Maven的使用

1. **安装maven**

**步骤1：**下载maven

**地址：**<http://maven.apache.org/docs/3.1.1/release-notes.html>

**步骤2：**配置环境变量

配置环境变量%MAVEN\_HOME%\bin 即可完成安装，（MAVEN\_HOME为解压包位置）。

配置好后，可以在dos下用命令进行检查是否安装成功 mvn –version

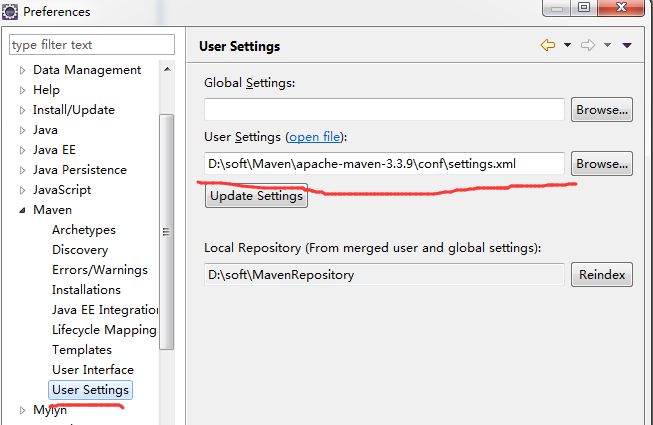


**步骤3：**修改仓库位置（仓库是用来放置工程所依赖的jar包或者第三方开源库的）

找到maven安装目录下的D:\apache-maven-3.0.5\conf中的setting.xml文件，修改仓库目录，如果maven安装目录下没有存在目录Repositories，那我们就新建一个空的Repositories文件夹



然后将settings.xml文件复制一份儿到自己的仓库文件目录下。最后设置eclipse中的maven参数

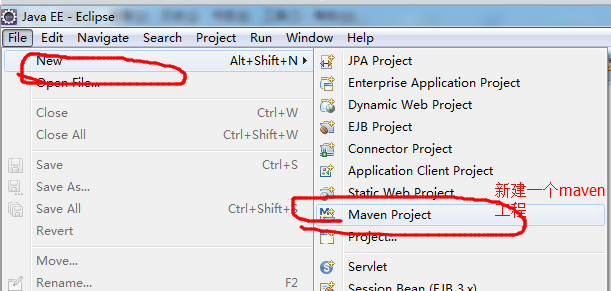


步骤4：在Eclipse中配置maven

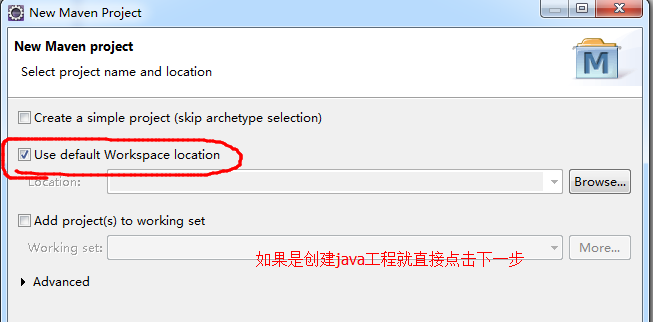


1. **创建java项目**

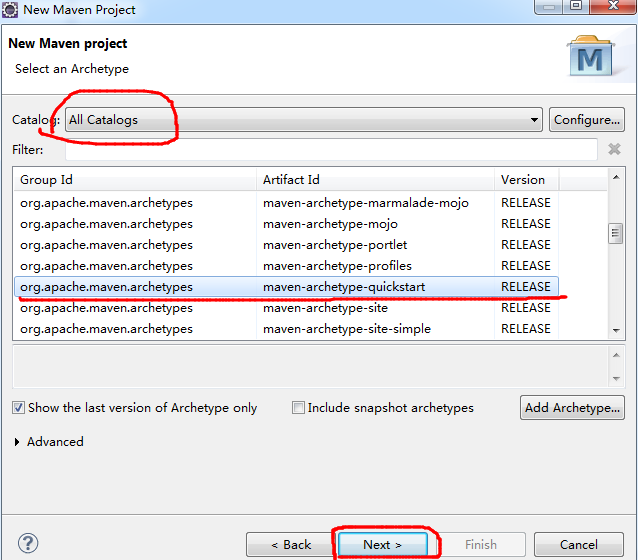
步骤1：



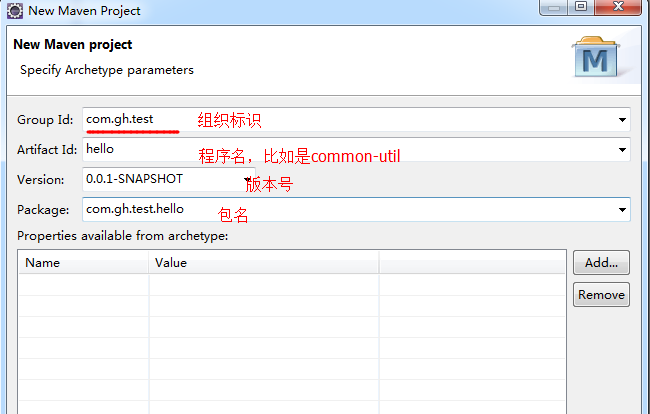
步骤2：



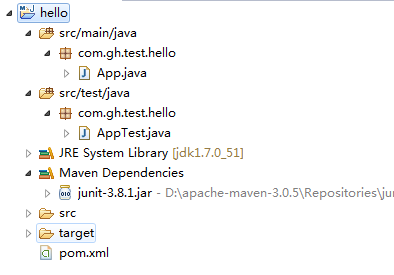
步骤3：



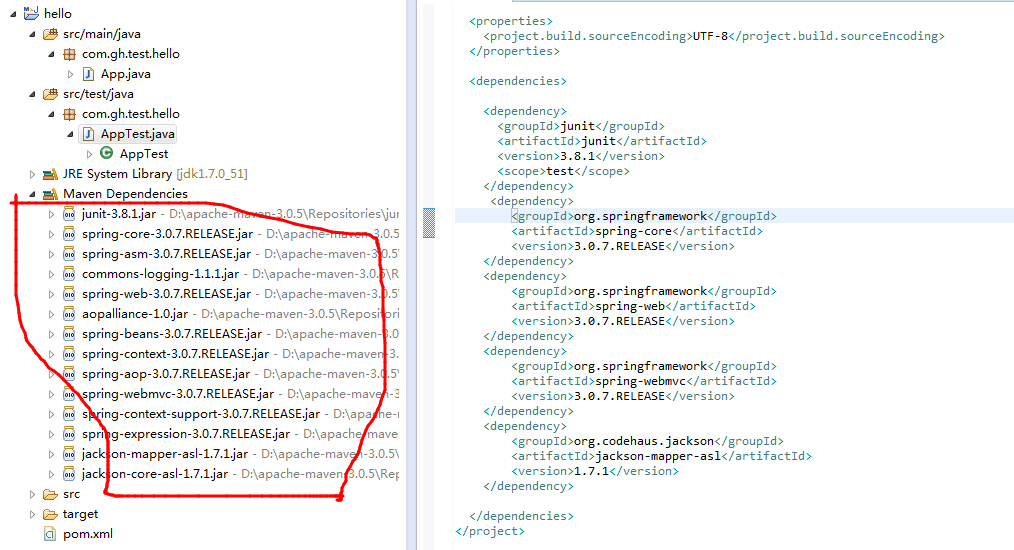
步骤4：



步骤4完了后，工程目录如下：



步骤5：在pom.xml文件中配置工程依赖的jar文件或者一些其他信息，配置完了保存该文件，maven会自动去下载所依赖的jar包。（注：如果依赖的jar文件在maven的仓库目录下已经存在，那么maven会直接引入，而不会再去下载）



