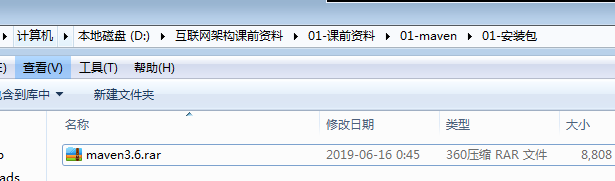
# 1.maven

## 1.1.maven是什么

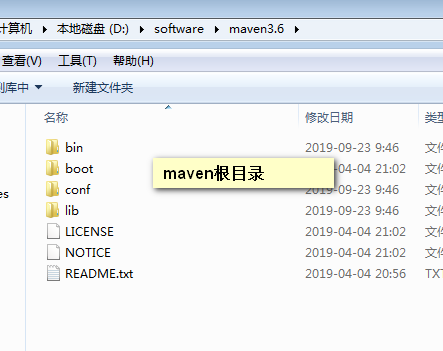
基于项目对象模型（POM）概念，利用中央信息片段管理一个项目的构建，生成，报告等步骤，是一个项目管理工具。

## 1.2.maven的安装和配置（windows）

### 1.2.1.解压maven

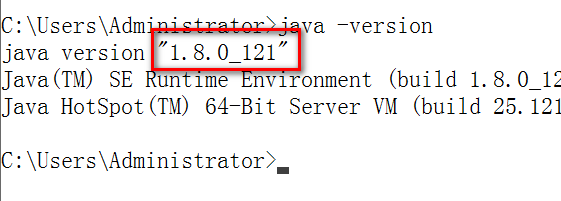


将maven3.6.rar解压到没有中文、没有空格的目录（D:/software

）

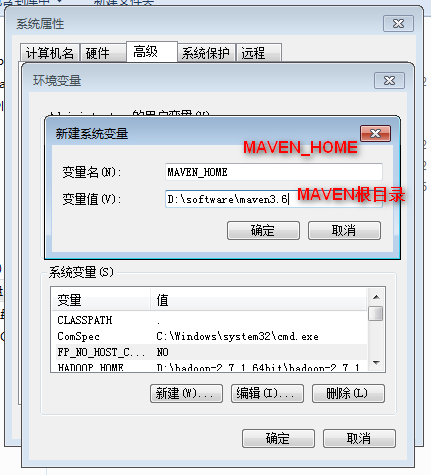
### 1.2.2配置windows环境

Jdk1.8的支持

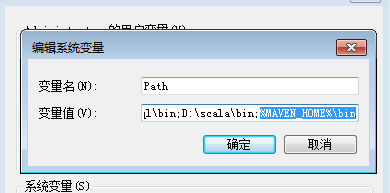


配置MAVEN\_HOME的环境变量

我的电脑右键->属性->高级系统设置->环境变量

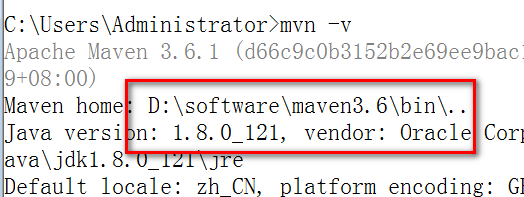


Path变量指向maven根目录的bin（在后面加上%MAVEN\_HOME%\bin）



检测环境变量配置

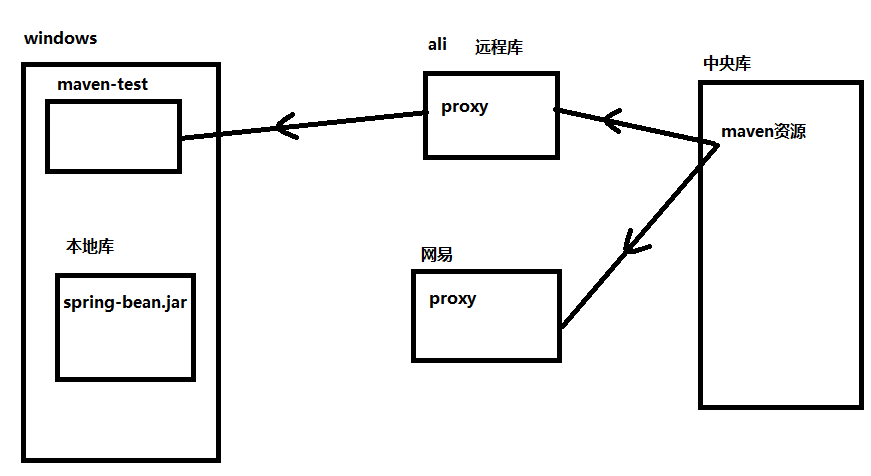
cmd中输入mvn -v



## 1.3.maven的一些概念

### 1.3.1.库

maven运行过程中，所有的命令执行都是基于插件完成的，引入依赖等内容，这些都称为maven的资源，资源的来源是maven社区中央库，在使用过程中可以经过远程库代理，获取资源存储在本地库。

