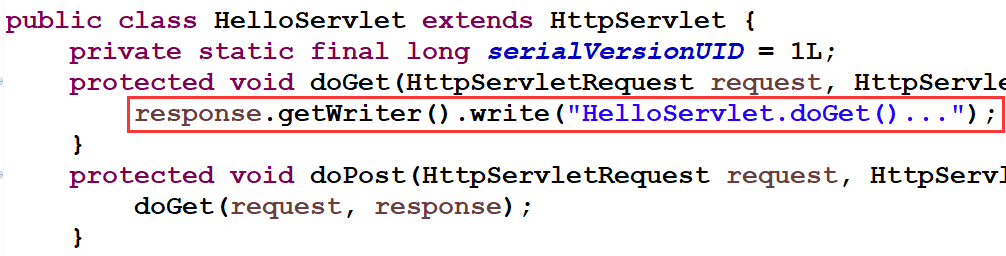
</dependencies>

添加后更新Maven项目：在项目上右键选择 **"Maven"** ---> **"Update Project…"** 更新工程即可！

扩展：*有时在进行上述操作完后，web.xml文件还会报错，可能是因为maven会修改web.xml文件中的头部信息，删除重复的属性即可，如下：*



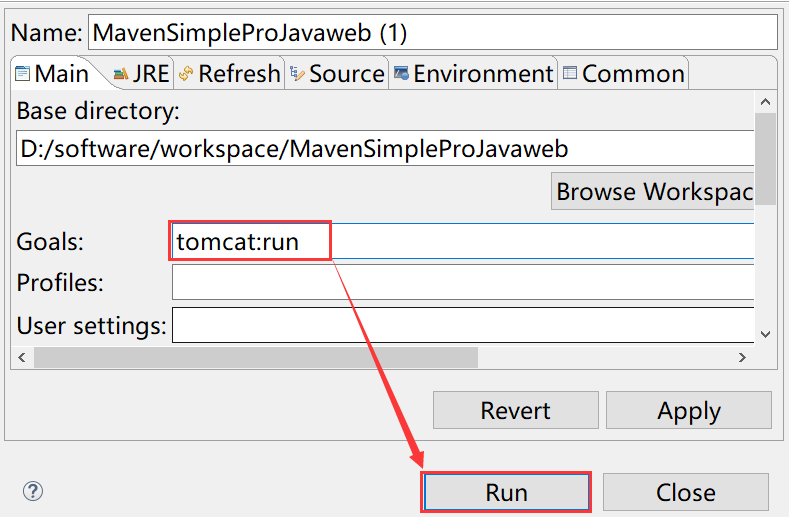
**7、实现Servlet程序**



**8、启动Tomcat运行项目: 在项目上右键选择 "Run As" ---> "Maven build..."**



**在弹出的窗口中配置如下内容（tomcat:run），点击Run按钮启动tomcat服务器。**



**9、测试访问：**打开浏览器访问地址：<http://localhost:8080/MavenSimpleProJavaweb/HelloServlet>

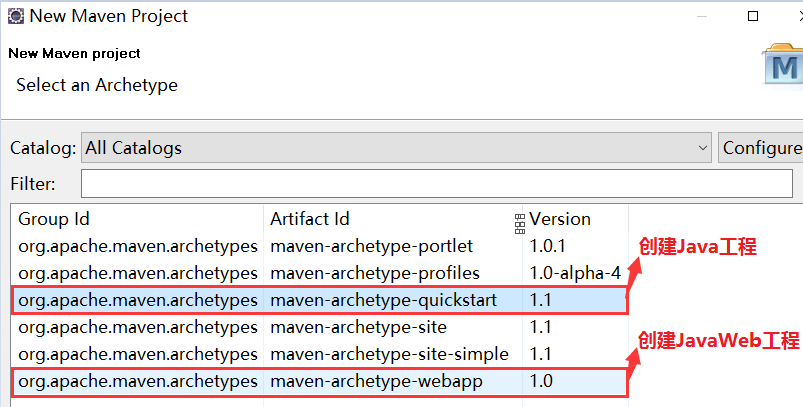


## 使用Maven构建项目2(使用骨架)

在利用Maven构建的项目第二种是：创建**使用骨架的工程**，即在创建时**不勾选前面的框**，使用骨架创建工程。



另，在创建使用骨架的工程时，还分为创建**Java工程**和**JavaWeb工程**。在下面会显示骨架选择，例如：

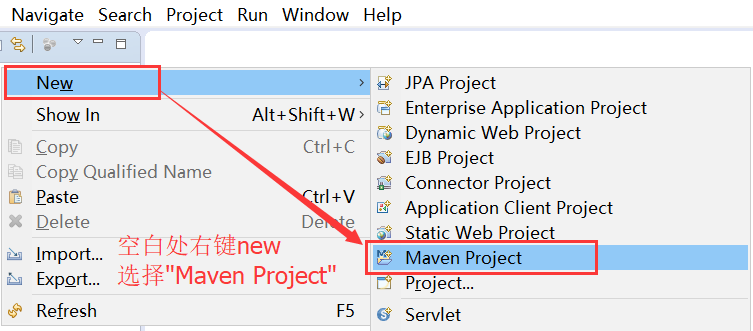


在上面的窗口中，选择 **"maven-archetype-quickstart"** 即创建**Java工程**

若选择 **"maven-archetype-webapp"** 即创建**Javaweb工程**，下面分别进行演示。

### 创建使用骨架的工程——Java工程（quickstart）

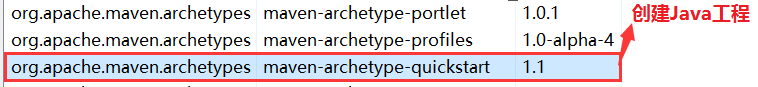
**1、空白处右键New ---> Maven Project：**



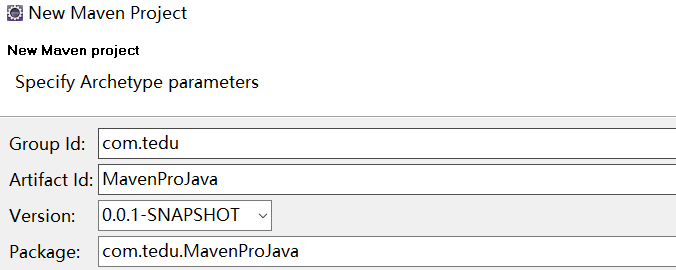
**2、在弹出的窗口中，不勾选前面的框，创建一个使用骨架的工程，进入下一步。**



**3、在骨架选择界面选择 "maven-archetype-quickstart"** 即创建**Java工程**



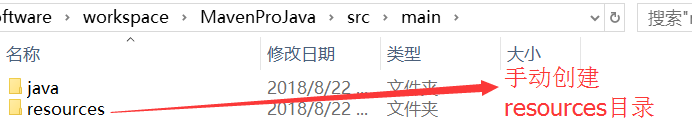
**4、在弹出的窗口中，填写内容点击完成即可创建。**



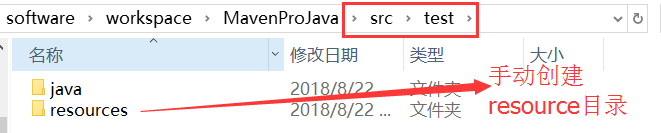
**5、创建完成后，会默认提供测试类，不需要删除即可；同时缺少的目录需手动补全**。