1. **新建web工程**

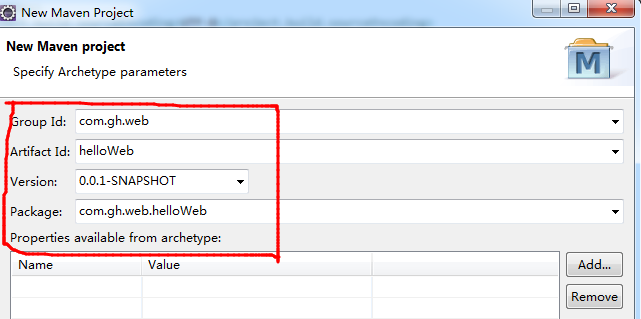
**第一种方法：**

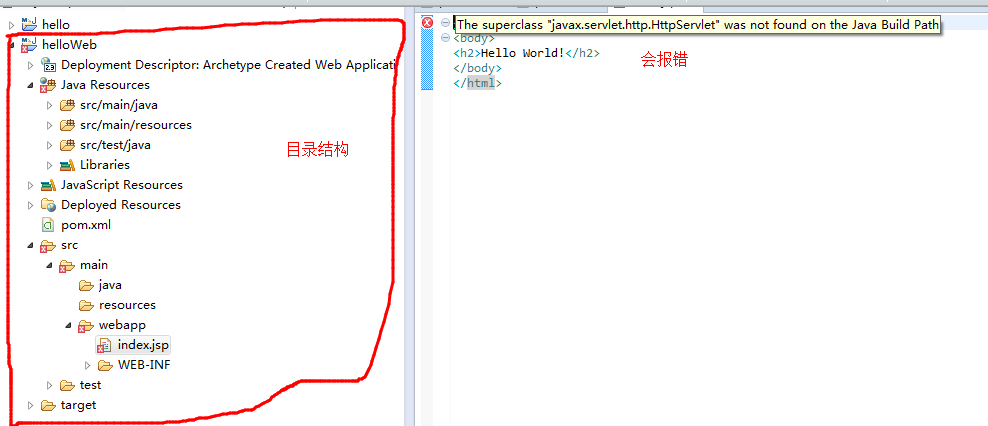
步骤1和2与创建java工程一样

步骤3：



步骤4：

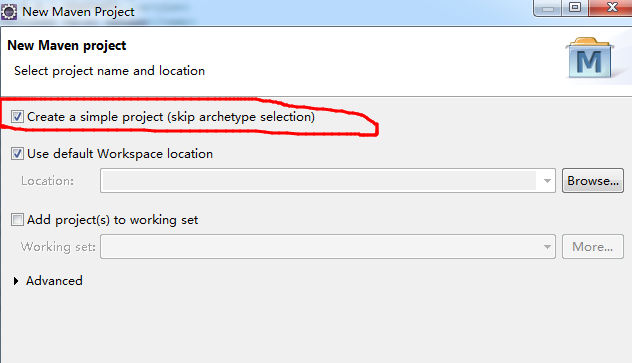




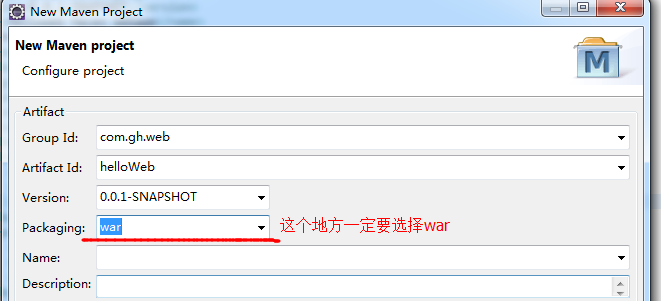
**第二种方法：**

步骤1与创建java工程一样

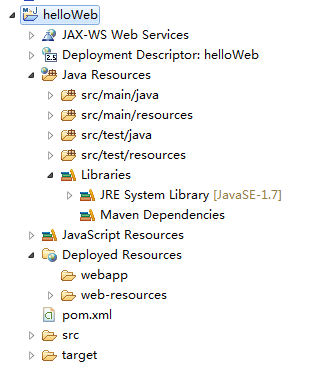
步骤2：



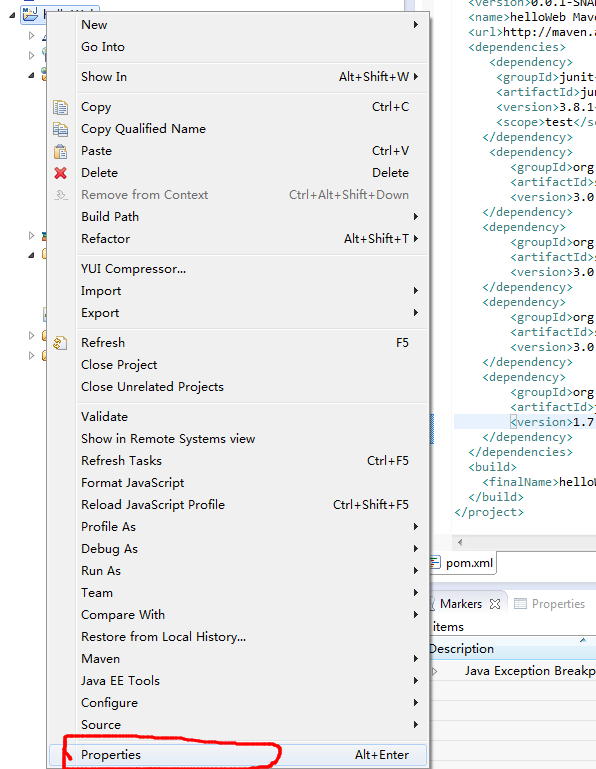
步骤3：



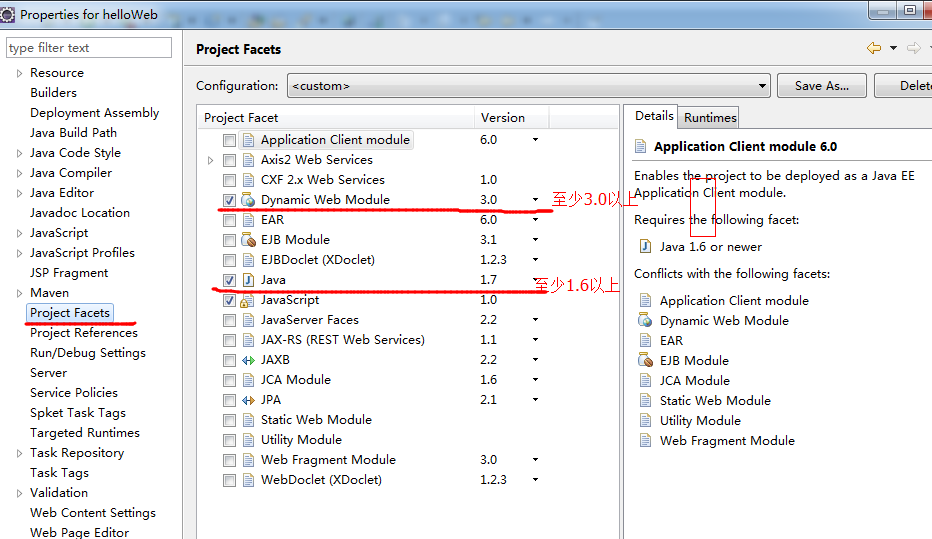
工程目录：



以下是第一种方法和第二种方法都需要的操作：



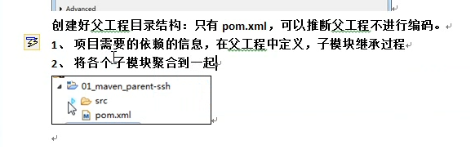


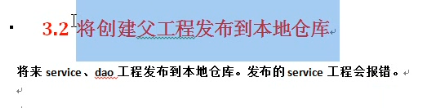


这样就可以在tomcat下运行了

1. **Maven创建多模块的工程**

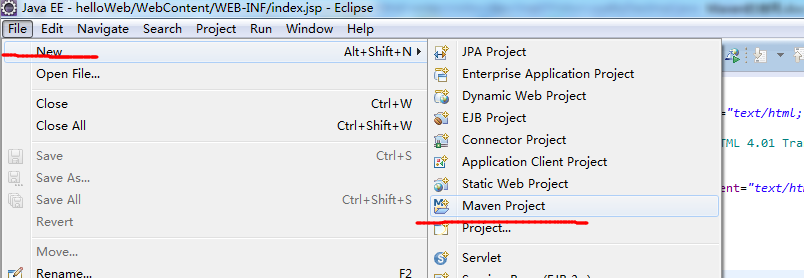
