我记得在搞懂maven之前看了几次重复的maven的教学视频。不知道是自己悟性太低还是怎么滴，就是搞不清楚，现在弄清楚了，基本上入门了。写该篇博文，就是为了帮助那些和我一样对于maven迷迷糊糊的人。有福了，看完基本上你就会发现原来这么简单。

参考博文：通俗理解maven

该篇文章篇幅很长，大概的思路如下

maven的介绍，初步认识，获取jar包的三个关键属性 --> 介绍仓库(获取的jar包从何而来)-->用命令行管理maven项目(创建maven项目) --> 用myeclipse创建maven项目 -->详细介绍pom.xml中的依赖关系(坐标获取、定位jar包的各种属性讲解。

--WH

# 一、简单的小问题?

解释之前，提1个小问题。

1.1、假如你正在Eclipse下开发两个Java项目，姑且把它们称为A、B，其中A项目中的一些功能依赖于B项目中的某些类，那么如何维系这种依赖关系的呢?

很简单，这不就是跟我们之前写程序时一样吗，需要用哪个项目中的哪些类，也就是用别人写好了的功能代码，导入jar包即可。所以这里也如此，可以将B项目打成jar包，然后在A项目的Library下导入B的jar文件，这样，A项目就可以调用B项目中的某些类了。

这样做几种缺陷

如果在开发过程中，发现B中的bug，则必须将B项目修改好，并重新将B打包并对A项目进行重编译操作

在完成A项目的开发后，为了保证A的正常运行，就需要依赖B(就像在使用某个jar包时必须依赖另外一个jar一样)，两种解决方案，第一种，选择将B打包入A中，第二种，将B也发布出去，等别人需要用A时，告诉开发者，想要用A就必须在导入Bjar包。两个都很麻烦，前者可能造成资源的浪费(比如，开发者可能正在开发依赖B的其它项目，B已经存储到本地了，在导入A的jar包的话，就有了两个B的jar)，后者是我们常遇到的，找各种jar包，非常麻烦(有了maven就不一样了)

1.2、我们开发一个项目，或者做一个小demo，比如用SSH框架，那么我们就必须将SSH框架所用的几十个依赖的jar包依次找出来并手动导入，超级繁琐。

上面两个问题的描述，其实都属于项目与项目之间依赖的问题**[A项目使用SSH的所有jar，就说A项目依赖SSH]**，人为手动的去解决，很繁琐，也不方便，所以使用maven来帮我们管理

# 二、maven到底是什么?

Maven是基于项目对象模型(POM project object model)，可以通过一小段描述信息（配置）来管理项目的构建，报告和文档的软件项目管理工具**[百度百科]**

这种又是大白话，如果没明白maven是什么，那么上面这句话跟没说一样，我自己觉得，Maven的核心功能便是合理叙述项目间的依赖关系，通俗点讲，就是通过pom.xml文件的配置获取jar包，而不用手动去添加jar包，而这里pom.xml文件对于学了一点maven的人来说，就有些熟悉了，怎么通过pom.xml的配置就可以获取到jar包呢？pom.xml配置文件从何而来？等等类

似问题我们需要搞清楚，如果需要使用pom.xml来获取jar包，那么首先该项目就必须为maven项目，maven项目可以这样去想，就是在java项目和web项目的上面包裹了一层maven，本质上java项目还是java项目，web项目还是web项目，但是包裹了maven之后，就可以使用maven提供的一些功能了(通过pom.xml添加jar包)。

所以，根据上一段的描述，我们最终的目的就是学会如何在pom.xml中配置获取到我们想要的jar包，在此之前我们就必须了解如何创建maven项目，maven项目的结构是怎样，与普通java,web项目的区别在哪里，还有如何配置pom.xml获取到对应的jar包等等，这里提前了解一下我们如何通过pom.xml文件获取到想要的jar的，具体后面会详细讲解该配置文件。

pom.xml获取junit的jar包的编写。

```
<dependencies>    //所要依赖的jar统一放在这个下面，一般说是依赖的构件，其实就是jar
  <dependency>        //依赖的jar，这里编写的是junit这个jar
  //通过groupId、artifactId、version三个属性来定位一个jar包。
    <groupId>junit</groupId>       //groupId:一般为包名，也就是域名的反写，
    <artifactId>junit</artifactId>    //artifactId:项目名
    <version>3.8.1</version>  //所需要jar的版本
    <scope>test</scope>        //这个暂时不讲，后面讲解，意思是该jar包只在测试的时候用。
  </dependency>

  <dependency>
     //其他所要依赖的jar就在dependency下编写，而dependency又统一放在dependencies下，不难理解
  </dependency>

  <dependency>
     //其他所要依赖的jar就在dependency下编写，而dependency又统一放在dependencies下，不难理解
  </dependency>

  ....

</dependencies>
```

为什么通过groupId、artifactId、version三个属性就能定位一个jar包?

加入上面的pom.xml文件属于A项目，那么A项目肯定是一个maven项目，通过上面这三个属性能够找到junit对应版本的jar包，那么junit项目肯定也是一个maven项目，junit的maven项目中的pom.xml文件就会有三个标识符，比如像下图这样，然后别的maven项目就能通过这三个属性来找到junit项目的jar包了。所以，在每个创建的maven项目时都会要求写上这三个属性值的。

```
//会有这三个属性来标识自己，目的就是为了让别人能通过这三个属性找到自己
<groupId>junit</groupId>  //给自己一个唯一标识
<artifactId>junit</artifactId>    //自己项目名称
<version>3.8.1</version>  //版本号
<packaging>jar</packaging>    //打包后为jar包
```

# 三、maven的安装

这一步maven环境的配置，我觉得有必要安装一下，目的为了使用命令行创建maven项目，和使用命令行操作maven项目。这里不细讲，给出链接，跟安装jdk环境类似，[maven的安装教程和配置]

还有注意，我以下用的是maven3.0.4版本(比较低的)，你们可以下载最新的版本，最好是使用jdk1.7.

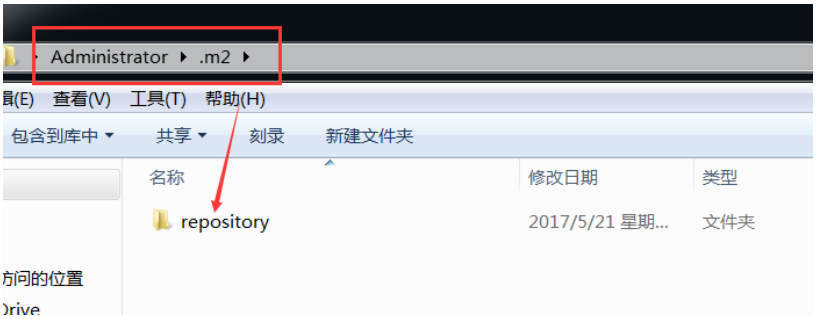
# 四、仓库的概念

通过pom.xml中的配置，就能够获取到想要的jar包(还没讲解如何配置先需要了解一下仓库的概念)，但是这些jar是在哪里呢？就是我们从哪里获取到的这些jar包？答案就是仓库。