手动补全目录，例如：找到main目录所在的硬盘目录，手动添加resources目录即可！



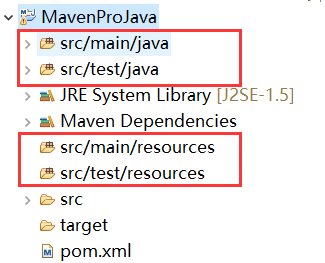
找到test目录所在的硬盘目录，手动添加resources目录即可！



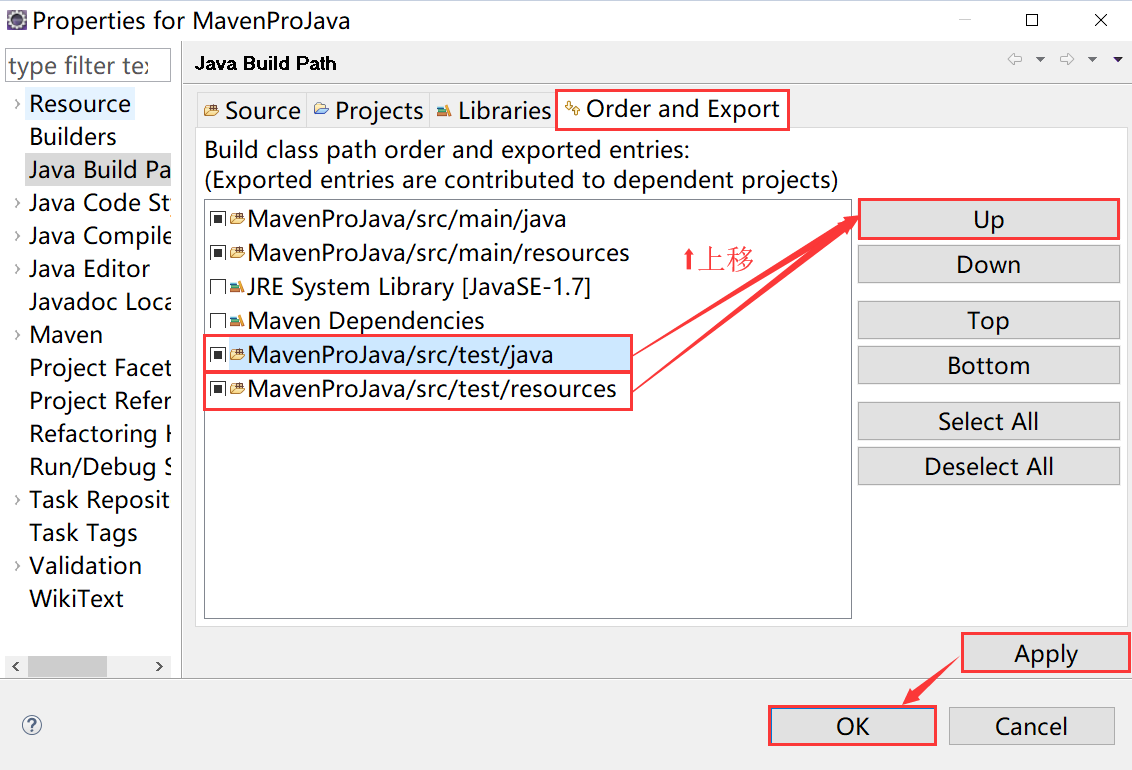
目录补全后，在项目上右键选择 **"Maven"** ---> **"Update Project…"**，即可看到添加的目录。

但是更新Maven工程后，发现项目的顺序不正确，修改方式见下一步。

**6、关于目录的顺序问题：**补全目录后，我们发现目录的顺序不正确



可以通过选中工程右键选择 **"Build Path"** --> **"Configure Build Path"**，在弹出的窗口中，选择 **"Order and Export"** 视图，按照下图调整目录的顺序即可！



**7、修改项目JDK的版本**

下面通过配置插件的方式配置JDK的版本。打开当前项目中的pom.xml文件，在根标签内部直接添加如下配置：

配置JDK插件：

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

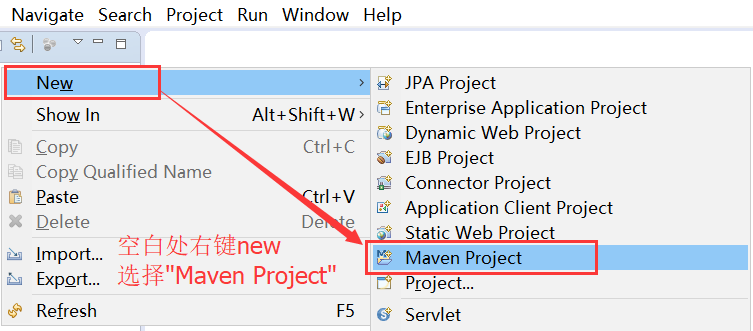
</plugins>

</build>

添加之后若还提示错误，解决办法：在项目上右键选择 **"Maven"** ---> **"Update Project…"**，在弹出的窗口中直接点击OK即可！

### 创建使用骨架的工程——JavaWeb工程（webapp）

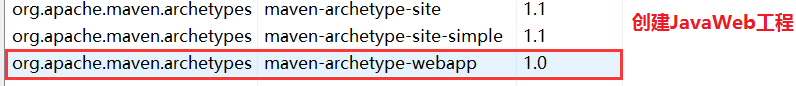
**1、空白处右键New ---> Maven Project：**



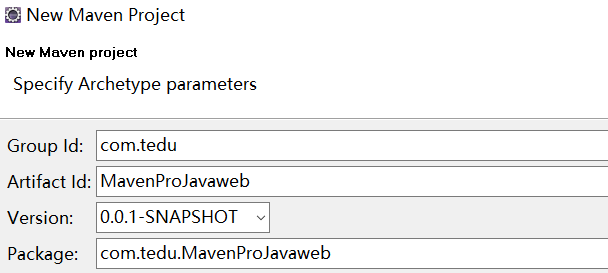
**2、在弹出的窗口中，不勾选前面的框，创建一个使用骨架的工程，进入下一步。**



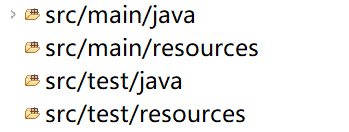
**3、在骨架选择界面选择 "maven-archetype-webapp"** 即创建**Javaweb工程**



**4、在弹出的窗口中，填写内容点击完成即可创建。**



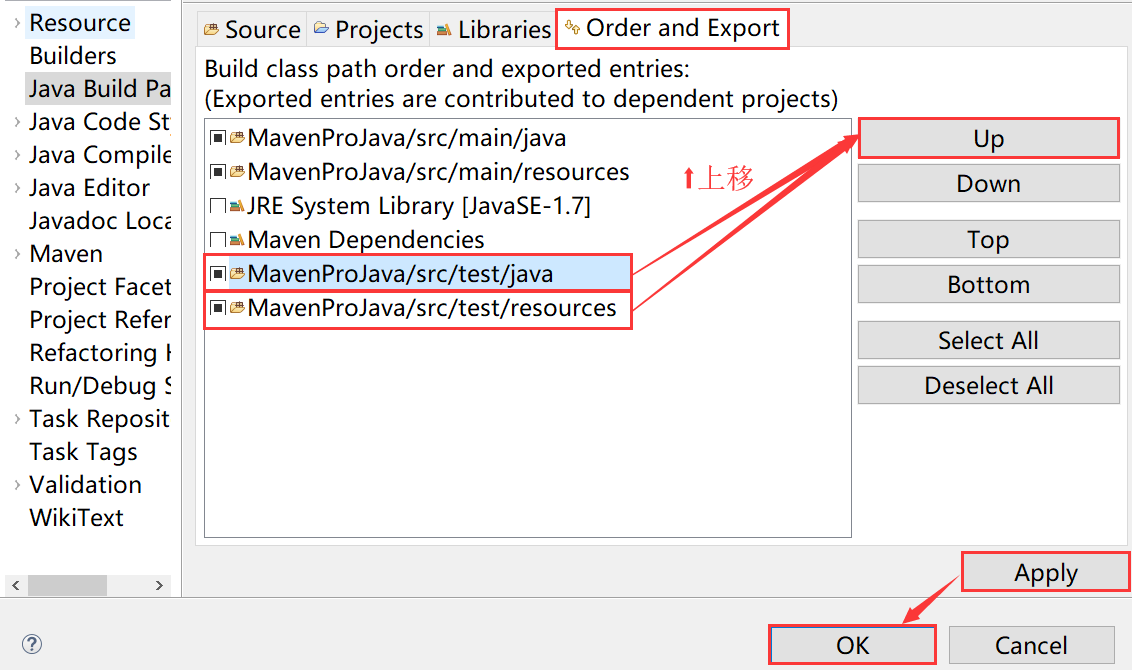
**5、检查工程目录结构，查看是否符合下列结构**：



没有的目录到工作空间位置将没有的目录手动补全即可

目录补全后，在项目上右键选择 **"Maven"** ---> **"Update Project…"**，即可看到添加的目录。

更新Maven工程后，发现项目的顺序不正确，可以通过选中工程右键选择 **"Build Path"** --> **"Configure Build Path"**，在弹出的窗口中，选择 **"Order and Export"** 视图，按照下图调整目录的顺序即可！