用maven创建一个包含多个子工程的工程

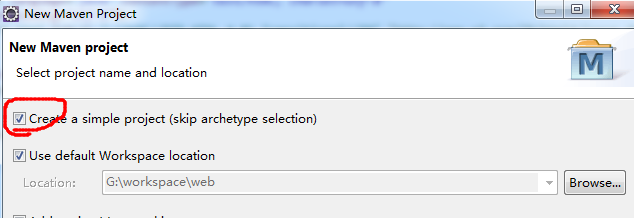


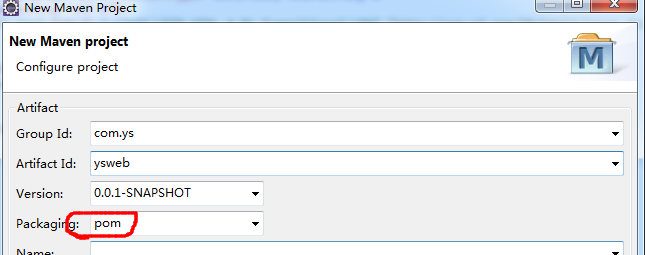


也就是maven install

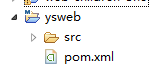
步骤1：创建父工程



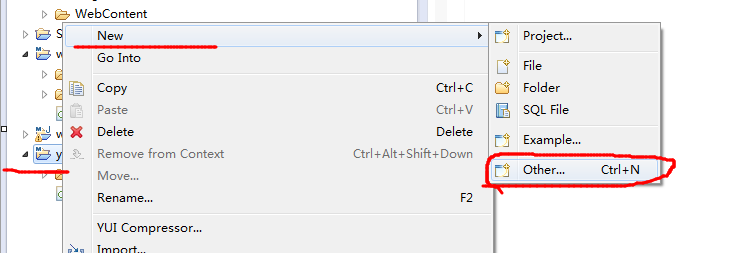


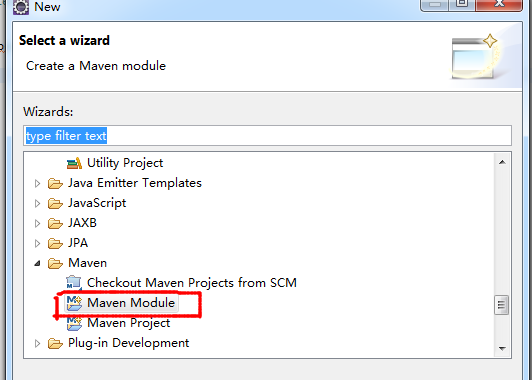


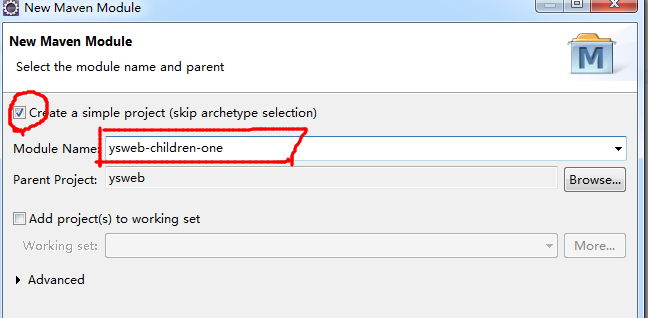
工程结构如下：

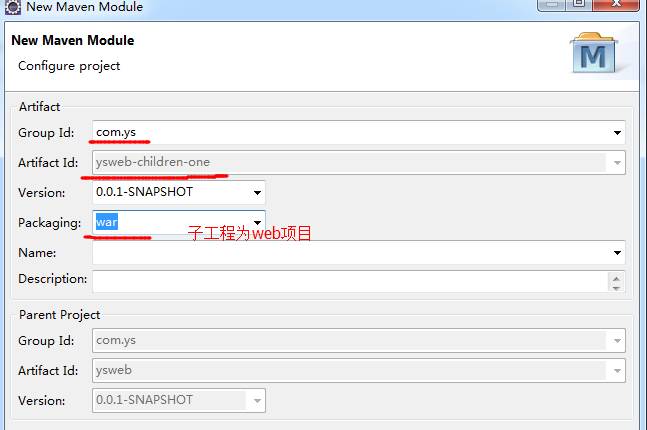


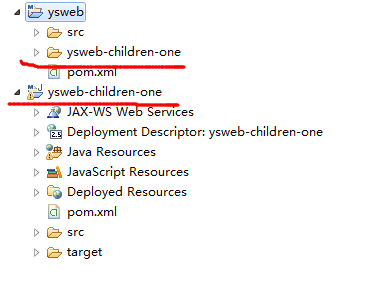
创建子工程：





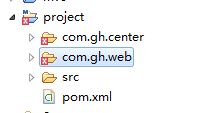






1. **配置Web项目依赖另外一个项目**

**目录结构：**



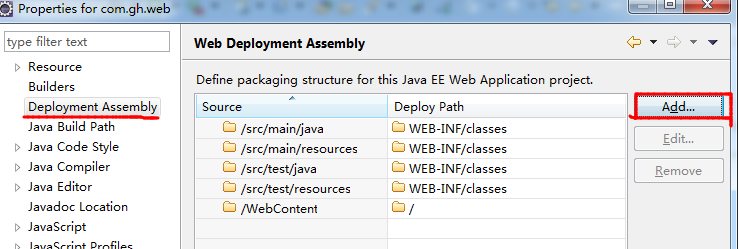
**问题描述：**如上所示，如果com.gh.web项目的运行需要依赖com.gh.center项目，这个maven需要如何配置呢？