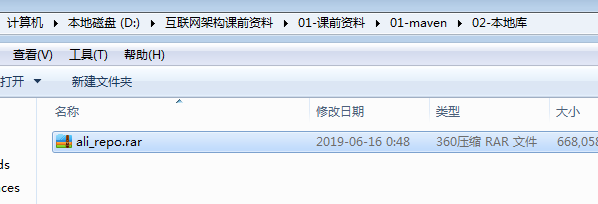


**中央库**：maven社区中心服务器，所有的maven资源都存储在中央库服务器中

**远程库**（私服）：除了中央库以外，中国有很多代理的远程库（镜像），例如阿里（速度最快的国内镜像代理）、网易；每个公司也可以有自己的远程库，供公司人员使用

**本地库**：所有经过代理，经过中央库使用的资源，都会下载到本地库使用

**调整配置本地库**，将ali\_repo.rar解压到没有中文空格目录下（d:/software）



在maven的根目录中，conf/settings.xml 55行修改本地库目录

<localRepository>D:/software/ali\_repo</localRepository>

注意：目录分隔符改为/

### 1.3.2.库中的资源

定位坐标：

对于库中的资源，根据中央信息片段定位，主要用到三个内容

<groupId>一般是域名的到写，代表大型项目，例如org.springframework

<artifactId>表示项目中的不同模块，例如spring-beans,spring-aop等

<version>表示模块版本号，例如4.3.7.RELEASE

通过这些坐标，可以在本地库寻找资源

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>4.3.7.RELEASE</version>

<groupId>对应本地库的目录org/springframework

<artifactId>对应本地库目录org/springframework/spring-beans

<version>对应本地库目录org/springframework/spring-beans/4.3.7.RELEASE

**资源内容**

\_remote.repositories：当前文件夹中所有资源来源（aliyun）

Jar包：打包的class文件

Source：源码资源

Javadoc:源码中抽取的文档

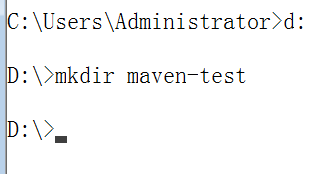
.sha1：判断下载资源是否正确的加密文件

### 1.3.3.maven生命周期

对于一个maven项目，从项目的创建，到项目的最终使用（发布到服务器，运行）

使用maven命令创建一个maven工程

准备一个管理测试maven工程的文件夹，意义和eclipse的workspace一致



在创建的文件夹中执行创建maven的第一个工程命令：mvn archetype:generate



下面执行选择题

Archetype：maven工程的骨架，maven作为管理工具，可以创建不同语言，不同结构的项目，maven准备了不同结构的骨架，在互联网架构阶段，使用两个骨架，webapps、quickstart