**6、修改项目JDK的版本**

下面通过配置插件的方式配置JDK的版本。打开当前项目中的pom.xml文件，在根标签内部直接添加如下配置：

配置JDK插件：

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

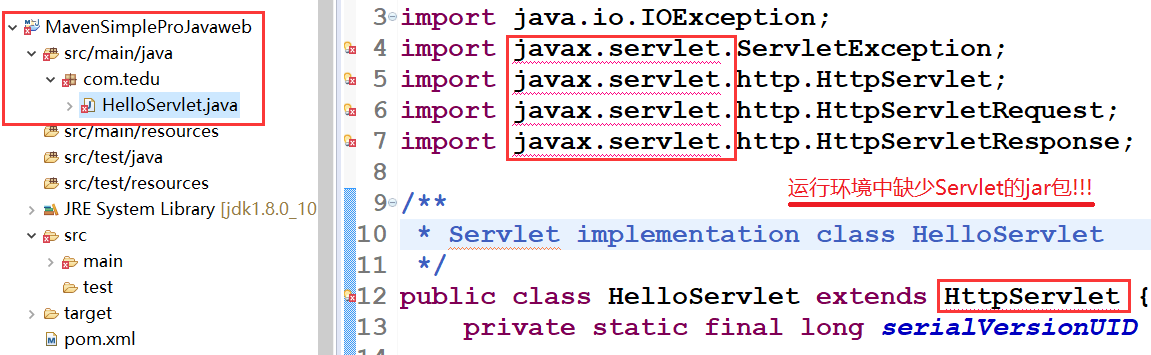
</plugin>

</plugins>

</build>

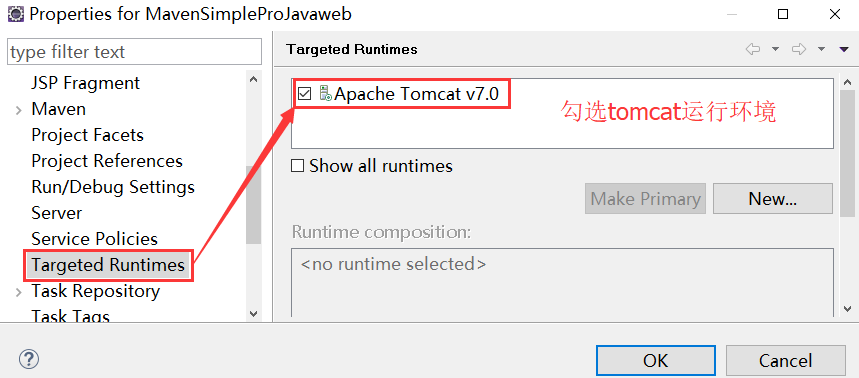
添加之后若还提示错误，解决办法：在项目上右键选择 **"Maven"** ---> **"Update Project…"**，在弹出的窗口中直接点击OK即可！

**7、创建Servlet程序，测试运行环境。**



上面的错误是因为运行环境中缺少Servlet的jar包，将tomcat运行环境添加过来即可！

在项目上点击鼠标右键，选择 "Properties" ---> "Targeted Runtimes":



**或者，**如果当前没有配置tomcat运行环境，在项目中的pom.xml文件中的根标签下添加Servlet的jar包的坐标，引入Servlet，如下：

添加依赖：

<dependencies>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

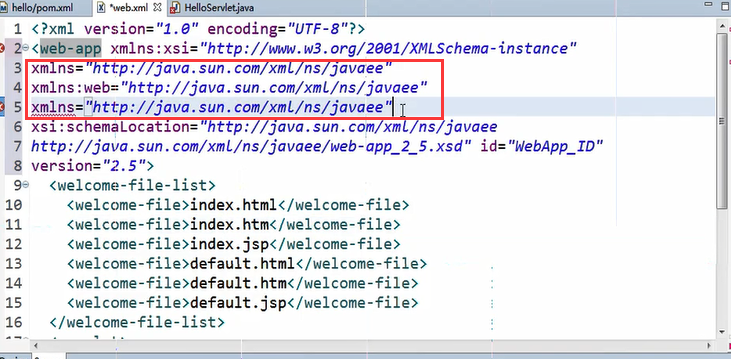
<scope>provided</scope>

</dependency>

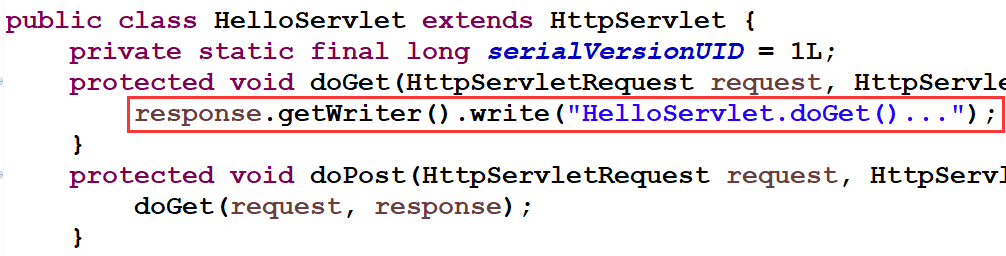
</dependencies>

添加后更新Maven项目：在项目上右键选择 **"Maven"** ---> **"Update Project…"** 更新工程即可！

**扩展：***有时在进行上述操作完后，web.xml文件还会报错，可能是因为maven会修改web.xml文件中的头部信息，删除重复的属性即可，如下*：



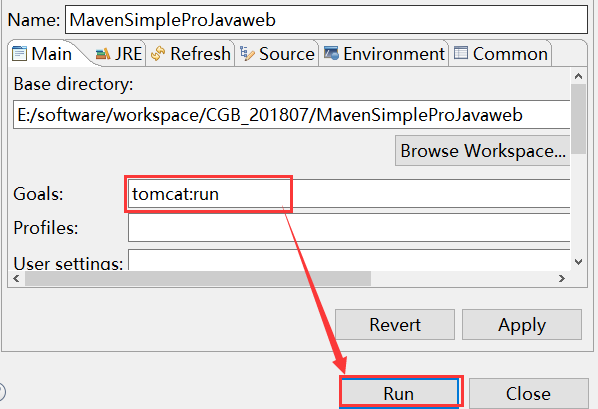
**8、实现Servlet程序**



**9、启动Tomcat: 在项目上右键选择 "Run As" ---> "Maven build..."**



**在弹出的窗口中配置如下内容，点击Run按钮启动tomcat服务器。**



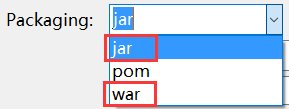
**10、测试访问：**打开浏览器访问地址：<http://localhost:8080/MavenProJavaweb/HelloServlet>



### 两种创建Maven工程方式的比较

**1、方式一：创建简单工程（Create a simple project）**

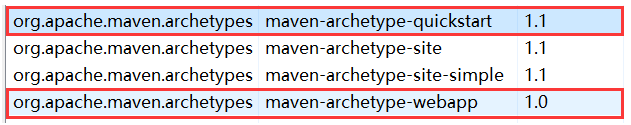
(1)使用方式一创建简单工程，在创建时选择不同的package(jar或者war)，来分别创建Java工程和JavaWeb工程。例如：