**解决方法：**

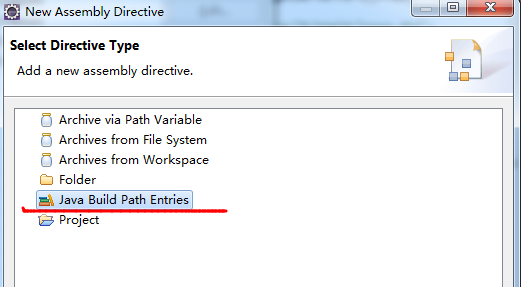
步骤1：选中com.gh.web项目，点击右键，选择属性。

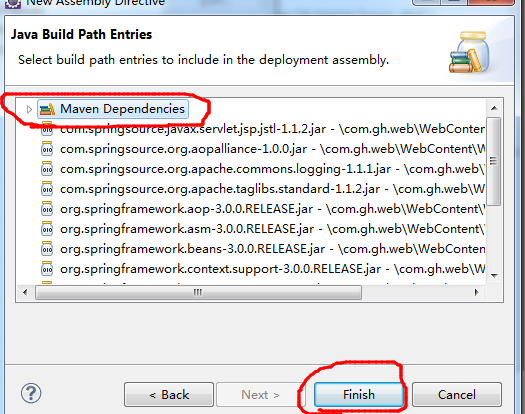


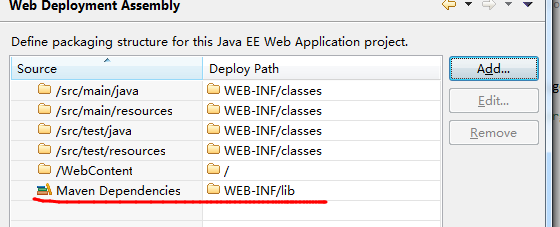
步骤2：



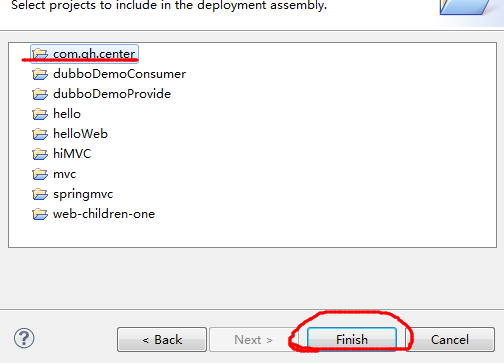
步骤3：添加maven依赖

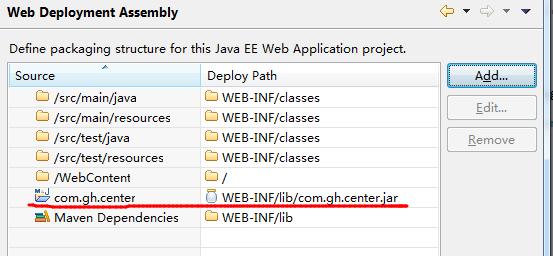




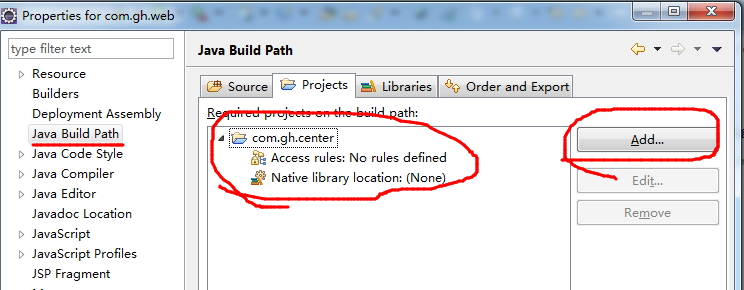


步骤4：添加所要依赖的项目com.gh.center



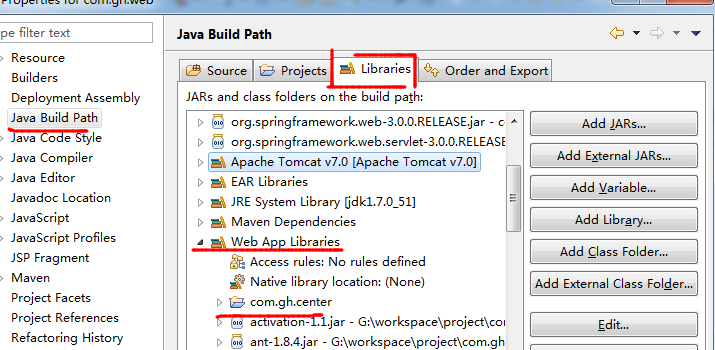


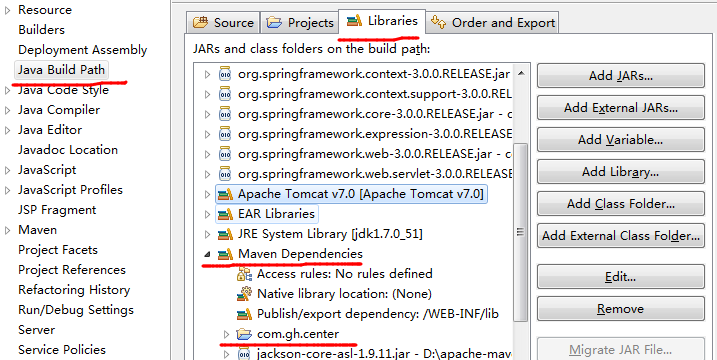
步骤5：查看build path选项，看看com.gh.center依赖是否也已经配置，如果配置好了，那么就不需要这步，如果没有则需要。



步骤6：记得要把com.gh.center项目打成jar包放到com.gh.web的lib目录下，并且add to build path到jar依赖目录下。

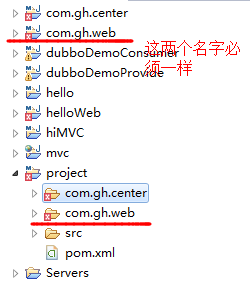
最后效果如下：





1. **如果maven配置依赖都是对的，但是依赖的com.gh.center包里面的东西在com.gh.web中还是没法用。**

**解决方法：**这个问题可能是由于我们修改了com.gh.web项目的名字，这个名字不能乱修改，必须要与父目录下的名字相同。



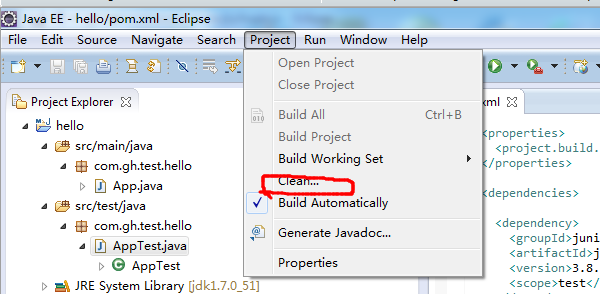
1. **Pom.xml文件配置好后，如果maven下载不了，会报错**

**错误信息：**

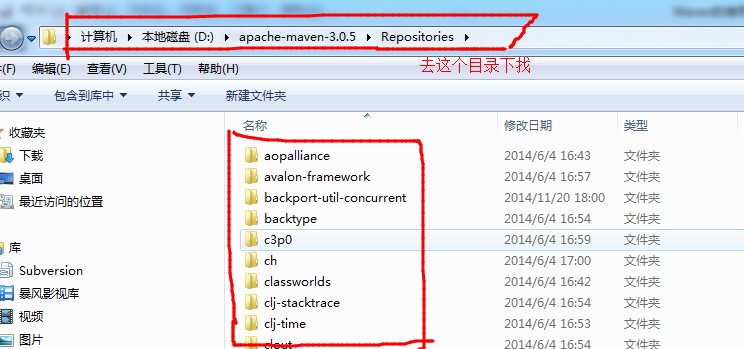
ArtifactTransferException: Failure to transfer org.apache.hadoop:hadoop-core:jar:1.1.1 from <http://repo.maven.apache.org/maven2>