仓库分为：本地仓库、第三方仓库(私服)、中央仓库

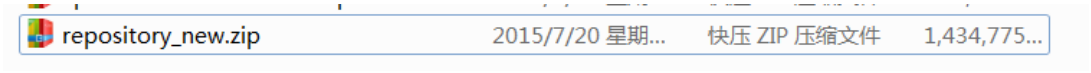
4.1、本地仓库

Maven会将工程中依赖的构件(Jar包)从远程下载到本机一个目录下管理，每个电脑默认的仓库是在 $user.home/.m2/repository下



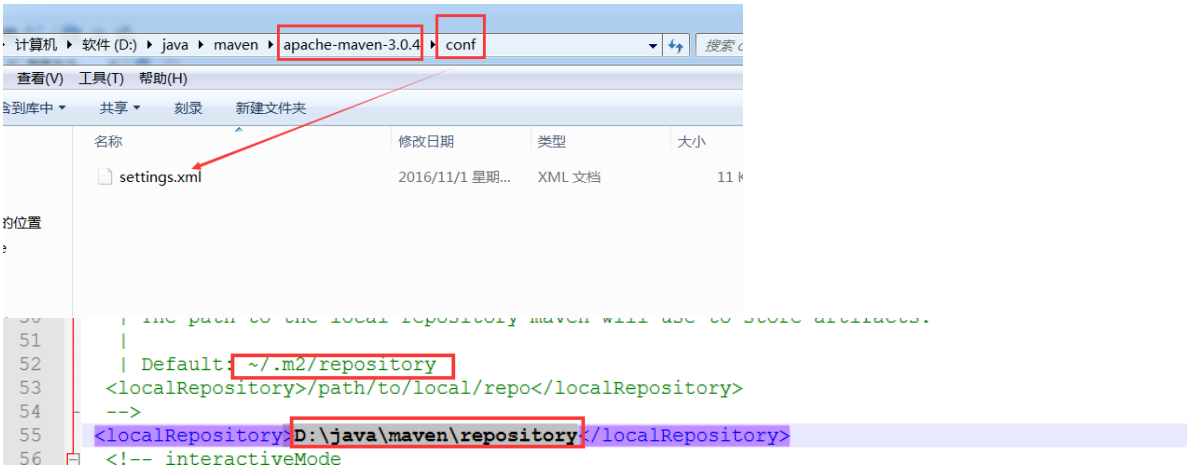

例如我的就在：C:\Users\Administrator\.m2\repository

一般我们会修改本地仓库位置，自己创建一个文件夹，在从网上下载一个拥有相对完整的所有jar包的结合，都丢到本地仓库中，然后每次写项目，直接从本地仓库里拿就行了




这里面有很多各种各样我们需要的jar包。

修改本地库位置：在$MAVEN_HOME/conf/setting.xml文件中修改，




D:\java\maven\repository：就是我们自己创建的本地仓库，将网上下载的所有jar包，都丢到该目录下，我们就可以直接通过maven的pom.xml文件直接拿。

4.2、第三方仓库

第三方仓库，又称为内部中心仓库，也称为私服

私服：一般是由公司自己设立的，只为本公司内部共享使用。它既可以作为公司内部构件协作和存档，也可作为公用类库镜像缓存，减少在外部访问和下载的频率。（使用私服为了减少对中央仓库的访问

私服可以使用的是局域网，中央仓库必须使用外网

也就是一般公司都会创建这种第三方仓库，保证项目开发时，项目所需用的jar都从该仓库中拿，每个人的版本就都一样。

注意：连接私服，需要单独配置。如果没有配置私服，默认不使用

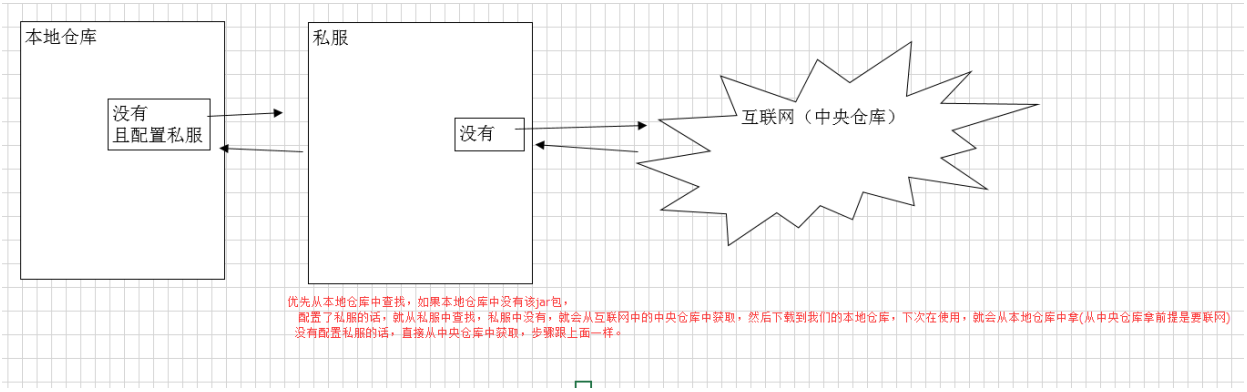
4.3、中央仓库

Maven内置了远程公用仓库：http://repo1.maven.org/maven2

这个公共仓库是由Maven自己维护，里面有大量的常用类库，并包含了世界上大部分流行的开源项目构件。目前是以java为主

工程依赖的jar包如果本地仓库没有，默认从中央仓库下载

总结：获取jar包的过程




# 五、使用命令行管理maven项目

## 5.1、创建maven java项目

自己创建一个文件夹，在该文件夹下按shift+右击，点开使用命令行模式，这样创建的maven[java]项目就在该文件夹下了。

命令：mvn archetype:create -DgroupId=com.wuhao.maven.quickstart -DartifactId=simple -DarchetypeArtifactId=maven-archetype-quickstart

mvn：核心命令

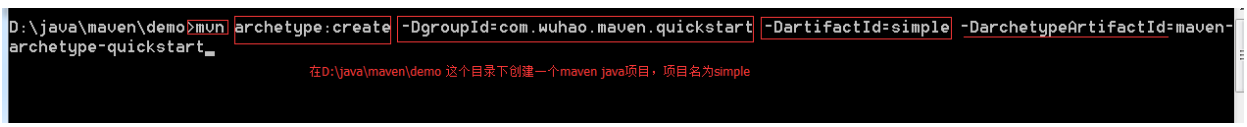
archetype:create：创建项目，现在maven高一点的版本都弃用了create命令而使用generate命令了。

-DgroupId=com.wuhao.maven.quickstart ：创建该maven项目时的groupId是什么，该作用在上面已经解释了。一般使用包名的写法。因为包名是用公司的域名的反写，独一无二

-DartifactId=simple：创建该maven项目时的artifactId是什么，就是项目名称

-DarchetypeArtifactId=maven-archetype-quickstart：表示创建的是[maven]java项目

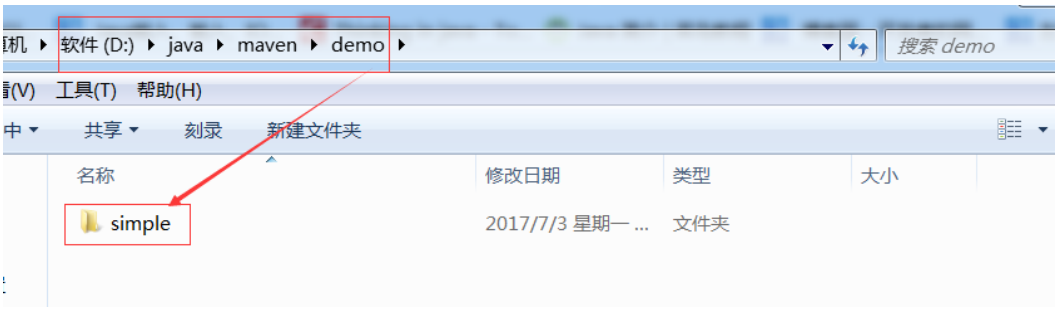
运行的前提：需要联网，必须上网下载一个小文件




运行成功后

```
D:\java\maven\demo>mvn archetype:create -DgroupId=com.wuhao.maven.quickstart -DartifactId=simple -DarchetypeArtifactId=maven-
archetype-quickstart
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-archetype-plugin:2.2:create (default-cli) @ standalone-pom ---
[WARNING] This goal is deprecated. Please use mvn archetype:generate instead
[INFO] Defaulting package to group ID: com.wuhao.maven.quickstart
[INFO] ------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:RELEASE
[INFO] ------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: D:\java\maven\demo
[INFO] Parameter: package, Value: com.wuhao.maven.quickstart
[INFO] Parameter: groupId, Value: com.wuhao.maven.quickstart   groupId
[INFO] Parameter: artifactId, Value: simple   artifactId
[INFO] Parameter: packageName, Value: com.wuhao.maven.quickstart
[INFO] Parameter: version, Value: 1.0-SNAPSHOT   version版本
[INFO] project created from Old (1.x) Archetype in dir: D:\java\maven\demo\simple
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS   成功
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.815s
[INFO] Finished at: Mon Jul 03 22:16:17 CST 2017
[INFO] Final Memory: 12M/232M
[INFO] ------------------------------------------------------------------------
D:\java\maven\demo>
```