(2)使用方式一创建简单工程，优点是项目目录完整（即目录比较全），但是缺点是创建的Web工程默认没有web.xml文件，需要手动添加。

**2、方式二：创建使用骨架的工程**

(1)使用方式二创建使用骨架的工程，在创建时选择不同的骨架（quickstart或webapp），来分别创建Java工程和JavaWeb工程。例如：



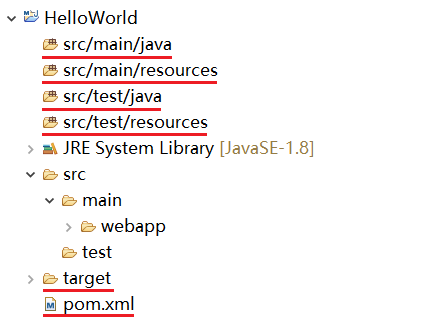
(2)使用方式二创建使用骨架的工程，优点是会提供web.xml文件（内容不能直接使用，还是要修改），缺点是目录不全，需要手工补全，有时还会创建不需要的类。

两种方式都可以使用，根据自己的习惯选择即可！

## Maven项目的目录结构

### Maven项目目录结构

使用Maven创建的工程我们称它为Maven工程，Maven工程具有一定的目录规范，对目录结构有严格的要求，一个Maven工程要具有如下目录结构：



Maven项目目录介绍：

1）/src/main/java -- 主目录下的Java目录，用于存放项目中的.java文件

2）/src/main/resources – 主目下的资源目录，存放项目中的资源文件(如框架的配置文件)

3）/src/test/java -- 测试目录下的Java目录，用于存放所有单元测试类的.java文件，如Junit测试类

4）/src/test/resources – 测试目录下的资源目录，用于存放测试类所需资源文件(如框架的配置文件)

5）/target -- 项目输出目录，编译后的class文件、及项目打成的war包等会输出到此目录中

6）/pom.xml -- maven项目的核心配置文件，文件中通过坐标来管理项目中的所有jar包和插件。

**思考**: Maven的目录中为什么将测试类及相关文件单独存放?

### Maven的pom.xml文件

通过Maven构建的项目中，有一个名称为pom的xml文件，这个文件的Maven项目的核心配置文件。通过该文件可以配置Maven工程所依赖的jar包或者是插件。

例1：通过pom.xml文件配置Maven工程所依赖的jar包。

<dependencies>

<!-- 添加Junit单元测试jar包 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

<scope>test</scope>

<dependency>

<dependencies>

例2：通过pom.xml文件配置Maven工程所依赖的插件。

<build>

<plugins>

<!-- 添加JDK插件(版本1.7) -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

</plugins>

</build>

在pom.xml文件中配置了所依赖的jar包或者插件后，Maven会根据文件中的配置信息到本地仓库中寻找指定的jar包或者插件，如果本地仓库中没有就会到私服或者中央仓库中下载相应的jar包或者插件。再引入到项目中使用！

## 导入已有的Maven项目

### 导入已有的Maven项目

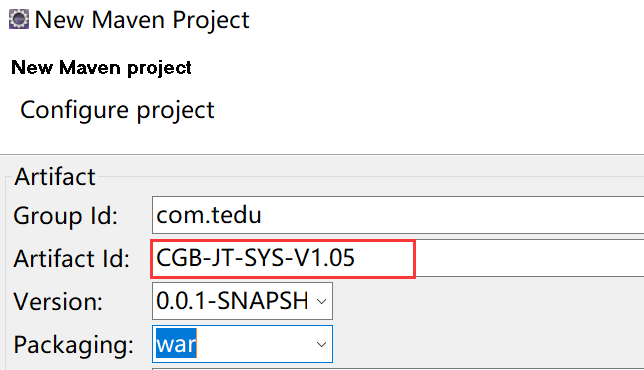
现将后面框架（SSM）阶段完成后实现的权限管理系统（Maven）项目导入到我们的开发工具中。

在导入项目时我们通常会通过 "File" --> "Import..." 来导入项目，但是这样能会产生环境问题：如果项目本身自带的环境和我们当前使用的开发环境不一致，就会产生问题。