**原因：**对于这个包从maven中心传输到本地仓库失败，决定不会重新尝试下载jar包，直到mavne再改更新索引，或强制更新。

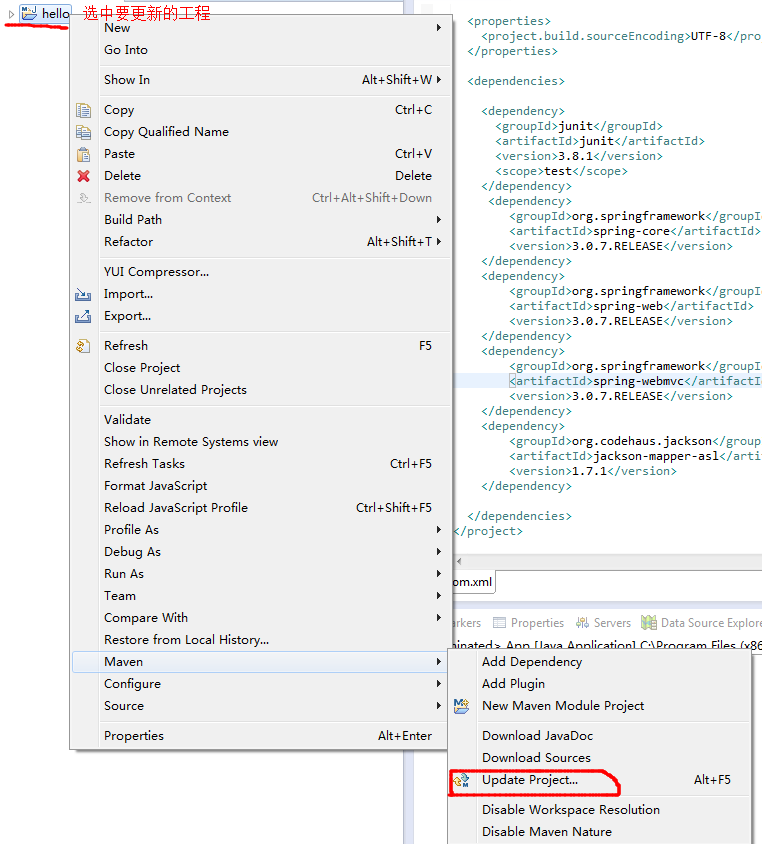
**解决方法**：我们首先可以先把工程进行clean



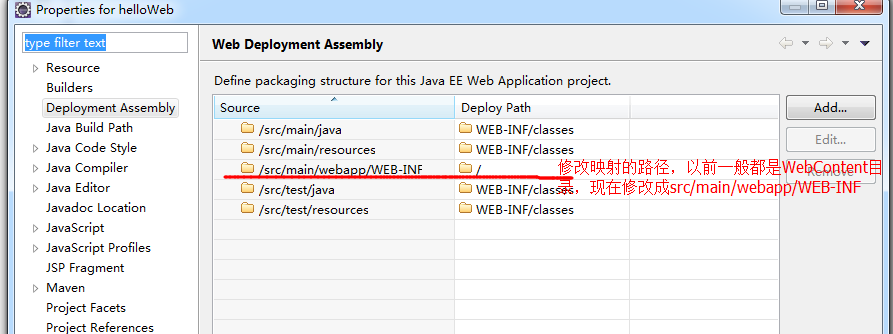
然后去本地仓库，把这个1.1.1的目录删除掉



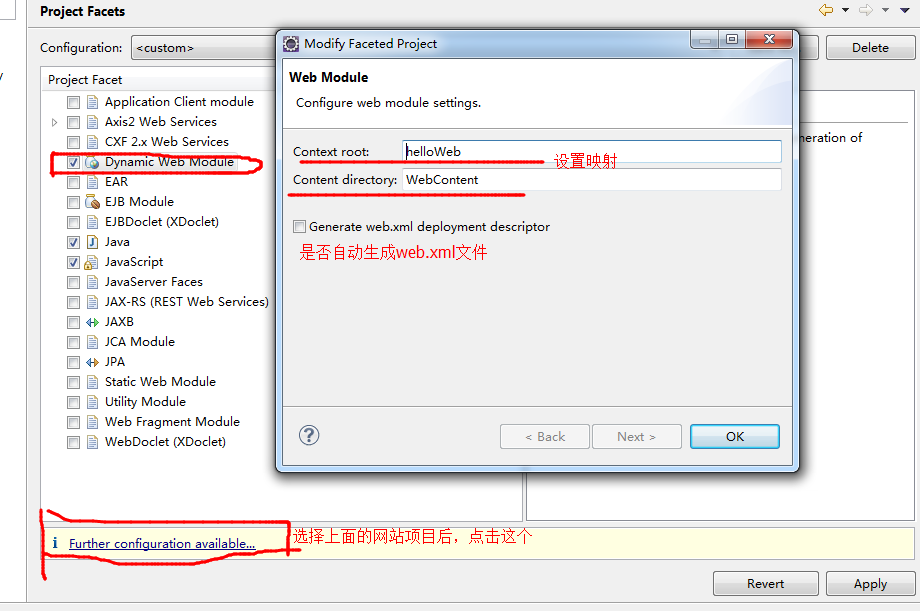
最后刷新你的项目



1. **修改WebContent的映射**



1. **配置网站映射**

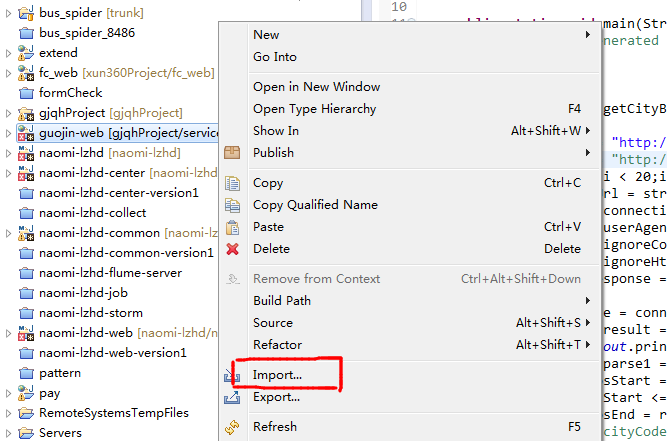


1. **SpringMVC+Maven+Dubbo项目实例**

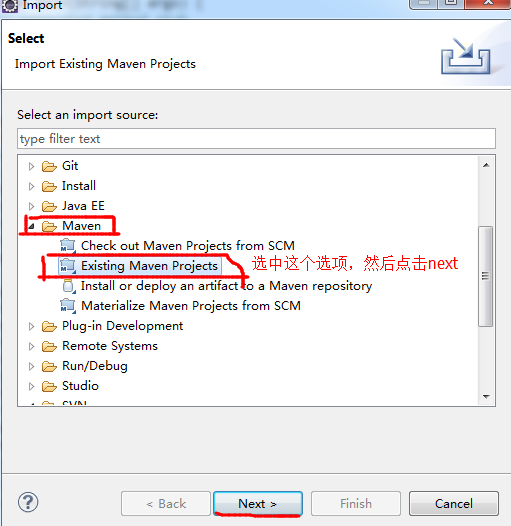
源码见：百度云盘-技术资料-web源码-SpringMVC+Maven+Dubbo项目

1. **Eclipse导入maven工程**

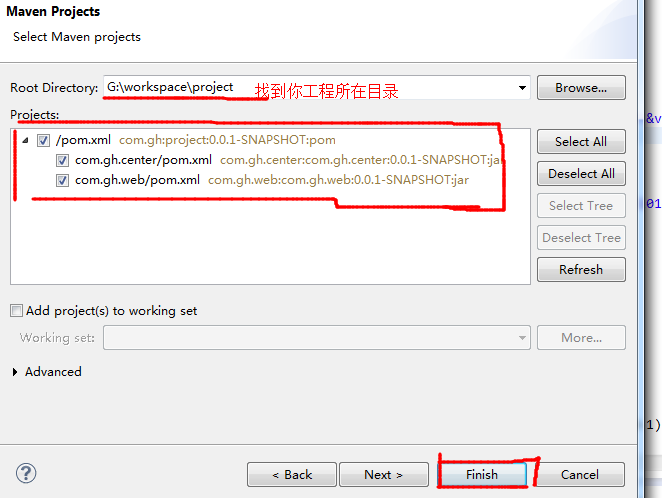
步骤1：



步骤2;



步骤3：



1. **使用maven命令把工程转化为maven工程**

**问题描述：**某些maven工程出现问题，在导入maven工程的时候出现各种问题，比如包显示不正确等等。

**解决方法：**可以使用maven命令，将工程重新转化为maven工程，这样再导入就不会有问题了。

使用终端进入工程目录所在位置，比如工程叫rose-example,则进入该目录，cd rose-example,然后在终端输入mvn eclipse:eclipse命令，将该工程转化为maven工程就行了，然后再次用eclipse导入工程，就不会出问题了。

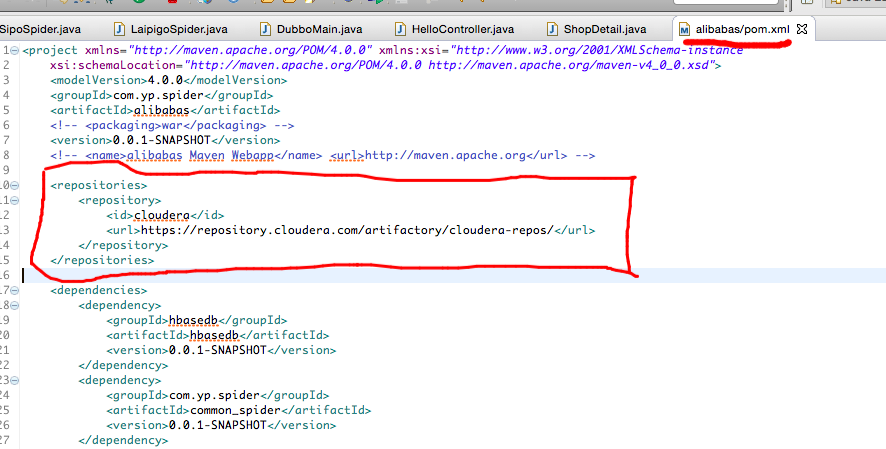
1. **Missing Artifact**

**问题描述：**我们在pom文件中配置jar后，出现Missing artifact commons-logging:commons-logging.jar错误。它的意思就是在库中找不到commons-logging.jar。

**原因：**可能是我们的网上的仓库中没有该jar包。

**解决方法：**maven在没有设置下载仓库的位置时，它会默认去maven的地址下载jar包，如果maven上么有该jar包就会报上面的错误，此时，我们可以从网上搜索包含该jar包的库，然后将该库配置到pom文件中，这样maven就会去我们配置的库中下载jar包文件。

我们工程的pom文件：



<repositories>

<repository>

<id>cloudera</id>

<url>http://10.151.30.26:8081/nexus/content/groups/public/</url>

</repository>

</repositories>

1. **Tomcat无法解析JSP文件**

**问题描述：**tomcat在请求JSP页面时报org.apache.jasper.JasperException: Unable to compile class for JSP:错误

**原因：**可能是自己的工程有依赖servlet-api.jar,jsp-api.jar，因为tomcat自己有该jar包，如果自己再添加，tomcat就会报错。

**解决方法：**将自己依赖的这两个jar包去掉。如果该jar不是独立的，而是别的jar包依赖的，比如paoding-rose框架就依赖servlet-api.jar，这时我们又是maven工程，我们只需要在pom文件中配置paoding-rose依赖的时候，不要依赖servlet-api.jar。配置如下：

<dependency>

<groupId>com.54chen</groupId>

<artifactId>paoding-rose</artifactId>

<version>1.0</version>

<exclusions>

<exclusion>

<groupId>javax.servlet</groupId>

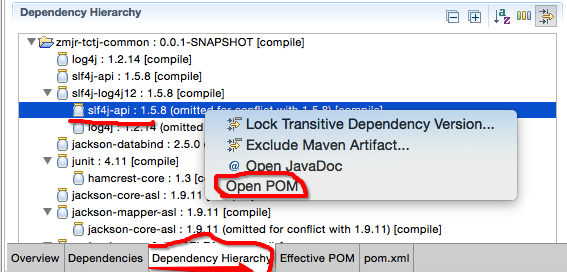
<artifactId>servlet-api</artifactId>

</exclusion>

</exclusions>

</dependency>

注：如果我们不知道servlet-api.jar的groupid和artifactId，可以在pom文件中查看。选中该jar，右键选中open pom就可以看到这两个值