在D:\java\maven\demo下就会生成一个simple的文件，该文件就是我们的maven java项目




## 5.2、maven java项目结构

simple

---pom.xml 核心配置，项目根下

---src

---main

---java java源码目录

---resources java配置文件目录

---test

---java 测试源码目录

---resources 测试配置目录

| | |
|---|---|
| pom.xml | 核心配置，项目根下 |
| src/main/java | java 源码目录 |
| src/main/resources | java 配置文件目录 |
| src/test/java | 测试源码目录 |
| src/test/resources | 测试配置目录 |
| target | 输出目录 |

图中有一个target目录，是因为将该java项目进行了编译，src/main/java下的源代码就会编译成.class文件放入target目录中，target就是输出目录。

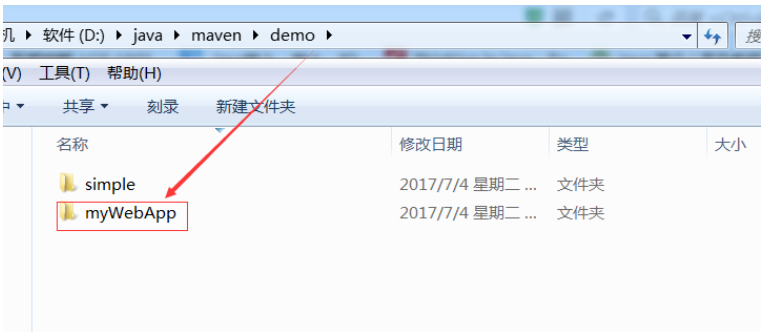
## 5.3、创建 maven web 项目

命令：mvn archetype:create -DgroupId=com.wuhao.maven.quickstart -DartifactId=myWebApp -DarchetypeArtifactId=maven-archetype-webapp -Dversion=0.0.1-snapshot

其他都差不多，创建maven web项目的话 -DarchetypeArtifactId=maven-archetype-webapp 比创建java项目多了一个 -Dversion=0.01-snapshot， 在创建java项目的时候也可以加上这个，如果不写，会默认帮我们加上1.0-snapshot。

```
D:\java\maven\demo>mvn archetype:create -DgroupId=com.wuhao.maven.quickstart -DartifactId=myWebApp -DarchetypeArtifactId=mave
n-archetype-webapp -Dversion=0.0.1-snapshot
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-archetype-plugin:2.2:create (default-cli) @ standalone-pom ---
[WARNING] This goal is deprecated. Please use mvn archetype:generate instead
[INFO] Defaulting package to group ID: com.wuhao.maven.quickstart
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-webapp:RELEASE
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: D:\java\maven\demo
[INFO] Parameter: package, Value: com.wuhao.maven.quickstart
[INFO] Parameter: groupId, Value: com.wuhao.maven.quickstart
[INFO] Parameter: artifactId, Value: myWebApp
[INFO] Parameter: packageName, Value: com.wuhao.maven.quickstart
[INFO] Parameter: version, Value: 0.0.1-snapshot
[INFO] project created from Old (1.x) Archetype in dir: D:\java\maven\demo\myWebApp
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.856s
[INFO] Finished at: Mon Jul 03 22:50:20 CST 2017
[INFO] Final Memory: 11M/184M
[INFO] ------------------------------------------------------------------------
D:\java\maven\demo>
```

在D:\java\maven\demo下就会生成一个myWebApp的文件




## 5.4、maven web项目结构

pom.xml 核心配置

src/main/java java源码

src/main/resources java配置

src/main/webapp myeclipse web项目中 WebRoot目录

|-- WEB-INF

|-- web.xml

src/test 测试

target 输出目录

## 5.5、命令操作maven java或web项目

编译：mvn compile　　--src/main/java目录java源码编译生成class（target目录下）

测试：mvn test　　--src/test/java 目录编译

清理：mvn clean　　--删除target目录，也就是将class文件等删除

打包：mvn package　　--生成压缩文件：java项目#jar包；web项目#war包，也是放在target目录下

安装：mvn install　　--将压缩文件(jar或者war)上传到本地仓库

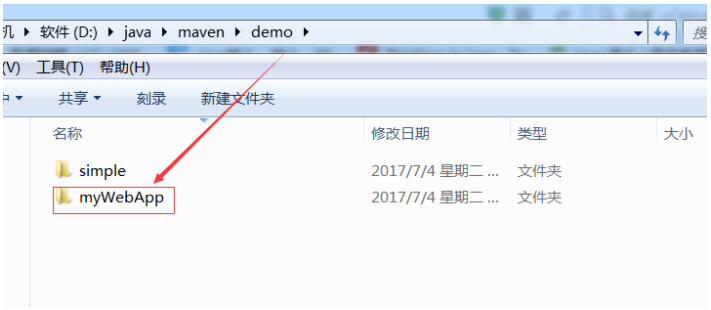
部署|发布：mvn deploy　　--将压缩文件上传私服

## 5.6、例子:使用命令操作maven java项目

注意：使用命令时，必须在maven java项目的根目录下，及可以看到pom.xml

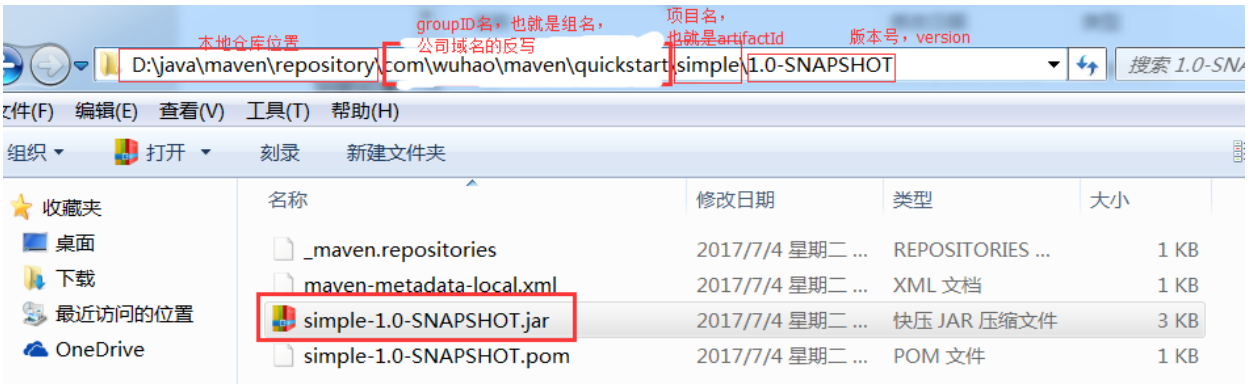
描述：将maven java项目打包上传到本地仓库供别人调用

使用 mvn install




在本地仓库中查看是否有该项目

D:\java\maven\repository\com\wuhao\maven\quickstart\simple\1.0-SNAPSHOT




通过在本地仓库中的目录可以发现为什么通过groupId、artifactId、version可以定位到仓库中得jar包，也可以知道为什么groupId要使用公司域名的反写(因为这样唯一，不会与别的项目重名导致查找到的内容不精确)