下面是导入的步骤：

**1、创建一个新的Maven工程（JavaWeb工程）**

确保已经配置好Maven的环境后，在Eclipse中创建一个新的Maven工程（Javaweb工程），新工程的名字和所导入的工程的名字可以相同也可以不同，例如：



**2、解压CGB-JT-SYS-V1.05目录，复制src目录和pom文件**

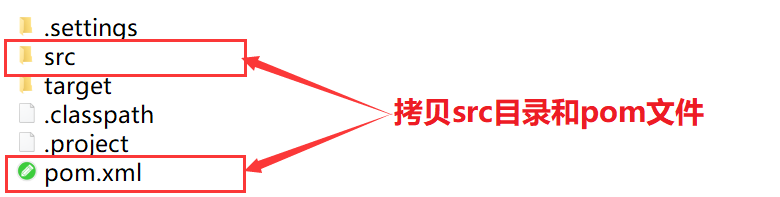
将下发的CGB-JT-SYS-V1.05项目解压出来：



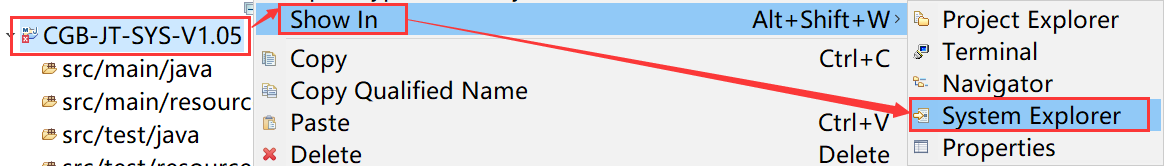
解压后的结构如下：



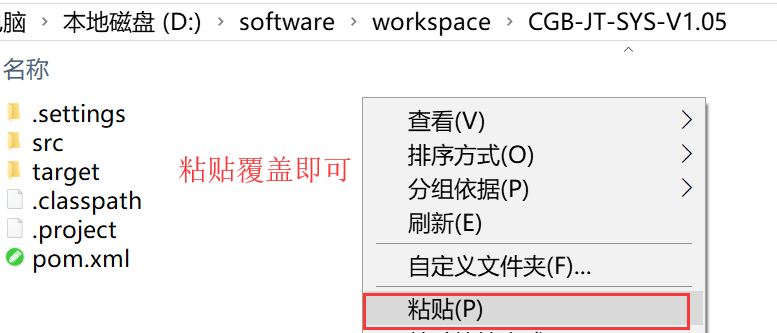
**3、将解压后的目录中的src目录和pom文件选中并复**制。

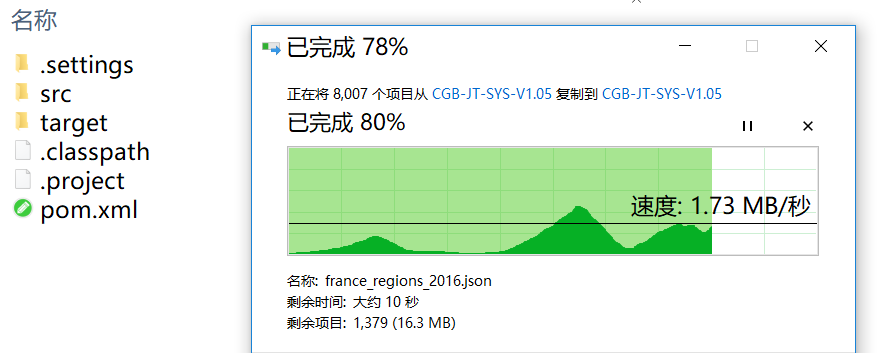


**4、打开新创建的Maven工程所在的硬盘目录**



将复制的src目录和pom文件直接粘贴在项目目录的根目录下即可：





**5、编辑pom.xml文件：配置JDK插件及tomcat插件**

将下面的配置信息直接复制到项目中的pom文件中（注意是复制到根标签下）

<build>

<!-- 添加插件 -->

<plugins>

<!-- 添加JDK插件(版本1.7) -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

<!-- 添加tomcat服务器插件 -->

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<!-- 配置tomcat端口 -->

<port>8080</port>

</configuration>

</plugin>

</plugins>

</build>

**6、更新工程：**在项目上右键选择 **"Maven"** ---> **"Update Project…"**，在弹出的窗口中直接点击OK即可！

**7、添加servlet运行环境**

在进行上述操作后，发现这里报错：



上述错误是因为缺少servlet运行环境导致，在pom文件中添加servlet的jar包对应的坐标，在dependencies标签内添加如下配置：

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

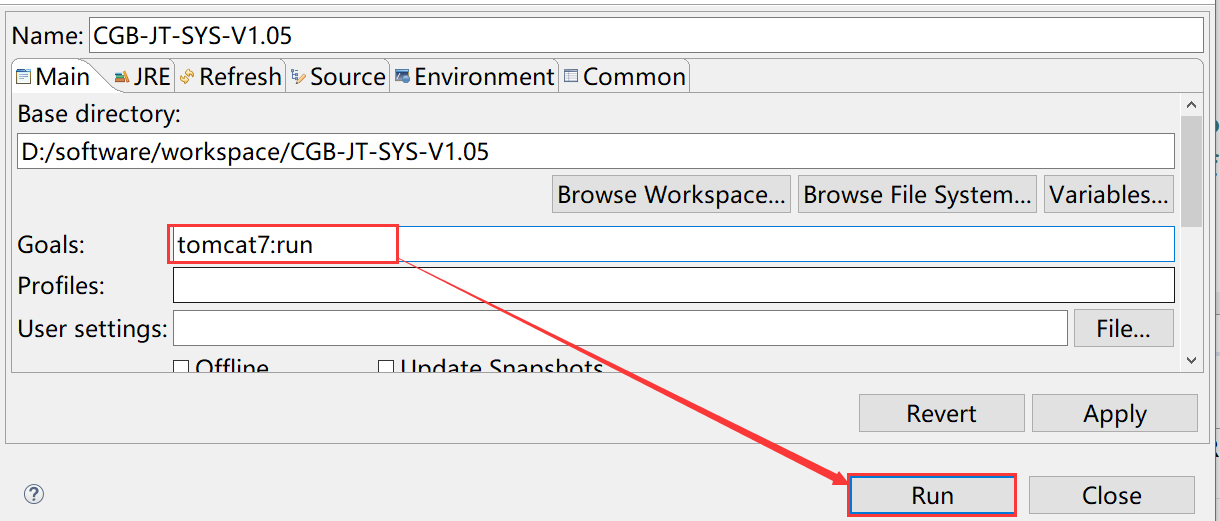
</dependency>

**8、启动服务器运行项目**

(1)启动Tomcat运行项目: 在项目上右键选择 "Run As" ---> "Maven build..."



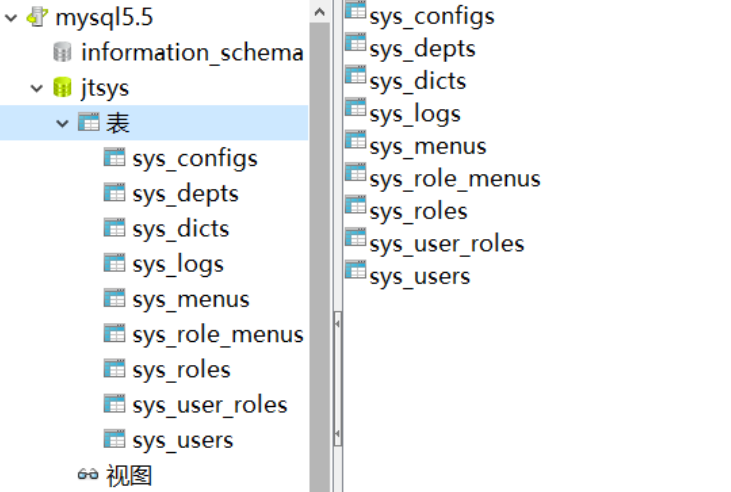
(2)在弹出的窗口中配置如下内容（tomcat:run），点击Run按钮启动tomcat服务器。



运行后查看控制台是否有错误。如果有，则说明环境未准备好，将环境调试好后再运行！

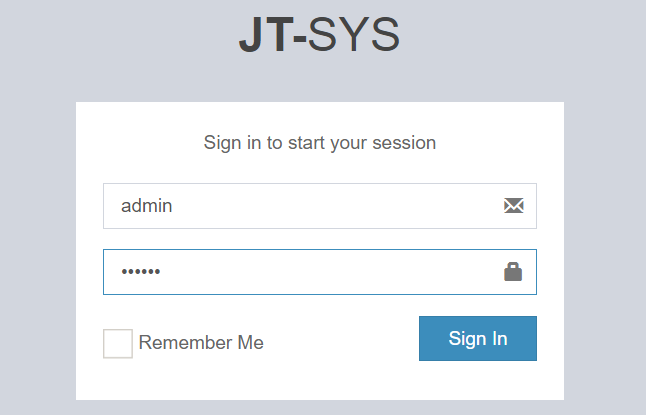
**9、导入数据**

打开navicat工具，连接mysql数据库，执行**jtsys.sql**中的SQL语句，创建数据库、表及插入记录。



**10、访问测试**

在正确完成上面的操作后，打开浏览器访问如下地址：<http://localhost:8080/CGB-JT-SYS-V1.05/loginUI>，可以看到如下界面：



**输入用户和密码，用户名：admin 密码：123456，点击登陆即可！**

**登陆成功看到如下界面，则访问测试成功！**



# 依赖管理

## 依赖管理

### 依赖管理--即jar包管理

**思考：项目中的jar包来自哪里？**

**1、**在Maven创建的项目中，如果需要引用jar包，只需要在项目的pom.xml文件中添加jar包的坐标(GroupID+ArtifactID+Version)即可将jar包引进项目中，之后就可以在项目中使用所引入的jar包了。

例如，现在我们在pom.xml文件中，添加servlet的jar包的坐标如下：

<dependency>

<groupId>**javax.servlet**</groupId>

<artifactId>**servlet-api**</artifactId>

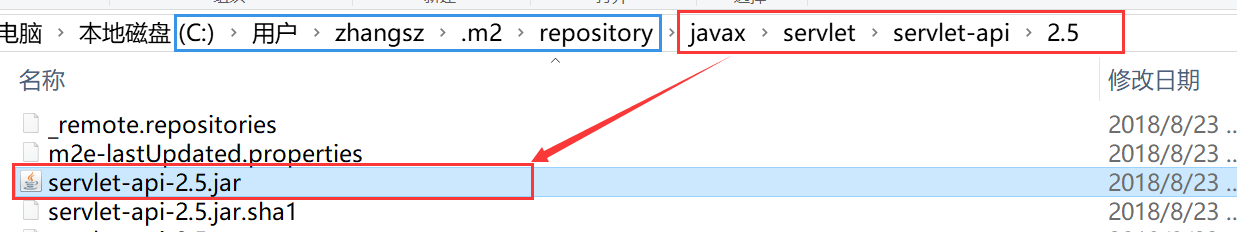
<version>**2.5**</version>

<scope>provided</scope>

</dependency>

**2、**那么问题来了，在pom文件中添加的servlet的jar包坐标，添加的仅仅是一个jar包对应的坐标，而这个**jar包**到底**存放在哪里呢?**

了解Maven管理jar包的规则之后，我们可以找到jar包的存放位置是在**仓库**（仓库默认是在：C:\Users\{当前用户}\.m2\repository）的: /**javax**/**servlet**/**servlet-api**/**2.5**/目录下，例如：