1. **在maven中配置生成项目文档**

**描述：**有的时候，我们写的程序需要自动生成程序文档，这样很方便。使用maven必须在pom文件里面进行配置。

具体操作，可以使用maven命令行生成文档，首先进入项目的根目录下，比如我的项目名称叫tcxy-common，那么用命令行进入该项目的目录下，然后输入mvn site 命令 进行生成文档。不过建议使用mvn clean site命令是最佳的，这样每次生成的文档可以保证是最新的

<build>

<pluginManagement>

<plugins>

<!-- 构建项目站点报告插件 -->

<plugin>

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

<artifactId>maven-site-plugin</artifactId>

<version>3.3</version>

<configuration>

<!-- 配置报告信息 -->

<reportPlugins>

<!-- 项目基本信息报告 -->

<plugin>

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

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>2.2</version>

<configuration>

<dependencyDetailsEnabled>true</dependencyDetailsEnabled>

<dependencyLocationsEnabled>false</dependencyLocationsEnabled>

</configuration>

</plugin>

<!-- 项目API doc报告 -->

<plugin>

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

<artifactId>maven-javadoc-plugin</artifactId>

<version>2.7</version>

</plugin>

<!-- 项目还需要做的TODO报告 -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>taglist-maven-plugin</artifactId>

</plugin>

<!-- 生成站点文件具体信息报告 -->

<plugin>

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

<artifactId>maven-linkcheck-plugin</artifactId>

<version>1.1</version>

<configuration>

</configuration>

</plugin>

<!-- 单元测试覆盖率报告 -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>cobertura-maven-plugin</artifactId>

</plugin>

<!-- 项目源代码分析报告 -->

<plugin>

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

<artifactId>maven-pmd-plugin</artifactId>

<version>2.5</version>

<configuration>

<linkXref>true</linkXref>

<sourceEncoding>GBK</sourceEncoding>

<minimumTokens>100</minimumTokens>

<targetJdk>1.5</targetJdk>

</configuration>

</plugin>

</reportPlugins>

<!-- 配置站点国际化 -->

<locales>zh\_CN</locales>

<!-- 输出编码 -->

<outputEncoding>GBK</outputEncoding>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

1. **无法在终端运行mvn命令**

**问题描述：**无法在终端运行mvn命令，出现如下报错：

Error: JAVA\_HOME is not defined correctly.

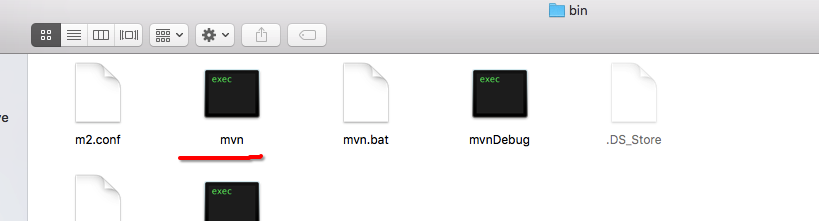
**解决方法：**进入maven安装目录的bin文件夹下，找到mvn执行脚本，修改此脚本。将这句

export JAVA\_HOME="`/usr/libexec/java\_home`"

替换掉原来的

export JAVA\_HOME=/usr/libexec/java\_home

这句





1. **mvn download出错**

**问题描述：**设置mvn远程仓库有三个地方可以设置，第一是在每个工程的pom文件里面，这个的作用域仅限于该工程，还有就是在mvn目录下的conf文件夹下的settings文件中可以设置，第三是在默认的仓库目录下设置settings文件（.m2/目录）。当我们在update project时候或者 mvn clean site的时候，会报如下错误：

****

这个地方就是mvn设置的下载仓库地址，如果不对，则需要修改。

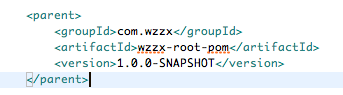
**注：**每次我们修改仓库地址的时候，最好统一修改，包括mvn目录下的conf中的settings文件以及.m2下的settings文件，都必须修改成一致的，不然就容易出现上述的错误。

建议mvn目录下的conf中的settings文件以及.m2下的settings文件都用mvn默认的，不要去修改，如果在某工程下要用特殊的仓库地址，则在每个工程的pom文件中设置，具体设置见问题13

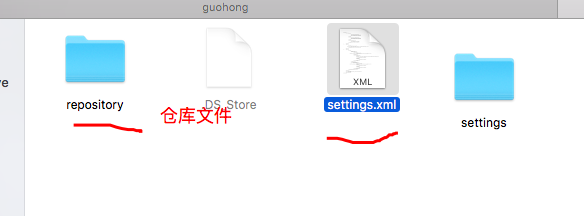
1. **将maven工程发布到指定私服**

**步骤1：**配置工程的pom文件。这里由于是采用易宝的私服地址，他在工程wzzx-root-pom里面配置了发布地址，所以我们只需要引用该工程为父工程就行。

如果需要改其他发布地址或者发布参数的话，也可以直接在工程pom.xml文件里面配置。



**步骤2：**设置maven的settings.xml文件，这个文件一定要跟仓库文件放在同级目录

****

****

**步骤3：**进入到工程目录下，比如我的工程名叫做abc,则在终端进入abc目录，然后输入maven命令。

mvn clean compile 这个命令是先清理工程，重新编译工程

mvn install 这个命令是将工程发布到本地的maven库

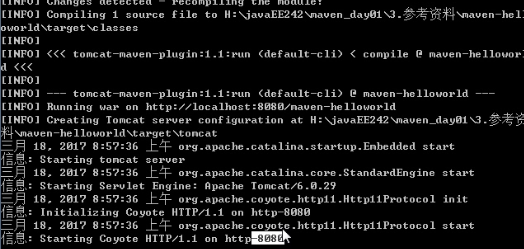
mvn deploy 这个命令是将工程发布到私服

1. **Maven常用命令**

mvn clean：清理编译好的工程文件，clean后工程根目录下的target文件会被干掉，这里面是编译后的class文件。

mvn compile:编译Java工程，在工程根目录下的生成target文件，执行编译并且将编译后的class文件放入该目录，跟mvn clean动作刚好相反。

mvn tomcat:run :将工程发布到Tomcat并且运行



mvn test :执行单元测试，步骤：编译—执行所有的单元测试，执行test的时候就会自己去编译，因此无需单独执行编译命令

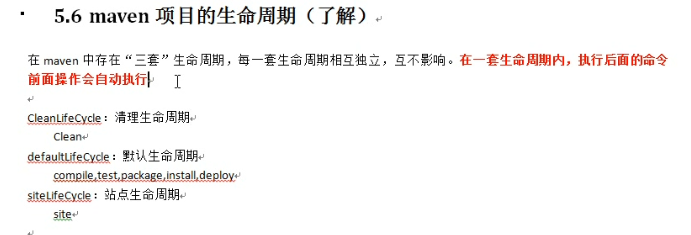


mvn package :对Java工程进行打包，web项目会打成war包，Java工程会打成jar包，软件会根据具体工程来进行打包，无需用户设置，打包地址会放入target目录下。执行打包的时候，maven会自己先执行编译。



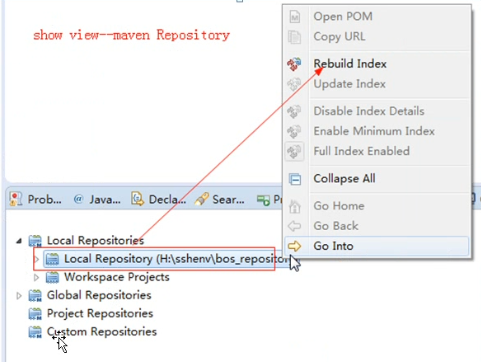
mvn install：将自己的工程加入到本地仓库，这样其他引用这个工程的项目就能配置依赖进行使用了。此命令也会先进行编译，并且会执行test,还会进行打包。