groupId：cn.tedu

artifactId:maven-test01

Version:保持默认1.0-SNAPSHOT

package:工程中默认的包名，默认和groupId保持一致

Y:是否同意上面的配置，Y同意



项目中可以包含的包结构

1.src/main/java:管理源码编写的文件夹，下一级就是包名+类

2.src/test/java:管理测试代码的文件夹

3.src/main/resources：管理项目的资源文件，例如spring.xml、c3p0config.properties等

4.src/test/resources:管理测试代码的配置文件

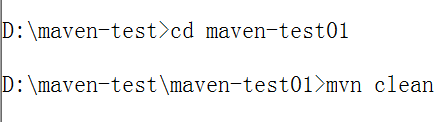
可以看到，maven项目的结构非常清晰，让开发更高效

**项目清理**

对于maven生命周期管理过程中，会不断根据项目变动，会产生各种文件，为了解决前后文件执行的冲突，一般在执行其他任何命令之前，都会添加一个clean的清空命令

mvn clean

进入到执行清空项目的文件夹下

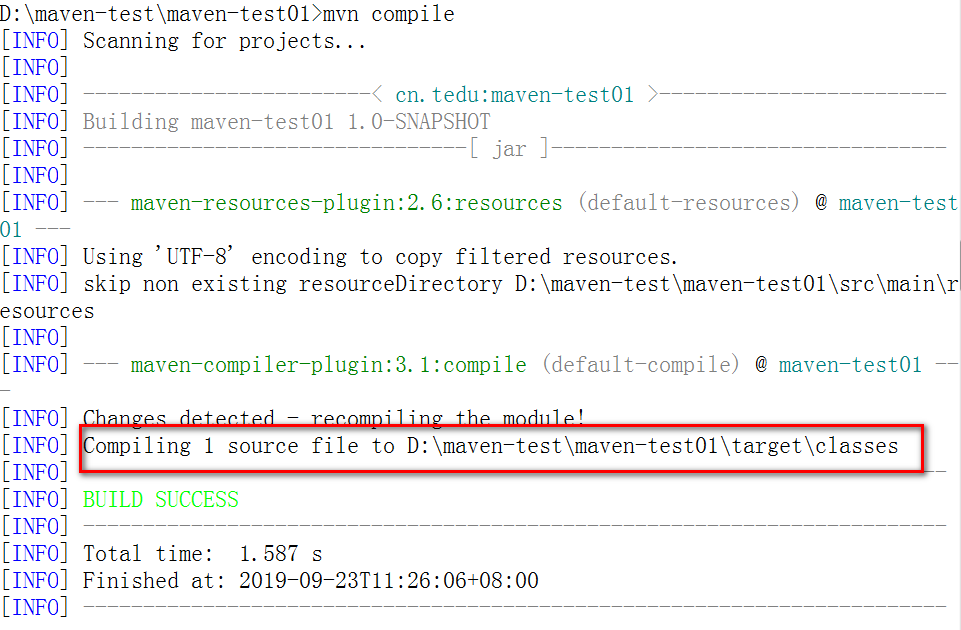


清空的是target文件夹中的内容（编译、打包、安装所有执行都会在target文件夹中生成对应的文件）

**项目的编译**

mvn compile

会将当前工程的所有源码和配置文件，编译输出到项目根目录的target/classes中，test包中的所有内容不会编译、不会打包、不会安装，但是可以测试



**项目测试运行**

mvn test

编写代码时，经常需要测试，可以利用maven提供的测试命令完成测试

例如，在D:\maven-test\maven-test01\src\test\java\cn\tedu下创建测试类MavenTest.java

|  |
| --- |
| package cn.tedu;  import org.junit.\*;  public class MavenTest{  @Test  public void test01(){  System.out.println("what is your name?");  }  } |

然后执行测试命令

**项目打包（jar\war）**

mvn package

将当前的项目打成java工程的包，根据核心管理配置文件pom.xml定义。

Maven项目核心配置文件pom.xml

每个maven项目都对应独立的一个pom.xml文件。记录当前项目的基本信息，例如groupId等

有一个packaging标签，定义当前项目打包类型

<packaging>jar</packaging>

一般只需打包命令前，先执行clean（mvn clean package），package命令包含了之前生命周期所有过程，包括编译、测试、打包等，为了避免冲突，先清空

**项目安装**

mvn install

Maven项目作为maven管理的资源，不仅表示一个运行的工程，还可以作为别人可以使用的资源，但是必须经过maven的安装过程，生成maven库可以使用的资源文件

**项目发布**

mvn deploy

将项目安装的过程是把maven项目封装整理成库可以使用的文件结构，如果有远程团队需要使用当前开发的maven项目资源，需要发布到公司的私服使用；

## 1.4maven的插件

作用：在打包、安装时生成的jar中制定指定的程序入口，也就是指定main方法

加入插件的方法：在pom.xml的跟标签下，创建<build><plugins></plugins></build>标签

在<plugins>标签中可以添加很多个插件<plugin>标签

### 1.4.1.main方法插件

|  |
| --- |
| <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-jar-plugin</artifactId>  <configuration>  <archive>  <manifest>  <addClasspath>true</addClasspath>  <mainClass>cn.tedu.App</mainClass> <!-- 此处为主入口-->  </manifest>  </archive>  </configuration>  </plugin>  </plugins>  </build> |