**3、**可以看出，在Maven中，所有的jar包全都存放在本地仓库的目录中，如果项目中需要使用某一个jar包，直接在项目的pom.xml文件中通过坐标(GroupID + ArtifactID + Version)引入指定位置的jar包即可。

**4、**这样可以将项目中所有使用的jar包集中在一个目录（本地仓库）中统一进行管理，而不是在每个项目中都拷贝一份。减少了项目体积，也节省了磁盘空间。

**5、**将来如果别人需要导入你的项目，只需要将项目（当然包括pom.xml文件）代码整体传给对方，无需将jar包发送给对方，对方在配置Maven的环境后，导入你的项目，Maven会自动根据pom.xml文件中配置的坐标，引入对应的jar包。

### 依赖管理--jar包下载

如果引入的jar包在本地仓库中没有呢？

如果是刚配置的Maven环境，本地仓库中还没有太多jar包，此时在pom文件中通过坐标引入jar包，而本地仓库中没有这个jar包，这时会怎么样呢？

**1、**例如，在Maven项目的pom文件中添加spring框架的坐标:

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.1.7.RELEASE</version>

**2、**如果在pom.xml文件中添加了新的坐标，将会到本地仓库中寻找对应的jar包（如果本地仓库中有，则直接引用），如果本地仓库中没有将会从远程仓库中下载对应的jar包到本地仓库中再使用

**3、**另，Maven在下载spring核心jar包的同时，也会将核心jar包所依赖的jar包也下载下来。

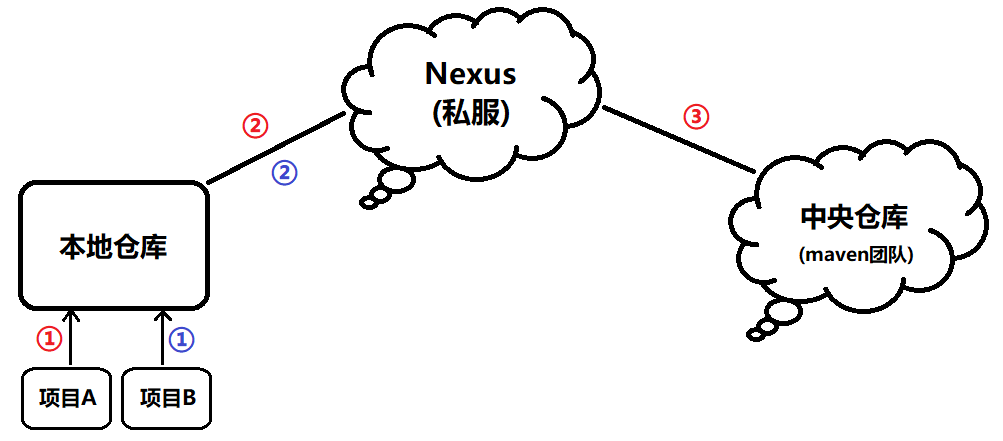
在上面的演示中， pom.xml文件中引入了spring的核心jar包，maven在下载spring核心jar包的同时，也会将spring核心jar包所依赖的相关jar包下载下来。

## Maven仓库

在上面所提到的本地仓库、远程仓库均是用来Maven用来更好的管理jar包的所采用的一种方式。下面来了解Maven的**三种仓库**，以及三种仓库之间的潜在联系。

### Maven的三种仓库

通过maven构建的项目，会通过项目中的pom.xml文件从远程仓库下载，并保存到本地仓库



**本地仓库**：默认的本地仓库位置在：c:/${user.dir}/.m2/repository，其中${user.dir}表示windows下的用户目录。本地仓库的作用是，用于保存（存储）从私服或者从中央仓库下载下来的jar包（或插件）。当项目中需要使用jar包和插件时，优先从本地仓库查找。

如果本地仓库中没有所需的jar包，可以到私服或者到中央仓库中下载后再保存到本地仓库。

**私服（Nexus）**：私服也叫做**镜像仓库**，私服一般由公司搭建并维护（也可以自己搭建）。比如达内有搭建自己的私服服务器（**http://maven.tedu.cn/nexus/content/groups/public/**），以及阿里云私服服务器（[**http://maven.aliyun.com/nexus**](http://maven.aliyun.com/nexus)**/content/groups/public/**）。

如果项目中使用到的jar包或者插件本地仓库没有，则可以到私服中下载，如果私服中有就直接将jar包保存到本地仓库中；而如果私服中也没有所需的jar包，就到公服上下载所需要的jar包，下载之后先在私服上保存一份，最后再保存到本地仓库。

**公服（中央仓库）**：公服也叫做**中央仓库**，在maven软件中内置一个远程仓库地址（<http://repo1.maven.org/maven2>）它就是中央仓库，服务于整个互联网，由Maven团队自己维护，里面存储了非常全的jar包，它包含了世界上大部分流行的开源项目的jar包。

那么我们在使用Maven构建的Java项目，项目中所使用的jar包会来自哪里呢？

例如，通过Maven先后构建项目A和项目B，在项目中都需要依赖第三方jar包：

1) 如果项目A中需要依赖第三方jar包，只需要在项目下的pom文件中引入jar包在**本地仓库**中的坐标即可使用。如果本地仓库没有所需要的jar包，则会连接私服（需要提前配置）下载所需jar包到本地仓库供项目使用。

2) 如果私服上也没有所需的jar包，则会连接中央仓库下载所需要的jar包保存到私服，再将jar包从私服下载至本地仓库，供项目使用。

3) 如果没有配置私服，则默认连接中央仓库下载所需要的jar包到本地仓库中供项目使用

4) 当项目B也需要依赖第三方jar包时，先到本地仓库中查找所需jar包，如果有则直接引用而无需再次下载，如果仍有部分jar包本地仓库中没有，则同上，即连接私服下载所需jar包到本地仓库。若私服中也没有所需jar包，则连接中央仓库下载jar包到私服，再从私服下载jar包到本地仓库中，供项目使用。

## 添加依赖

### 添加依赖：第一种方式

**1、添加依赖的第一种方式：**是直接在pom.xml文件中的<dependencies>标签内部添加。例如：在pom.xml文件中添加如下配置，就可以将junit单元测试的jar包引入到项目中来了。

添加依赖：

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

<scope>test</scope>

<dependency>

<dependencies>

2、手动添加依赖需要指定所依赖jar包的坐标，但是在大部分情况下，我们是不知道jar包的坐标的。可以通过访问如下网址，在互联网上搜索查询：

<http://mvnrepository.com>

或者在公司镜像仓库中搜索查询:

<http://maven.aliyun.com/nexus>

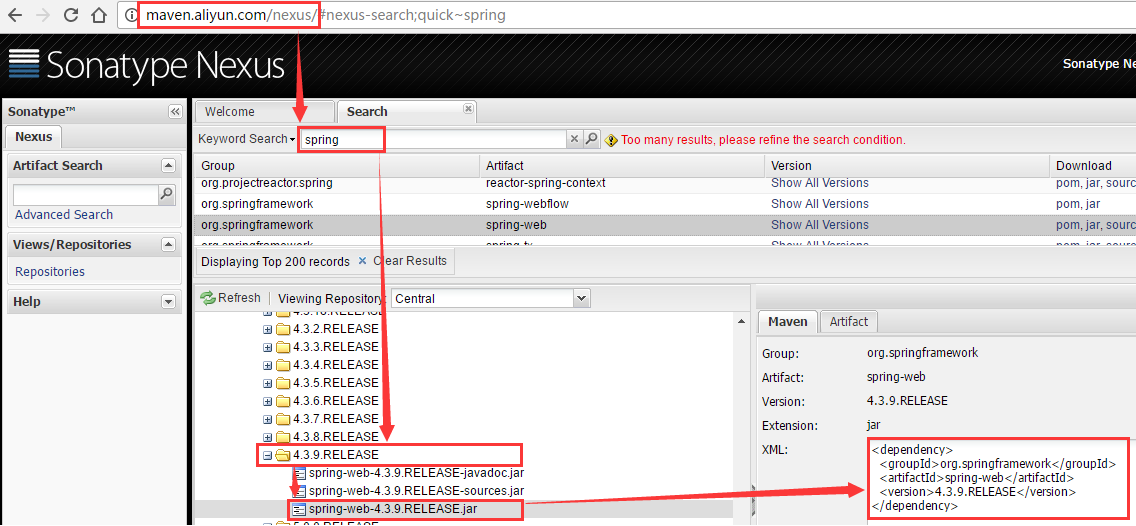
<http://maven.tedu.com/nexus>

4、示例：添加spring的jar包的坐标到项目中

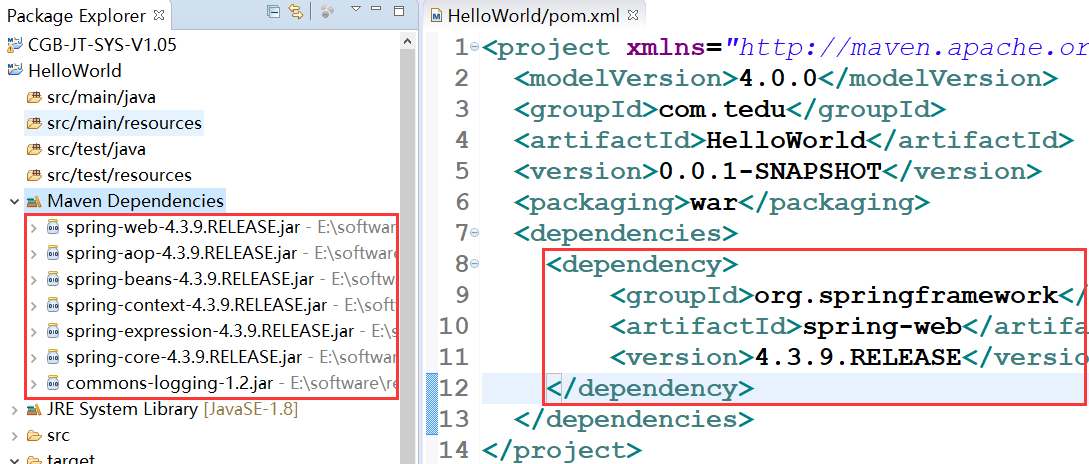
(1)访问上面其中的一个网址，在搜索框中搜索 "spring"

(2)在搜索出来的内容中，选择所需要的版本并点击版本，查看该版本的spring jar包所对应的坐标：

(3)将坐标直接拷贝到项目的pom.xml文件中即可：



5、将上图中右侧选中的坐标拷贝到pom.xml文件中:

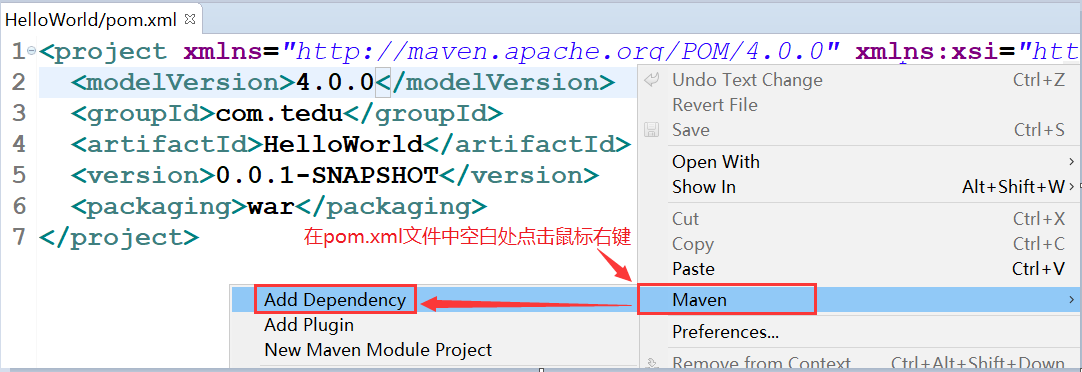


### 添加依赖：第二种方式

这种方式需要本地仓库中已经包含了该jar包，否则搜索不到!!!

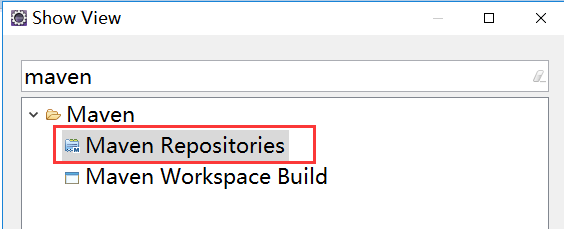
**1、添加依赖的第二种方式**：是使用maven插件的索引功能快速添加jar包

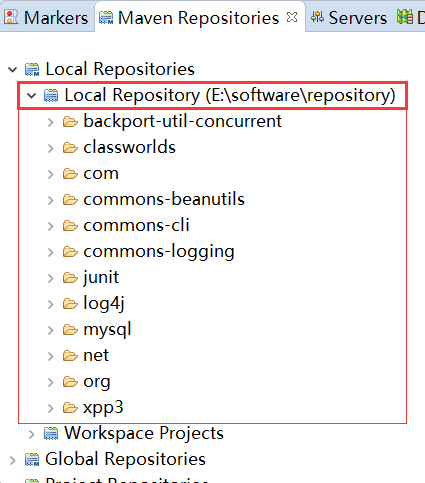
如果本地仓库中有我们需要的jar包，可以在项目中的pom.xml文件中空白处右键--> Maven --> Add Dependency在弹出的窗口中添加所需要的依赖(jar包)，如图：



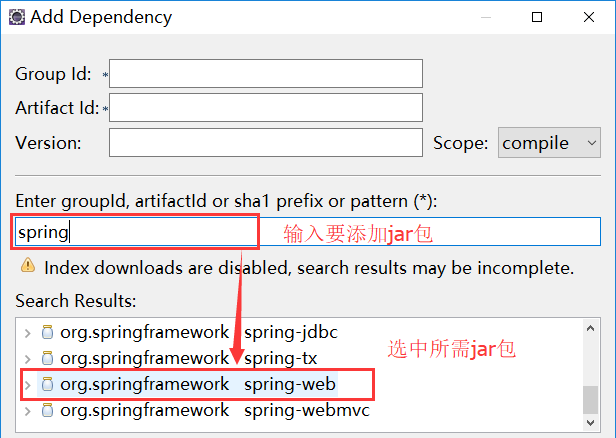
**2、添加依赖示例：**添加spring的jar包的坐标到项目中

(1)如果maven已经重建过索引，则可以直接执行第3步。若重建过索引，在 "Maven Repositories" 视图窗口中可以看到如下内容：

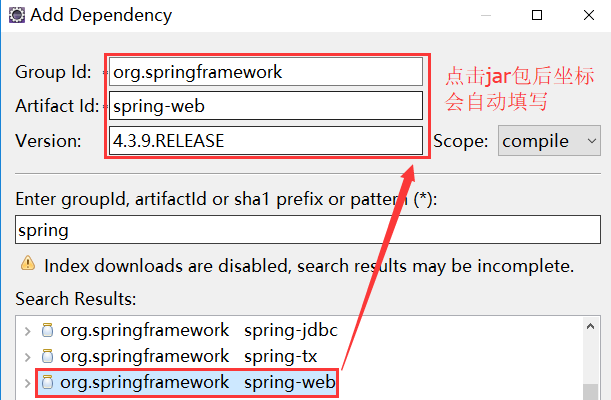




(2) 在项目中的pom.xml文件中右键 -> Maven -> Add Dependency，在弹出的窗口中输入 "spring":