## 5.7、maven项目的完整生命周期，当执行生命周期后面命令时，**前面步骤的命令自动执行**

- validate
- generate-sources
- process-sources
- generate-resources
- process-resources　复制并处理资源文件，至目标目录，准备打包。
- compile　编译项目的源代码。
- process-classes
- generate-test-sources
- process-test-sources
- generate-test-resources
- process-test-resources　复制并处理资源文件，至目标测试目录。
- test-compile　编译测试源代码。
- process-test-classes
- test　使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。
- prepare-package
- package　接受编译好的代码，打包成可发布的格式，如 JAR 。
- pre-integration-test
- integration-test
- post-integration-test
- verify
- install　将包安装至本地仓库，以让其它项目依赖。
- deploy　将最终的包复制到远程的仓库，以让其它开发人员与项目共享。

红色标记字体的意思就是当我们直接使用mvn install命令对项目进行上传至本地仓库时，那么前面所有的步骤将会自动执行，比如源代码的编译，打包等等。

## 5.8、其他命令

maven java或web项目转换Eclipse工程

mvn eclipse:eclipse

mvn eclipse:clean　清楚eclipse设置信息，又从eclipse工程转换为maven原生项目了

...转换IDEA工程

mvn idea:idea

mvn idea:clean　同上

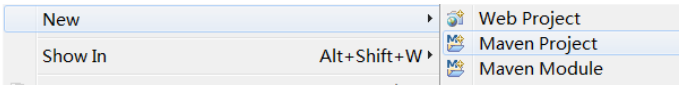
# 六、使用Myeclipse创建maven自定义项目

使用myeclpse创建项目前，需要在myeclipse中配置maven的一些信息

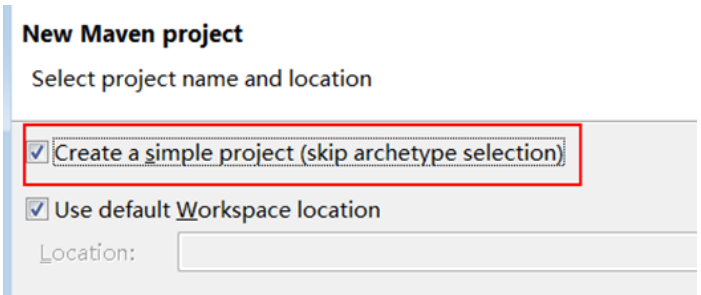
比如：配置本地仓库、安装自定义maven(myeclipse中高版本自带了maven)等，这里省略。

## 6.1、java项目

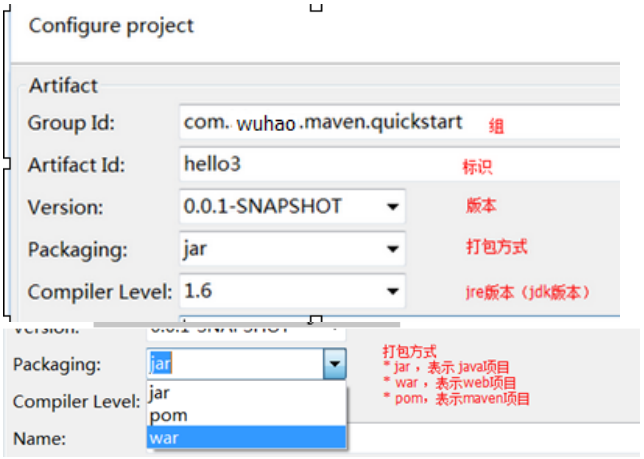
1、选择maven project，如果右键新建没有，通过other获得




2、创建一个简单项目（跳过骨架选择）



3、设置项目参数，创建java项目




4、创建java项目结果

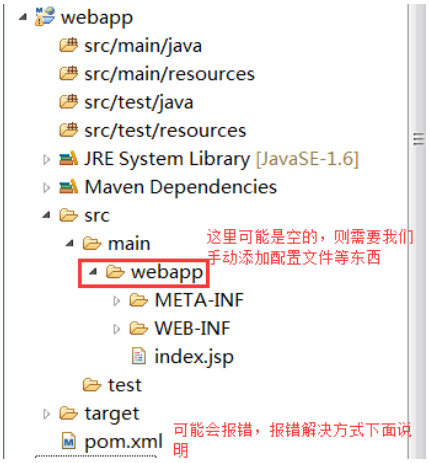

## 6.2、创建maven web项目

1、同上

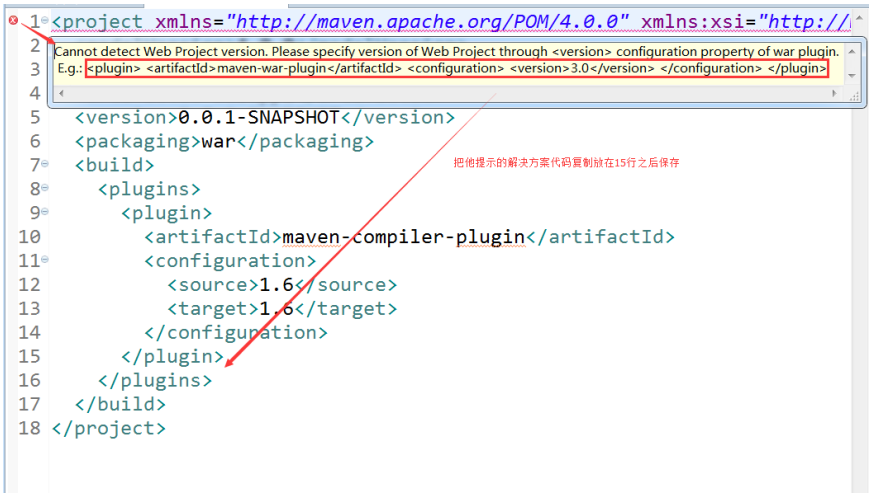
2、同上

3、设置项目参数，其他一样，选择打包方式不一样。


4、创建web项目结果




5、可能报错1：pom.xml报错

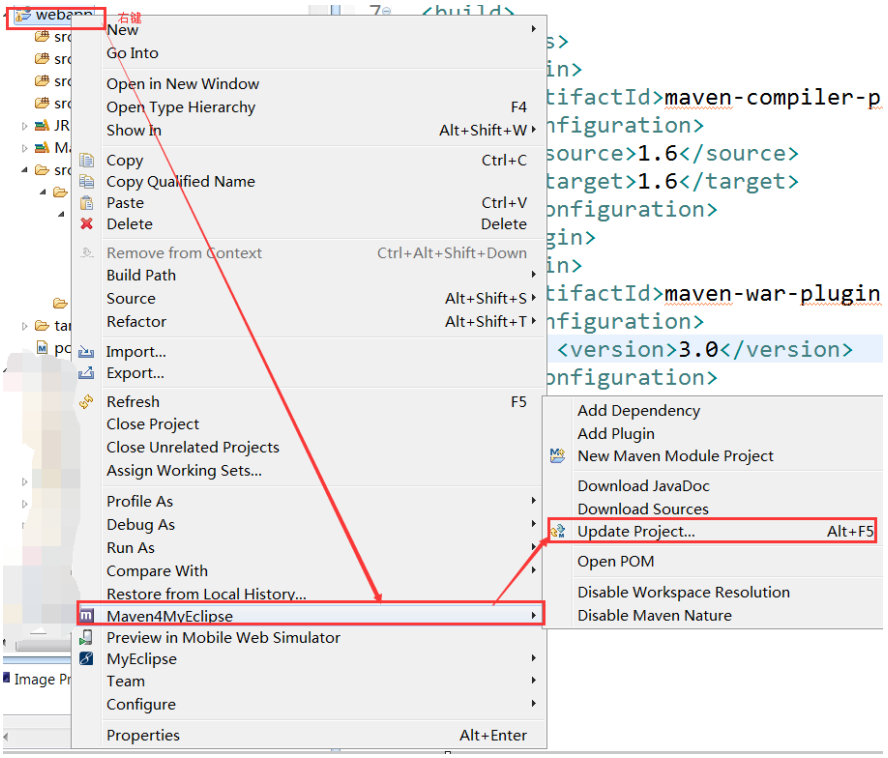



```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

结果如下