1. **Maven项目的生命周期**

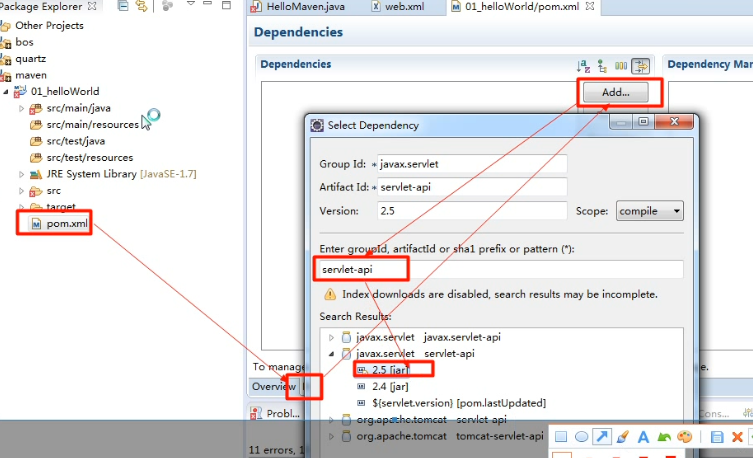


1. **构建maven索引**

有可能maven本地仓库有10个G的jar包，如何让maven快速的找到依赖的jar包，可以构建索引。



1. **查找依赖**



1. **Pom.xml添加jdk编译插件**

<build>

<plugins>

<plugin>

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

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

<configuration>

<source>1.7</source>

<target>1.7</target>

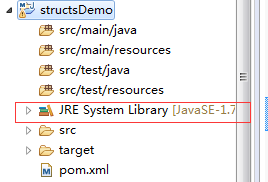
<encoding>UTF-8</encoding>

</configuration>

</plugin>

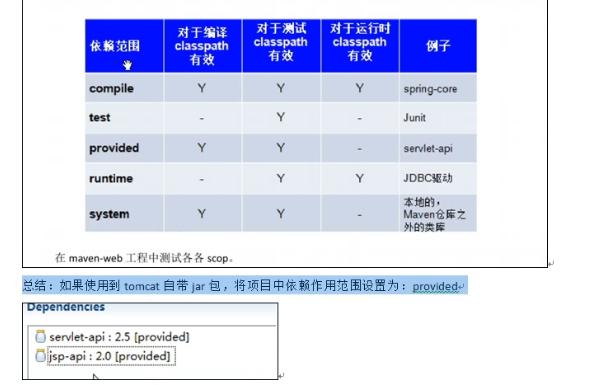
</plugins>

</build>



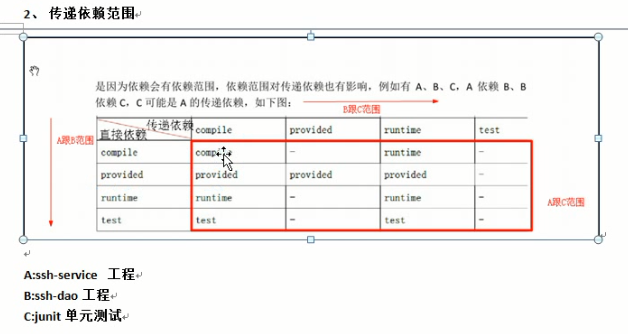
1.7指Java的版本，UTF-8指字符编码，这里是配置maven的jdk编译版本以及字符编码

1. **依赖范围scope**



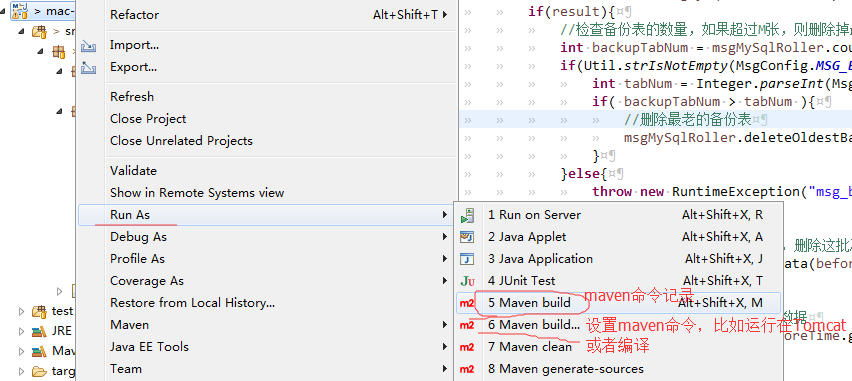
如果依赖范围是test，则运行打包时，就不会有这个jar依赖

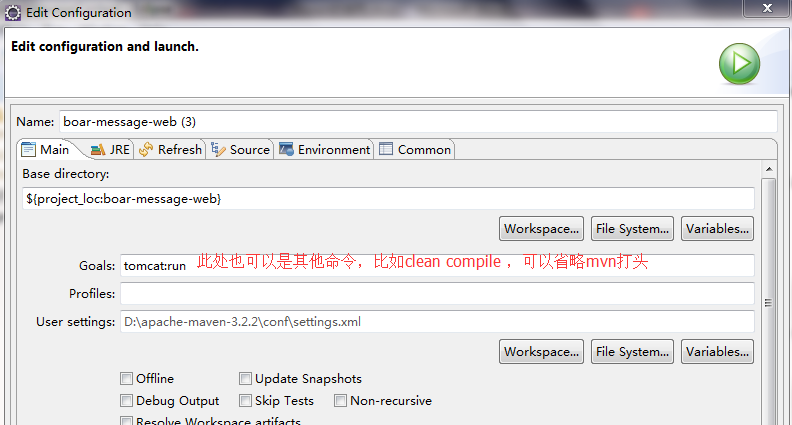
下面是传递依赖的范围



1. **Eclipse中工程配置使用maven命令**

**示例1：**使用maven命令将项目发布到Tomcat并运行，此处是run as，debug as会有所区别

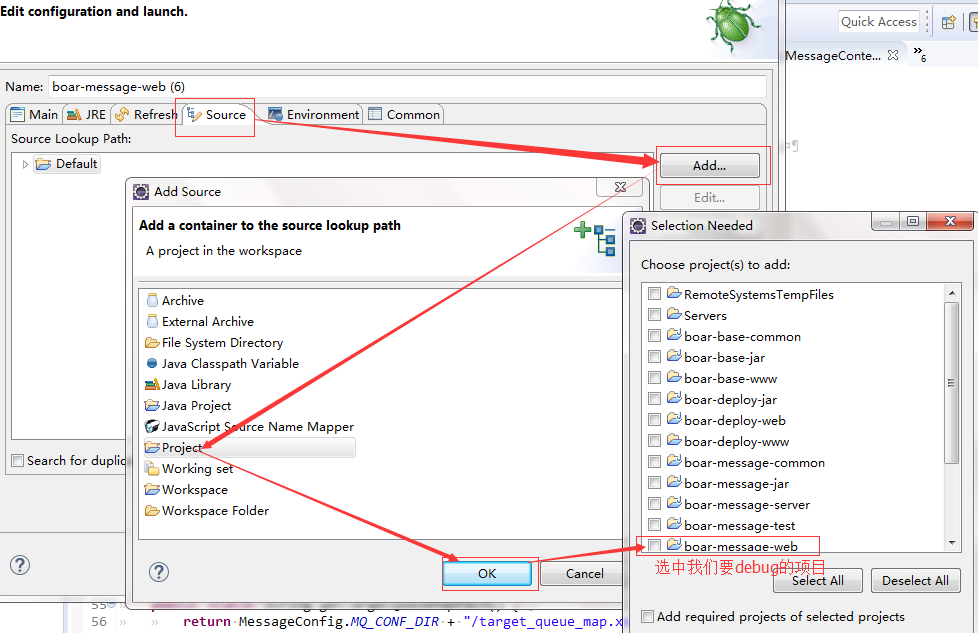




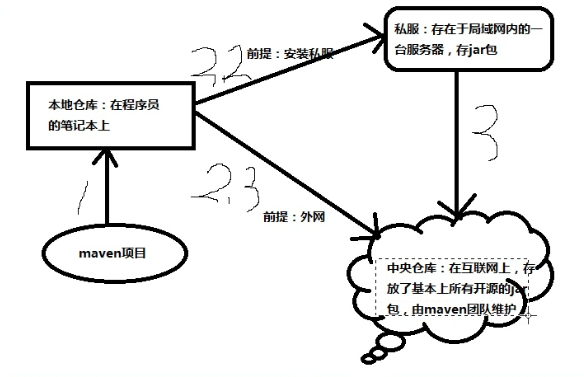
效果图：过程：compile-package-发布-运行



Debug as 会有不同，比run as多一步，添加source



1. **关于Maven仓库**



1. **传递依赖冲突解决**



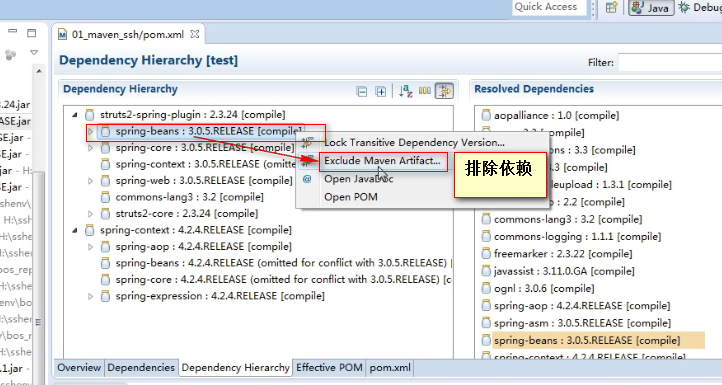
这时C版本就会产生冲突。

Maven有自己的原则去解决冲突：

第一：声明者优先原则，在同一个pom文件中，谁先定义，则用谁的版本。

第二：路径近者优先原则,直接依赖级别高于传递依赖。有直接依赖配置时，优先使用直接依赖的版本。

Maven也可以自己用配置文件来控制依赖：排除依赖





版本锁定：推荐使用

指定项目中依赖的版本,(版本锁定并不会将依赖包加入工程，他只会起指定版本的作用，依赖包还得另外配置)