选中要添加的jar包（坐标会自动填写），点击OK即可完成添加



**课堂练习：**在项目中的pom.xml文件中添加mysql驱动jar包和c3p0连接池jar包的坐标。

## 添加插件

Maven是个项目管理工具，如果不告诉Maven我们的代码要使用什么版本的JDK编译的话，Maven会使用默认的JDK版本来进行编译，这样容易产生版本不匹配的问题，甚至导致编译不通过的问题。如果在想配置项目使用的JDK版本，可以在pom.xml中进行！

除此之外，在运行项目时，需要启动tomcat服务器，Maven默认使用的tomcat版本是tomcat6，如果想更换tomcat版本也可以在pom.xml文件中进行配置！

### 引入JDK插件

在当前项目的pom.xml文件中，可以通过坐标的形式配置项目使用的JDK版本。例如：

<build>

<plugins>

<!-- 添加JDK插件(版本1.7) -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.7</source>

<target>1.7</target>

<encoding>UTF-8</encoding>

</configuration>

</plugin>

</plugins>

</build>

配置完后，保存pom.xml文件，Maven会自动到本地仓库引入JDK（如果本地仓库没有则会到私服或者远程仓库中下载）

### 引入tomcat插件

课程中使用的Maven，默认使用的是tomcat6，端口默认的8080，path默认的项目名称，如果想修改这三项，可以在build中添加插件来进行配置.

<!-- 添加tomcat服务器插件 -->

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<version>2.2</version>

<configuration>

<!-- 配置tomcat端口 -->

<port>8080</port>

<!-- Web应用的访问路径

<path>HelloWorld</path>

-->

</configuration>

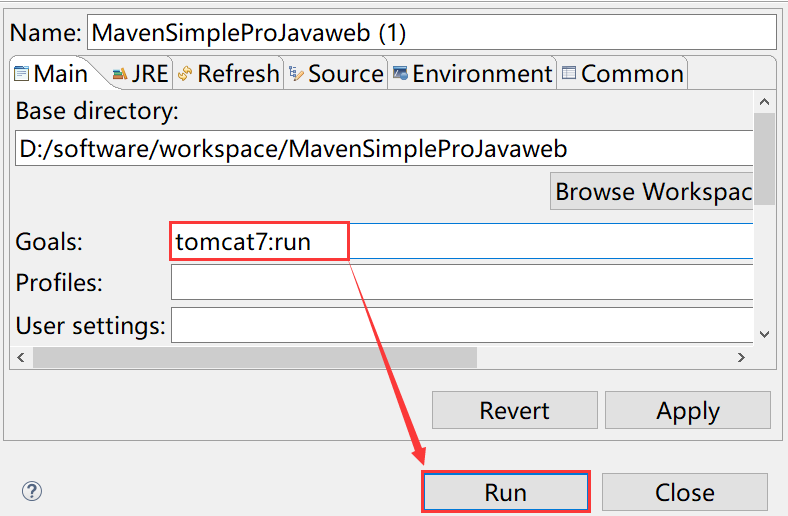
</plugin>

需要注意的是：如果使用项目tomcat7插件运行项目，在通过启动Tomcat运行项目时: 点击项目右键，选择 "Run As" ---> "Maven build..."



在弹出的窗口中配置如下内容（**tomcat7：run**），点击Run按钮启动tomcat服务器。

注意内容是：**tomcat7:run**



## FAQ：常见问题

### 常见的Maven环境问题

在实际开发中，Maven的环境很容易遭到破坏。而Maven的环境一旦遭到破坏，项目运行就会失败，例如在项目上右键点击运行，在项目启动的过程中，报了如下错误：

...

[INFO] ------------------------------------------------------------

[ERROR] **No plugin found** for prefix 'tomcat7' in the current project and in the plugin groups [**org.apache.maven.plugins**, **org.codehaus.mojo**] available from the repositories [local (C:\Users\bjzhangsz\.m2\repository), nexus-tedu (http://maven.tedu.cn/nexus/content/groups/public/)] -> [Help 1]

[ERROR]

[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.

[ERROR] Re-run Maven using the -X switch to enable full debug logging.

[ERROR]

[ERROR] For more information about the errors and possible solutions, please read the following articles:

[ERROR] [Help 1] <http://cwiki.apache.org/confluence/display/MAVEN/NoPluginFoundForPrefixException>

...

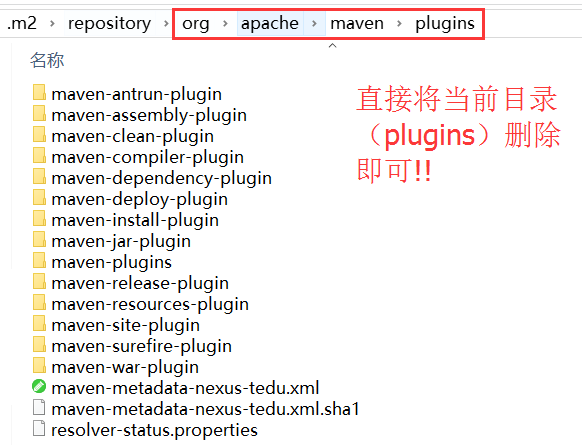
**问题分析：**从上面的错误信息中，我们可以提取主要的报错信息，可以发现是本地仓库中的 [org.apache.maven.plugins] 和 [org.codehaus.mojo] 两个目录中的插件找不到了。

**解决方案：**由于这两个目录中插件较多，不知道是哪一个插件丢失或遭破坏造成的，所以，我们可以将两个目录直接删除，让Maven再重新下载一次即可！

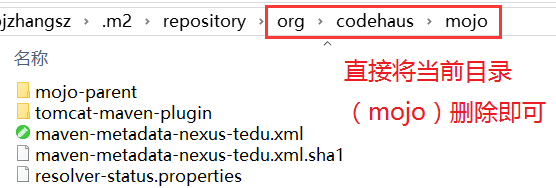
需要注意的是，此时直接删除会提示目录或文件正在被占用，需要将eclipse先关闭，待删除目录后，启动eclipse，重新更新工程（Update Project…），重新下载删除的内容即可解决！

(0)保存eclipse中的代码，关闭eclipse。

(1)删除本地仓库中的所缺少的目录, 比如上面提示: org.apache.maven.plugins(根据上面错误信息提示的jar包路径选择删除)目录：



(2)删除本地仓库中的org.codehaus.mojo(根据上面错误信息提示的jar包路径选择删除)目录：

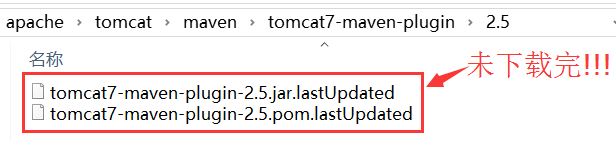


### 找不到jar包问题

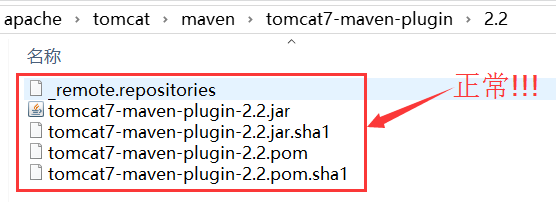
在项目中通过坐标引入了jar包（或者插件），并且本地仓库中也存在对应的jar包，但是项目还是报错，提示内容说找不到。

解决方法：如果引入的jar包，在本地仓库中存在，但是还是提示找不到，可以将本地仓库中jar包或插件的所在目录整个删除，重新保存pom.xml文件，并更新工程，让maven再次下载上面的jar包即可！

未下载完全示例：



正常下载完全示例：



### 拷贝别Maven仓库

如果因为网络环境的问题，导致jar包无法下载，也可以将别人下载好的（完整的）Maven的本地仓库拷贝过来，放在自己配置的本地仓库中。因为Maven可以支持拷贝别人的仓库。