```
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <configuration>
            <version>3.0</version>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

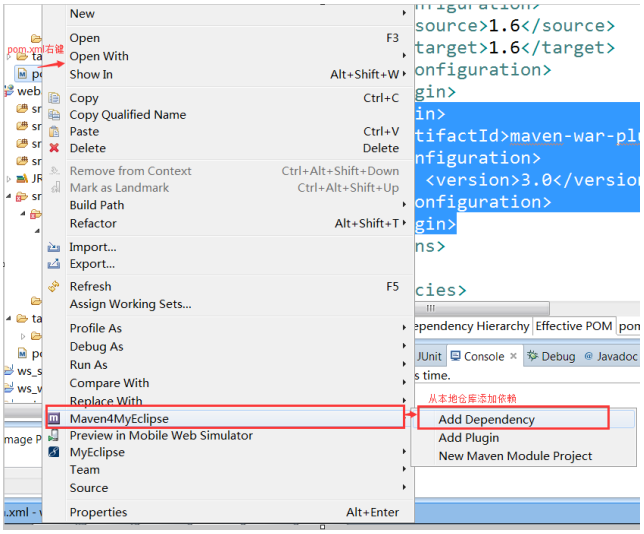
然后需要更新一下项目，就不报错了。



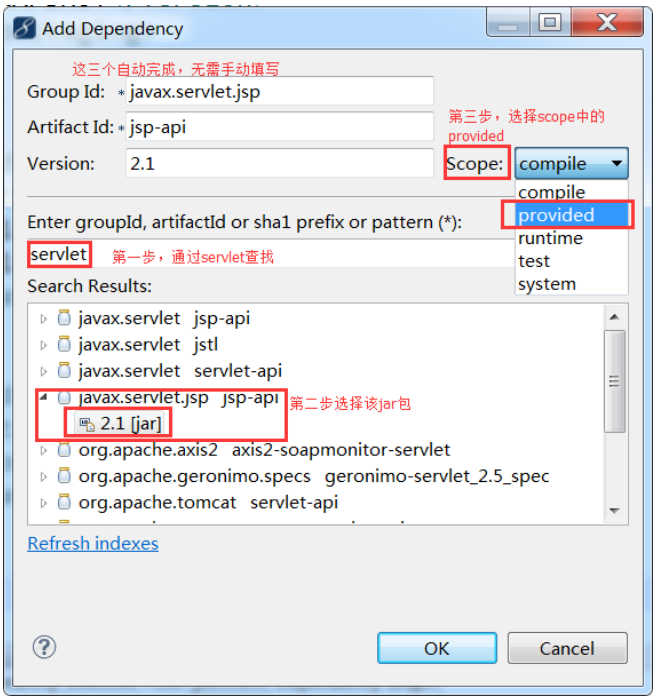

6、报错2，编写jsp时报错


错误信息为缺少serlvet-jar和jsp-jar包

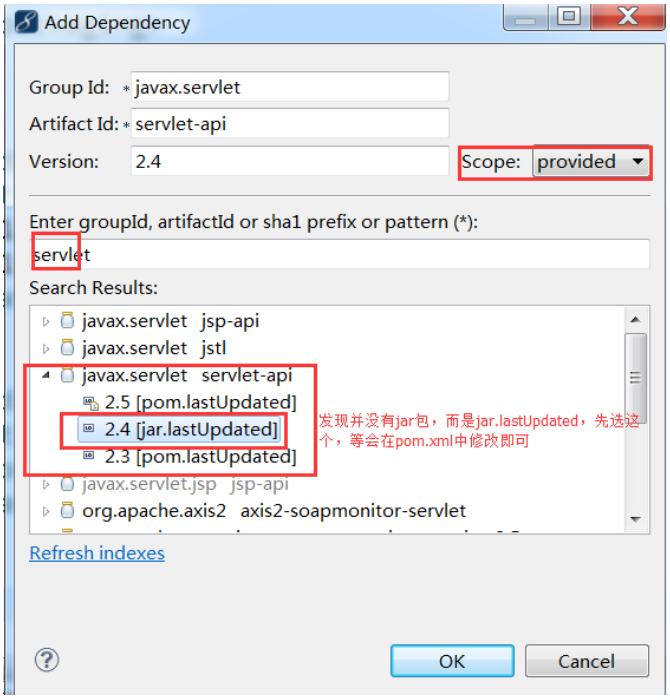

解决：使用pom.xml定位导入相应的jar包




找到jsp-jar包，scope的作用是表示该jar包的作用范围，provided表示在编译和测试期有效，在运行期则无效。具体后面讲解pom.xml时会讲解到。




找到servle-jar包



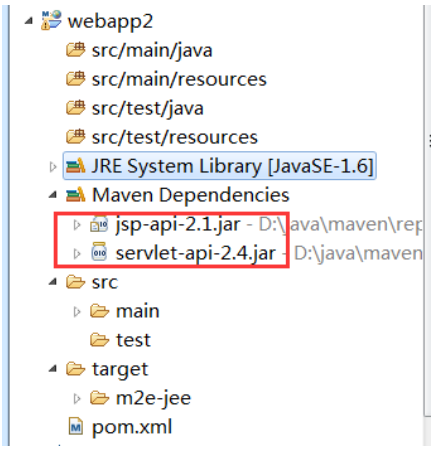

pom.xml中检查，修改

```
24  <dependencies>
25    <dependency>
26        <groupId>javax.servlet.jsp</groupId>
27        <artifactId>jsp-api</artifactId>
28        <version>2.1</version>
29        <scope>provided</scope>
30    </dependency>
31    <dependency>
32        <groupId>javax.servlet</groupId>
33        <artifactId>servlet-api</artifactId>
34        <version>2.4</version>
35        <type>jar.lastUpdated</type>
36        <scope>provided</scope>
37    </dependency>
38  </dependencies>
```