### 1.4.2.打源码包的插件

在项目打包过程中，生成除了jar包外，还要绑定生成source源码。

|  |
| --- |
| <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-source-plugin</artifactId>  <configuration>  <attach>true</attach>  </configuration>  <executions>  <execution>  <phase>compile</phase>  <goals>  <goal>jar</goal>  </goals>  </execution>  </executions>  </plugin> |

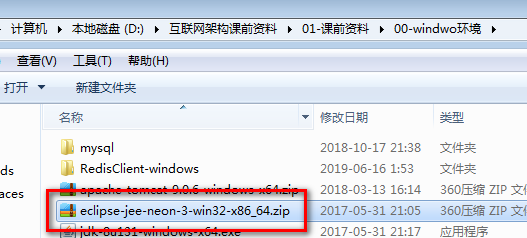
## 1.5.eclipse的maven整合

Maven作为一个项目管理的软件，具备各种执行的命令，调用插件资源的jar包执行，需要配合开发工具使用，需要eclipse、idea等将maven整合

### 1.5.1.使用eclipse的版本

Mars2稳定版（可以使用）

Neon版本（肯定可以使用）



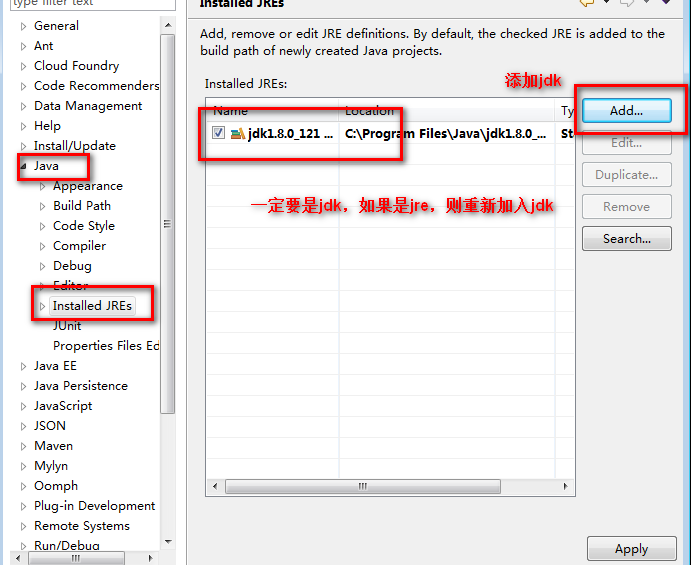
直接将eclipse压缩包解压即可，解压目录不能有中文空格

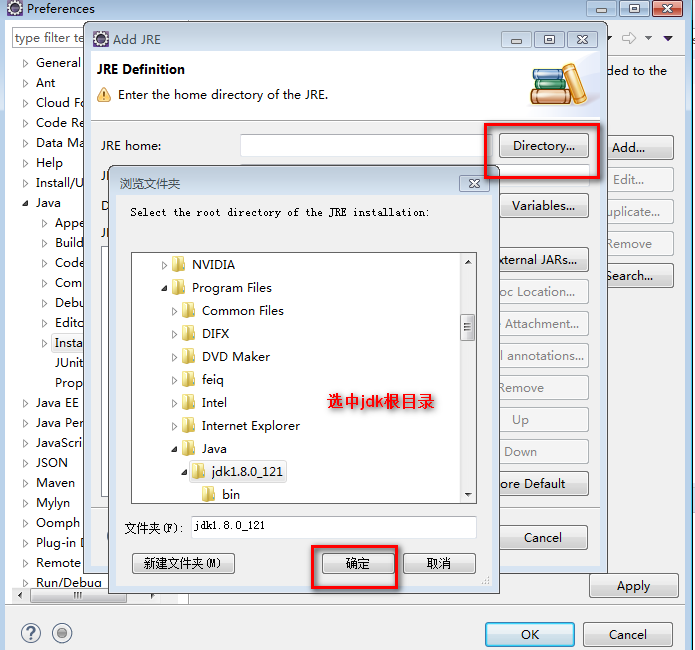
### 1.5.2.调试eclipse的环境

1.创建工作空间

**2.java的库资源必须是jdk**

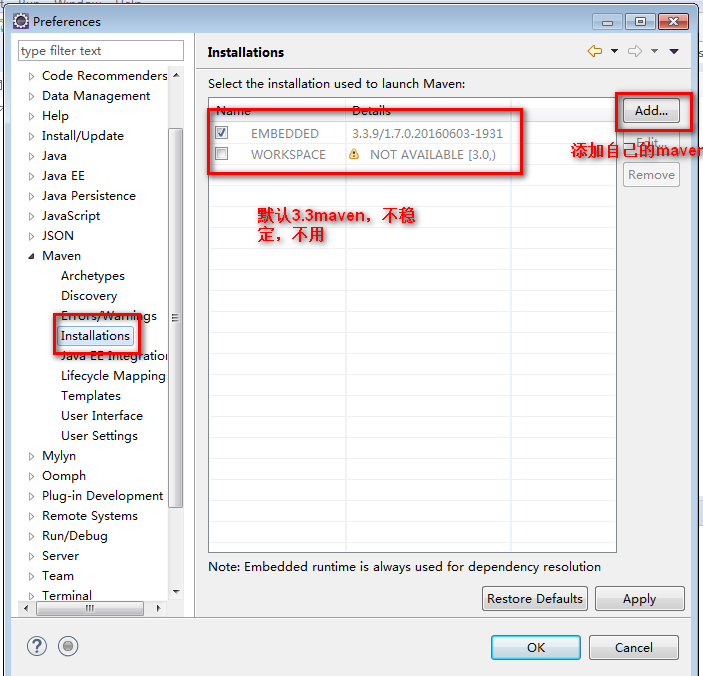
Window->preferences->java

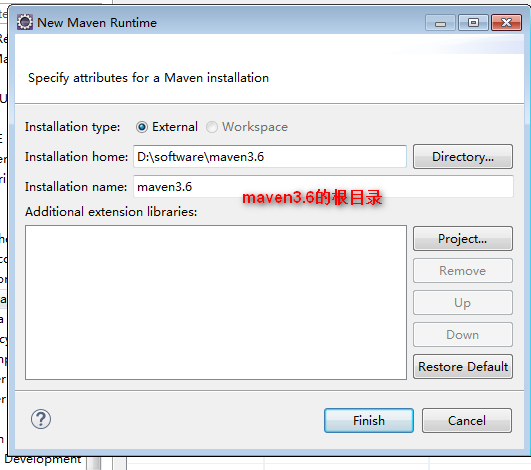


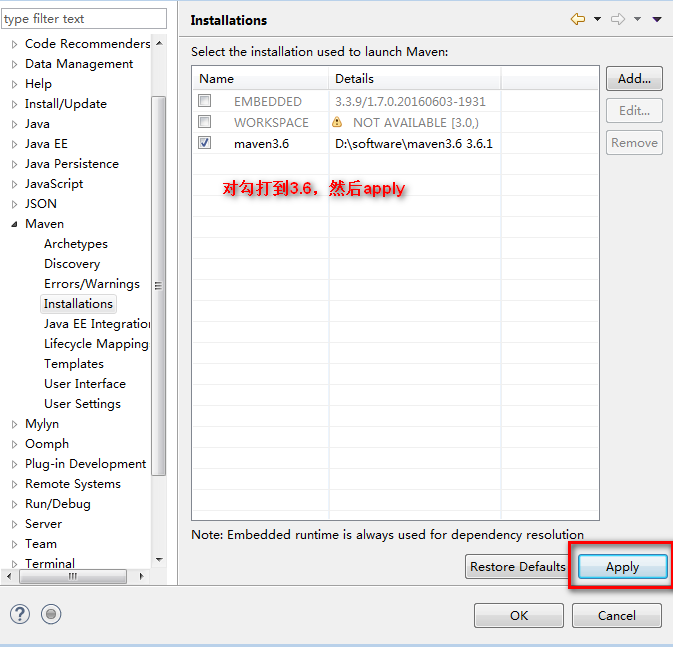


**3.Maven插件安装**

Window->preferences->maven

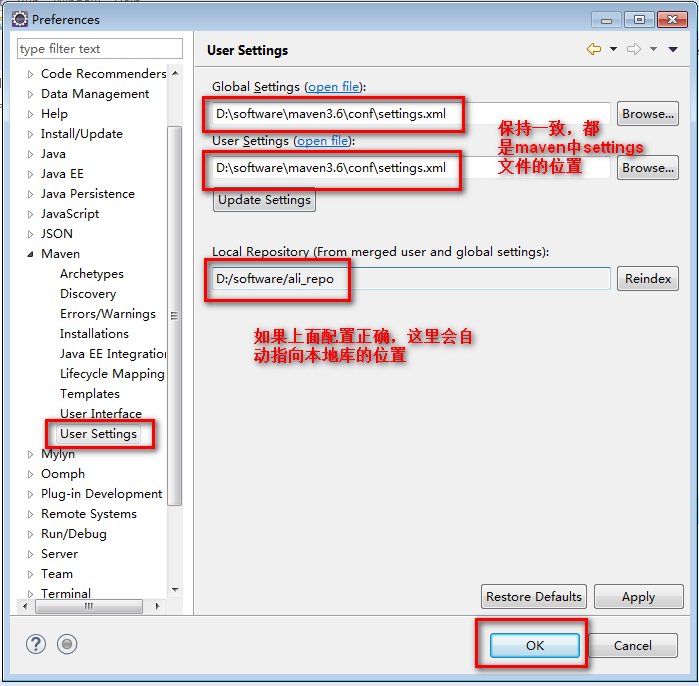






**4.绑定配置文件settings.xml**

Window->preferences->maven



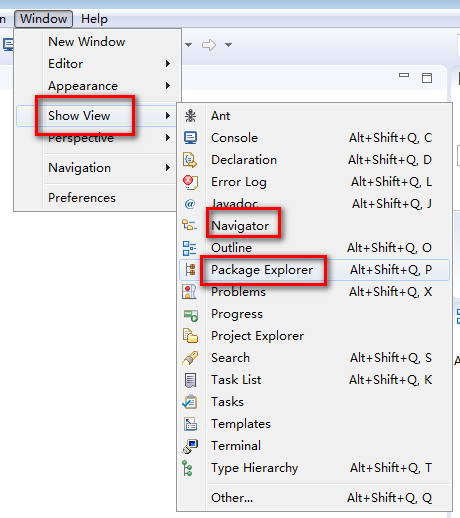
**5.调整eclipse的视图结构（必须和我保持一致）**

Package explorer(项目管理视图结构)