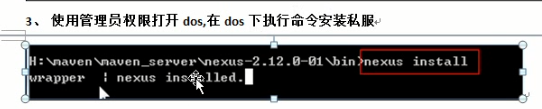
1. **私服安装**

**步骤1：下载安装包**

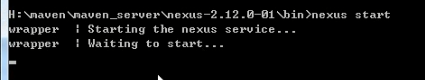


**步骤2：解压压缩包**

**步骤3：安装私服**



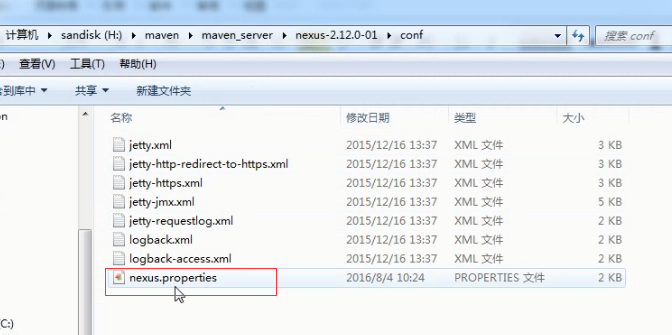
**步骤4：启动服务**



也可以去Windows的服务里面去启动，方法跟启动mysql服务一样

**步骤5：找到私服的访问地址，地址可以去安装目录下的conf中的nexus.properties文件中查找，密码和账户是默认的**



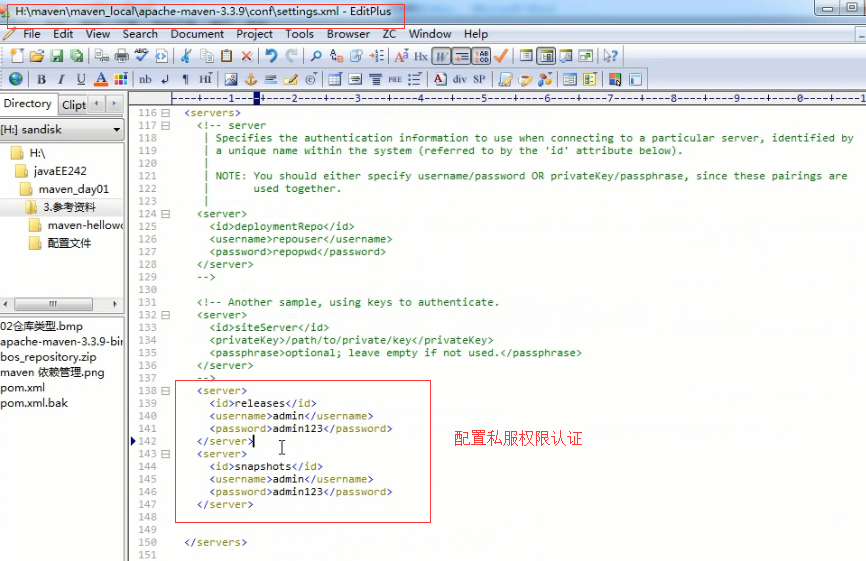


1. **私服仓库类型**



**注：推荐用组仓库**

1. **上传jar包到私服**

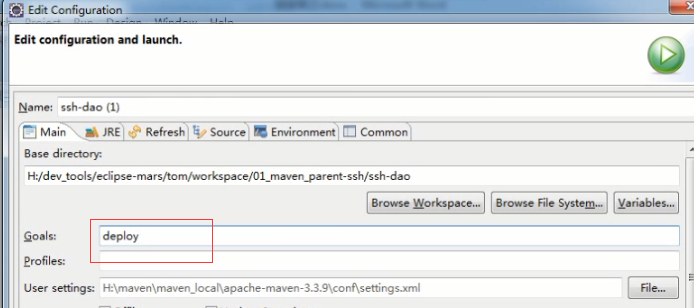


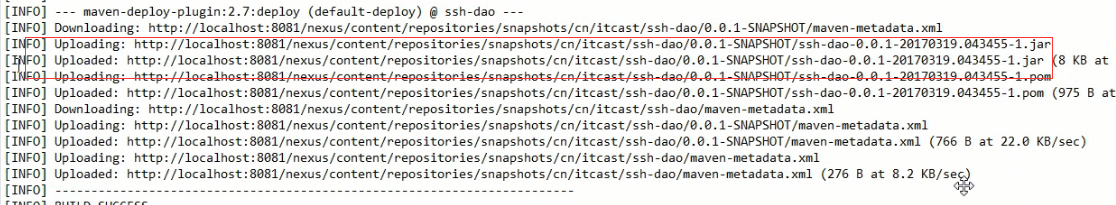
然后配置jar上传的路径



执行命令将项目发布到私服

deploy



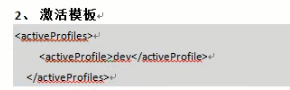


1. **从私服地址下载jar包到本地仓库**

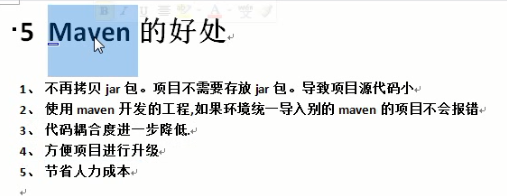
在maven的settings.xml配置文件中进行配置（模板）



然后在setting.xml文件中配置激活模板，其实可以不用配置



1. **maven的好处**

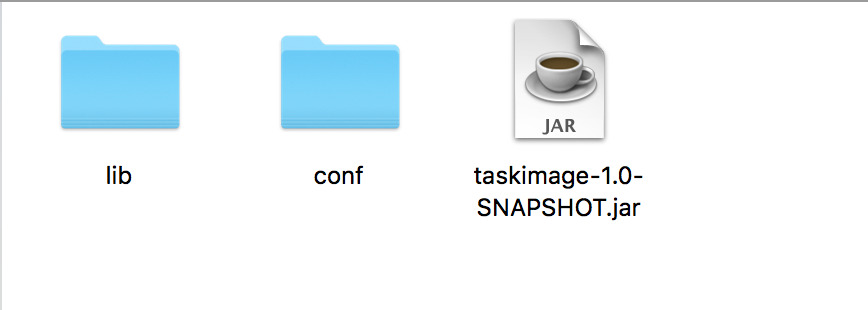


1. **Maven打包可运行的jar**

Maven打包可运行的jar包，但是不打入lib依赖包和配置文件，在pom文件中添加如下信息

|  |
| --- |
| <**build**>  <**plugins**>  <**plugin**>  *<!--指定打包编译的Java版本-->* <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-compiler-plugin</**artifactId**>  <**configuration**>  <**source**>1.8</**source**>  <**target**>1.8</**target**>  </**configuration**>  </**plugin**>  <**plugin**>  <**groupId**>org.apache.maven.plugins</**groupId**>  <**artifactId**>maven-jar-plugin</**artifactId**>  <**configuration**>  *<!--将配置文件不打入jar包中，配置文件需要自己手动放到jar包的同级目录resources下，这样方便修改-->  <!-- 不打包资源文件（配置文件和依赖包分开） -->* <**excludes**>  <**exclude**>\*.properties</**exclude**>  </**excludes**>  <**archive**>  <**manifest**>  <**addClasspath**>true</**addClasspath**>  <**useUniqueVersions**>false</**useUniqueVersions**>  <**classpathPrefix**>lib/</**classpathPrefix**>  *<!--添加主启动类-->* <**mainClass**>com.ruima.task.MainClass</**mainClass**>  </**manifest**>  <**manifestEntries**>  *<!--MANIFEST.MF 中 Class-Path 加入资源文件目录 -->* <**Class-Path**>./conf/</**Class-Path**>  </**manifestEntries**>  </**archive**>  </**configuration**>  </**plugin**>   </**plugins**> </**build**> |

打包完成后，将lib和配置文件信息放到同一级目录下



直接运行jar就行