我们添加的两个依赖。查看，是否需要修改代码。

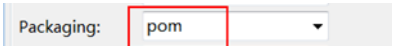
将35行代码删除，不然jar包无法加载进来。

修改完后，发现两个jar包都加载进来了，项目完好，不在报错了。

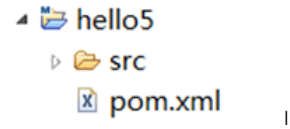



## 6.3、创建maven项目

都一样，在项目参数那里修改即可

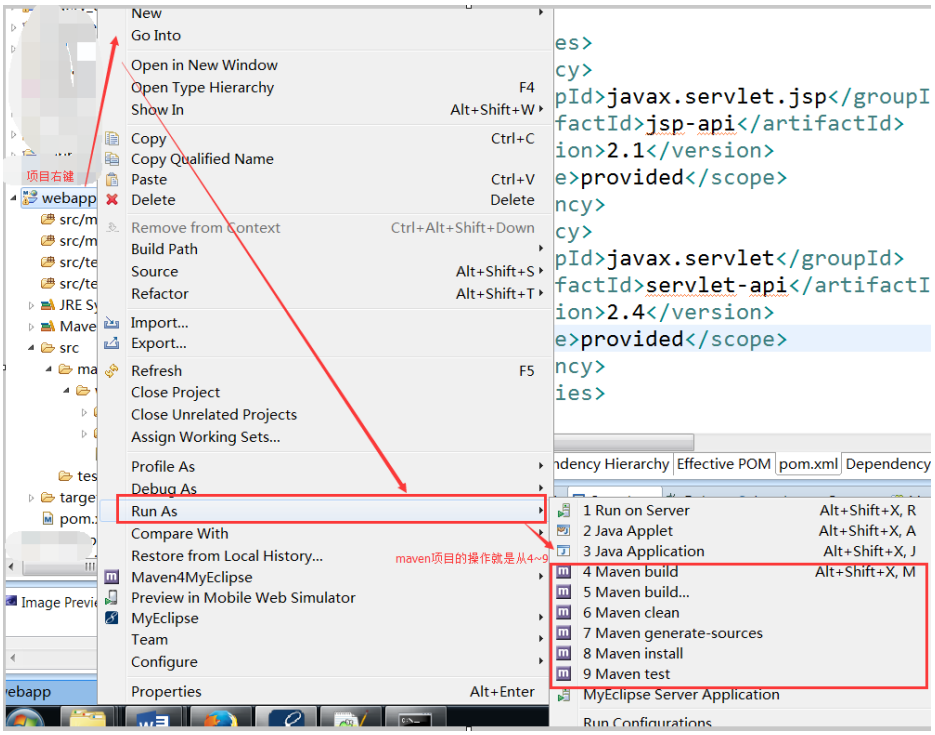



结果




maven项目一般没用，在开发中将一个项目拆分成多个项，就需要使用maven项目（pom项目）将其他子项目进行整合，下一章节讲解，很重要。很重要。

## 6.4、myeclipse maven操作




6--9 都是快捷方式

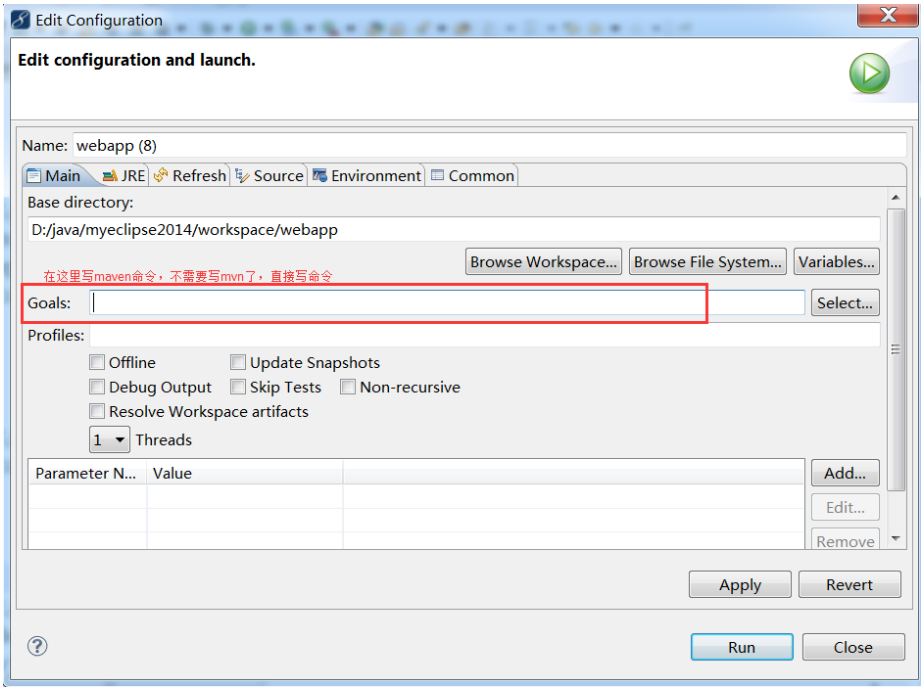
9 测试，相当于命令行 mvn test

8 安装，相当于命令行 mvn install 作用：将其上传到本地仓库，具体见上面讲解

7 关联源码，这个不需要解释吧，平常我们使用别的jar包也关联过源码

6 清理，mvn clean

5 maven bulid 执行maven命令，等效 mvn




4 maven build  5快速的操作

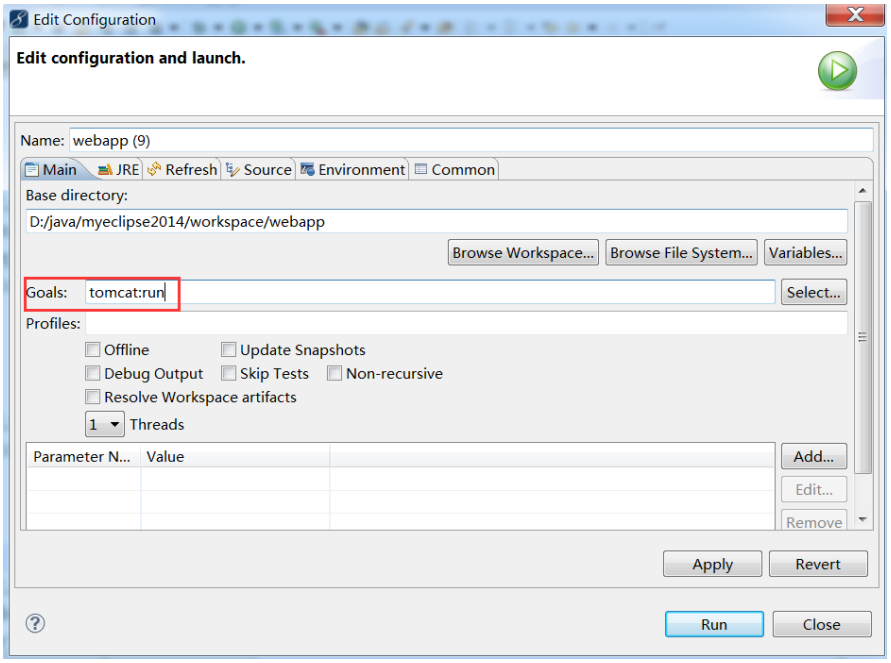
如果没有操作过，与5相同

如果操作过一次，将直接执行上一次5的命令

如果操作多次，将提供选择框

## 6.5、例子，将maven web项目发布到tomcat运行

命令：tomcat:run




通过网址即可访问，同时会将该项目上传到本地仓库。

# 七、pom.xml的依赖关系讲解(重点)

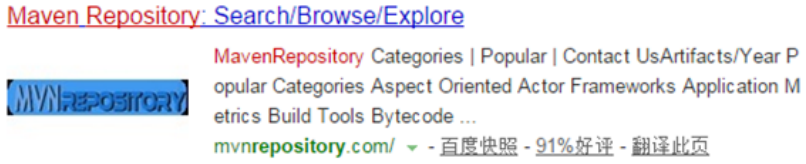
之前一直在使用pom.xml中找jar包最关键的三个属性，groupId、artifactId、version，应该有些印象了，也知道为什么通过这三个能找到对应的jar包，但是没有细讲其中的一些小的知

识点，比如上面添加servlet-jar和jsp-jar的依赖时，出现的一些属性就不太懂，所以，这一章节，就将依赖关系全面分析。

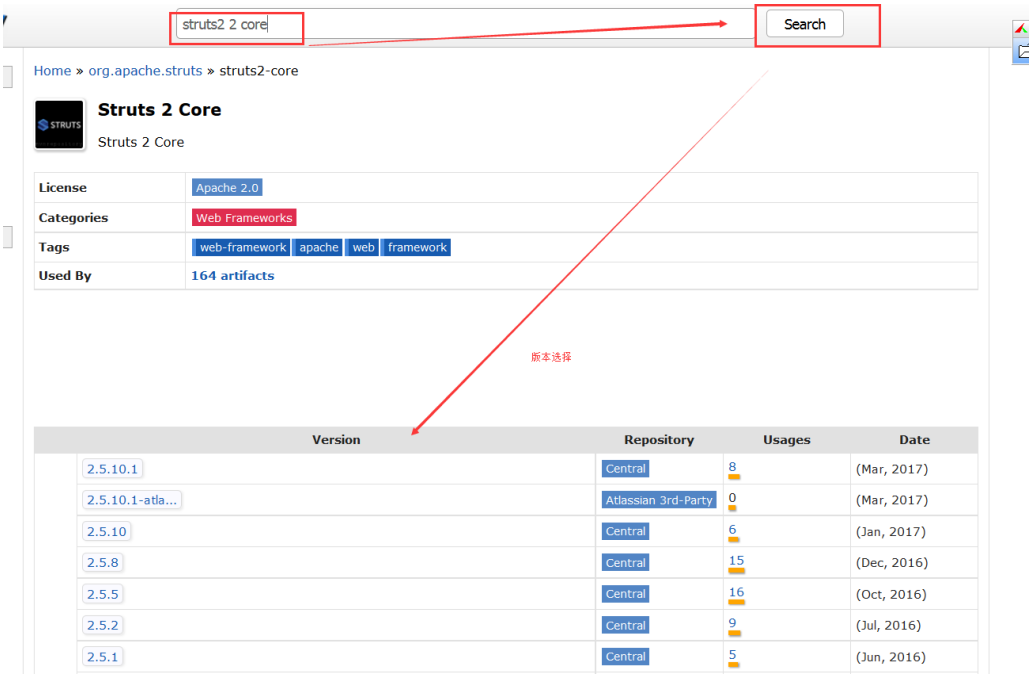
## 7.1、如何获取坐标(也就是三个关键属性值)