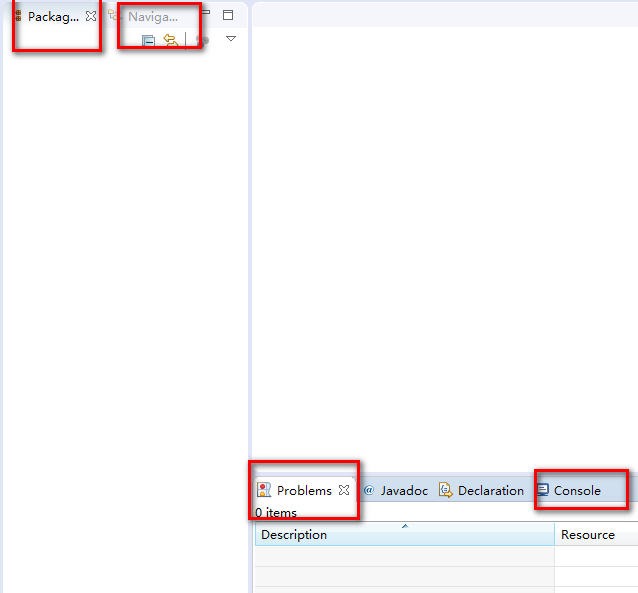
开发项目使用

Navigator（项目磁盘机构视图）

复制粘贴时使用



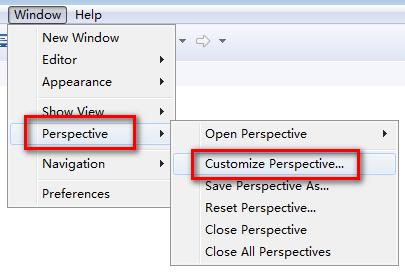
保证有四个标签

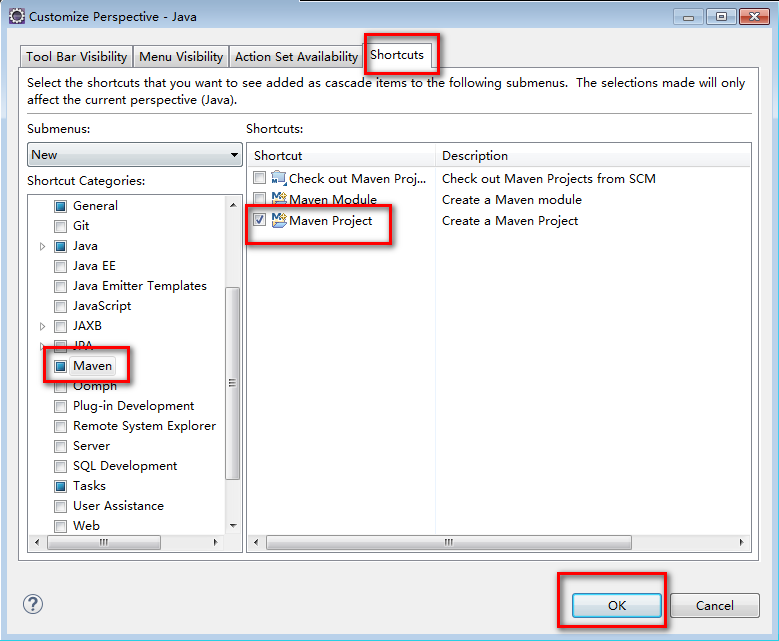