方式1：使用网站搜索[从中央仓库拿]

步骤一：百度搜索关键字“maven repository”




步骤二：输入关键字查询获得需要内容，确定需要版本

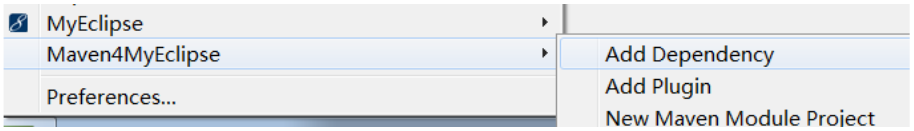



步骤三、获得坐标

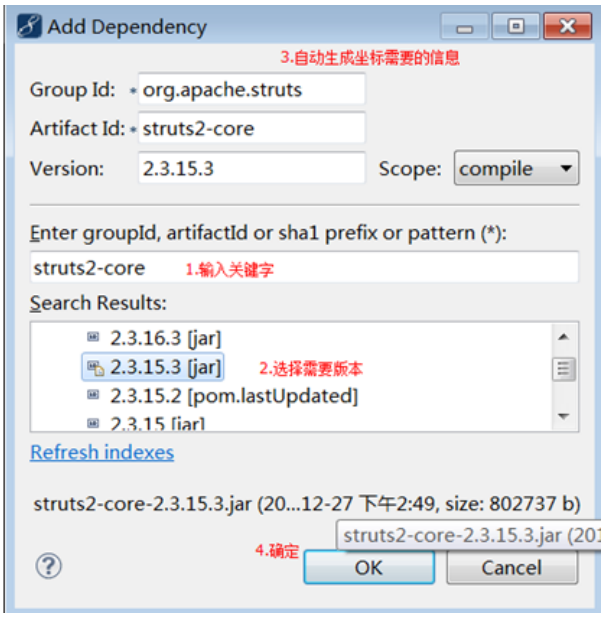

方式2、使用本地仓库，通过myeclipse获得坐标

上面已经介绍过了如何从本地仓库获取对应jar，这里在简单阐述一下

步骤一：添加依赖，pom.xml文件中，右键




步骤二：获得坐标




## 7.2、依赖(坐标)的常见配置

为了避免不知道说的哪些配置属性，看下面图就明白了，就是dependency下的属性配置，全部有9个，讲其中的7个。


```
23
24  <dependencies>
25      <dependency>
26          <groupId>javax.servlet.jsp</groupId>
27          <artifactId>jsp-api</artifactId>
28          <version>2.1</version>
29          <scope>provided</scope>
30          <!-- type、optional等等属性，这就是依赖(坐标)的配置 -->
31      </dependency>
32      <dependency>
33          <groupId>javax.servlet</groupId>
34          <artifactId>servlet-api</artifactId>
35          <version>2.4</version>
36          <scope>provided</scope>
37      </dependency>
```

groupId、artifactId、version是依赖的基本坐标，缺一不可，这三个可以不用将，都知道，重要的是除了这三个之外的配置属性需要我们理解

type：依赖的类型，比如是jar包还是war包等

默认为jar，表示依赖的jar包

注意：pom.lastUpdated 这个我们在上面添加servlet-jar的时候就遇到过，看到lastUpdated的意思是表示使用更新描述信息，占位符作用，通俗点讲，选择该类型，jar包不会被加载进来，只是将该jar包的一些描述信息加载进来，使别的jar包在引用他时，能够看到一些相关的提示信息，仅此而已，所以说他是个占位符，只要记住他的jar包不会被加载进来。

optional：标记依赖是否可选。默认值false

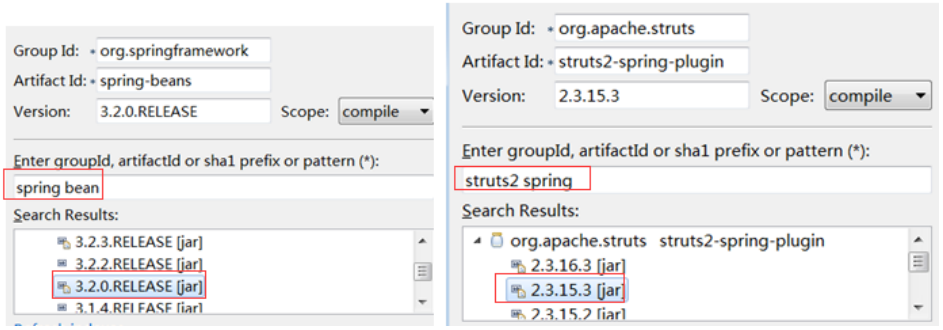
比如struts2中内置了log4j这个记录日志的功能，就是将log4j内嵌入struts2的jar包中，而struts2有没有log4j这个东西都没关系，有它，提示的信息更多，没它，也能够运行，只是提示的信息就相对而言少一些，所以这个时候，就可以对它进行可选操作，想要它就要，不想要，就设置为false。

exclusions：排除传递依赖，解决jar冲突问题

依赖传递的意思就是，A项目 依赖 B项目，B项目 依赖 C项目，当使用A项目时，就会把B也给加载进来，这是传递依赖，依次类推，C也会因此给加载进来。

这个有依赖传递有好处，也有坏处，坏处就是jar包的冲突问题，比如，A 依赖 B(B的版本为1)，C 依赖 B(B的版本为2)，如果一个项目同时需要A和C，那么A,C都会传递依赖将B给加载进来，问题就在这里，两个B的版本不一样，将两个都加载进去就会引起冲突，这时候就需要使用exclusions这个属性配置了。maven也会有一个机制避免两个都加载进去，maven 默认配置在前面的优先使用，但是我们还是需要使用exclusions来配置更合理，这里使用spring bean 和 struts2 spring plugin 来举例子说明这个问题并使用exclusions解决这个问题。（spring bean 和 struts2 spring plugin都需要依赖spring-core，但版本不一样）

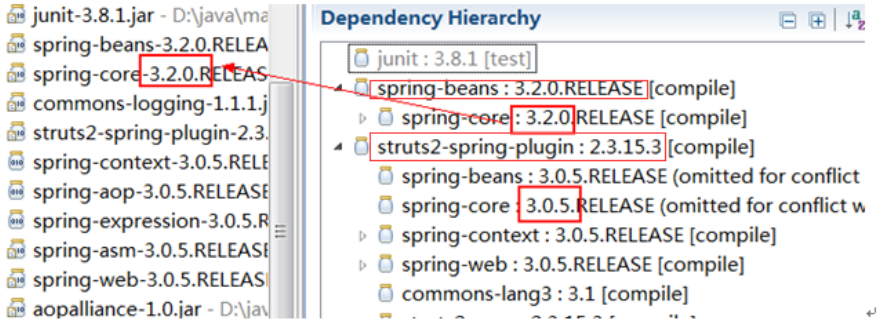
从本地仓库中找到这两个jar包



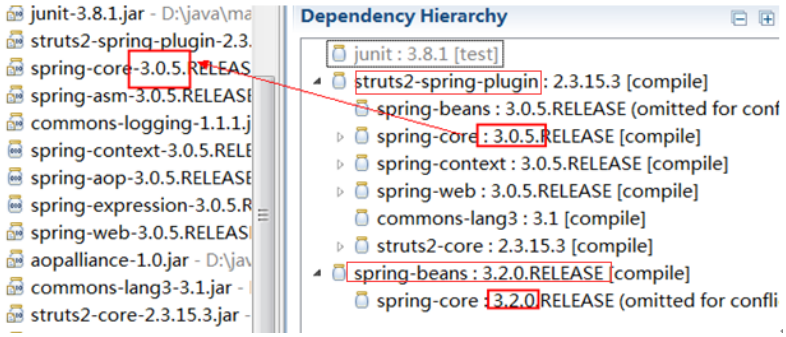

maven自己的解决方案如下

maven 默认配置在前面的优先使用，下面是证明

先将spring-beans加载进去的，所以会将spring-beans依赖的spring-core的版本加载进来。

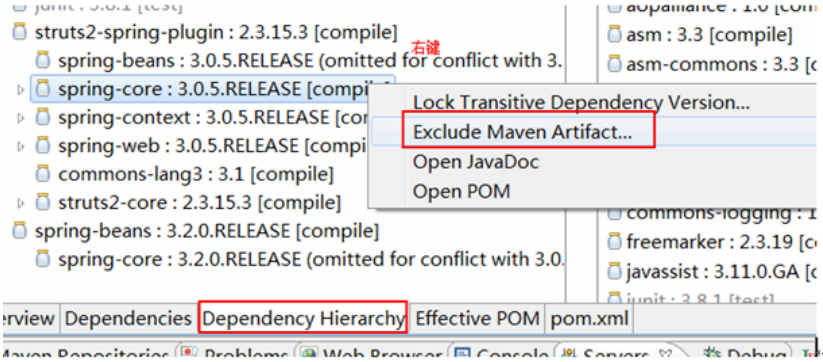



先将struts2-spring-plugin加载进来，那么就会将其依赖的spring-core的版本加载进来




使用exclusions来配置

即使struts2-spring-plugin 配置在前面，也需要使用3.2.0版本。则需要为struts2-spring-plugin 排除依赖（不使用3.0.5依赖）




注意：这样，就将struts2-spring-plugin依赖的spring-core的版本排除依赖了，也就是该依赖的spring-core不会在加载进来，查看代码，看是否符合要求，如果不符合要求，需要手动的修改

```
37      <!-- struts2-spring-plugin -->
38      <dependency>
39          <groupId>org.apache.struts</groupId>
40          <artifactId>struts2-spring-plugin</artifactId>
41          <version>2.3.15.2</version>
42          <exclusions>
43              <exclusion>
44                  <artifactId>spring-core</artifactId>
45                  <groupId>org.springframework</groupId>
46              </exclusion>
47          </exclusions>
48      </dependency>
49      <!-- spring-beans -->
50      <dependency>
51          <groupId>org.springframework</groupId>
52          <artifactId>spring-beans</artifactId>
53          <version>3.2.0.RELEASE</version>
54          <exclusions>
55              <exclusion>
56                  <artifactId>spring-core</artifactId>
57                  <groupId>org.springframework</groupId>
58              </exclusion>
59          </exclusions>
60      </dependency>
61
62
```

会发现，原本我们只需要在struts2-spring-plugin中排除依赖spring-core，结果同样的代码也出现在了psring-beans中，就不合理了，所以将54到59行代码删除，这样就搞定了

scope：依赖范围，意思就是通过pom.xml加载进来的jar包，来什么范围内使用生效，范围包括编译时，运行时，测试时

| 依赖范围 | 对于编译classpath有效 | 对于测试classpath有效 | 对于运行时classpath有效 | 例子 |
| --- | --- | --- | --- | --- |
| compile | Y | Y | Y | spring-core |
| test | - | Y | - | Junit |
| provided | Y | Y | - | servlet-api |
| runtime | - | Y | Y | JDBC驱动 |
| system | Y | Y | - | 本地的，Maven仓库之外的类库 |

compile：默认值，如果选择此值，表示编译、测试和运行都使用当前jar

test：表示只在测试时当前jar生效，在别的范围内就不能使用该jar包。例如：junit 。此处不写也不报错，因为默认是compile，compile包扩了测试

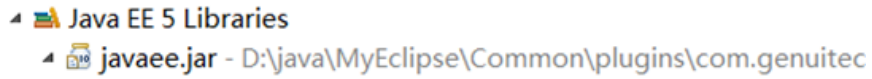
runtime，表示测试和运行时使用当前jar，编译时不用该jar包。例如：JDBC驱动。JDBC驱动，在编译时(也就是我们写代码的时候都是采用接口编程，压根就没使用到JDBC驱动包内任何东西，只有在运行时才用的到，所以这个是典型的使用runtime这个值的例子)，此处不写也不报错，理由同上

provided，表示编译和测试时使用当前jar，运行时不在使用该jar了。例如：servlet-api、jsp-api等。【必须填写】

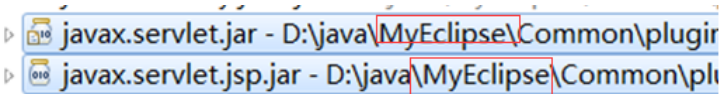
什么意思呢？ 在我们以前创建web工程，编写servlet或者jsp时，就没导入过jar包把，因为myeclipse或者别的ide帮我们提供了这两个jar包，内置了，所以我们在编译期测试期使用servlet都不会报缺少jar包的错误，而在运行时期，离开了myeclipse或别的ide，就相当于缺失了这两个jar包，但此时tomcat又会帮我们提供这两个jar，以便我们不会报错，所以，这两个很特殊。看图

1、开发阶段(MyEclipse提供)，看下图以此证明我们说的

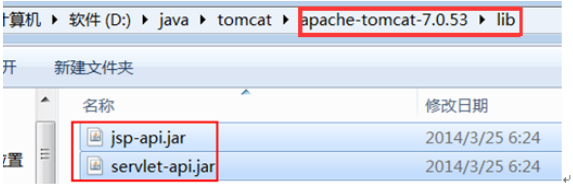
java web 5.0项目：




java web 6.0项目：




2、运行阶段(tomcat提供)




所以，根据这个特点，如果使用maven开发项目，就不是web项目了，那么myeclipse就不会在给我们提供这两个jar包，我们就必须自己手动通过坐标从仓库中获取，但是针对上面的分析，当运行的时候，tomcat会帮我们提供这两个jar包，所以我们自己从

仓库中获取的jar包就不能和tomcat中的冲突，那么就正好可以通过provided这个属性，来设置这两个jar的作用范围，就是在变异时期和测试时期生效即可。

这个例子就可以解释上面创建maven web时产生的错误和解决方案了。

system:表示我们自己手动加入的jar包，不属于maven仓库(本地，第三方等)，属于别得类库的这样的jar包，只在编译和测试期生效，运行时无效。一般不用

## 7.3、依赖调节原则

这个就是maven解决传递依赖时jar包冲突问题的方法，按照两种原则，上面已经介绍了一种了，就是下面的第二原则

1、第一原则：路径近者优先原则

A-->B-->C-->D-->X(1.6)

E-->D-->X(2.0)

使用X(2.0)，因为其路径更近

2、第二原则：第一声明者优先原则。就是如果路径相同，maven 默认配置在前面的优先使用

A-->B --> X(1.6)

C-->D--> X(2.0)

这样就是路径相同，那么如果A在前面，C在后面，则使用X(1.6)

maven会先根据第一原则进行选择，第一原则不成，则按第二原则处理。

# 八、总结

这篇文章的篇幅有点长，也消耗了我挺多的时间的，因为其中遇到一些bug，一直找不出原因，一度想放弃，但还是坚持了下来，这也只是maven的入门，知道大概怎么用，看别的应该就看得懂，其实项目中真正用的还是下一节所要讲解的。

如何搭建私服?

如何从私服中获取jar包

使用maven对父工程与子模块的拆分和聚合。

下一节就讲这些东西把，加油。