## 1.6.eclipse创建maven测试工程

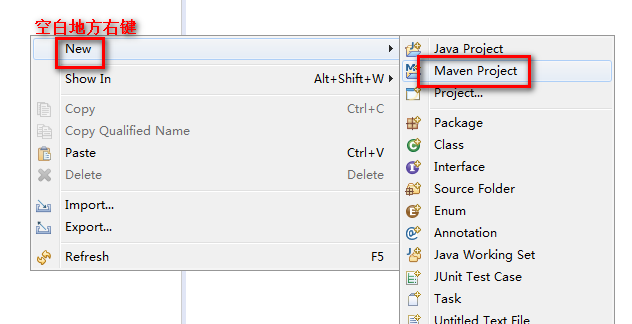
### 1.6.1.创建一个eclipsemaven工程

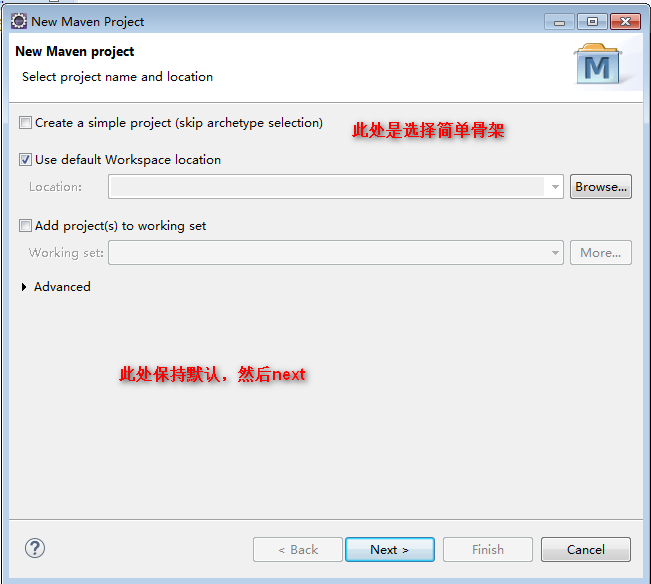
在new中添加mavenproject（自己想要的视图）

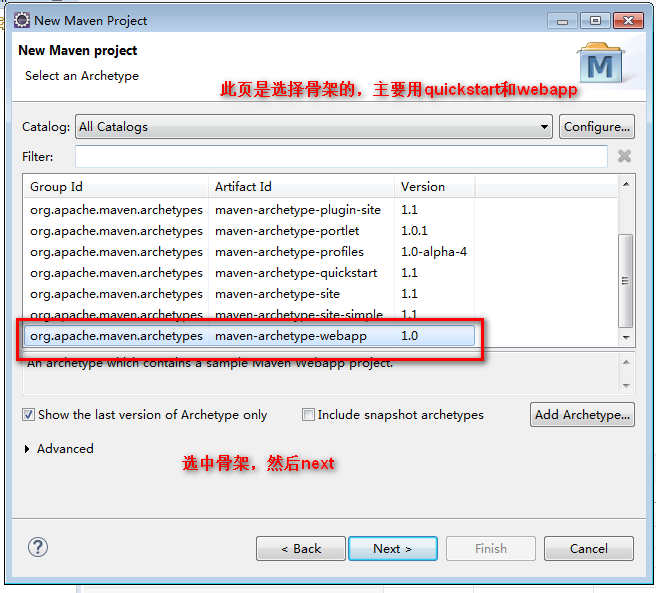


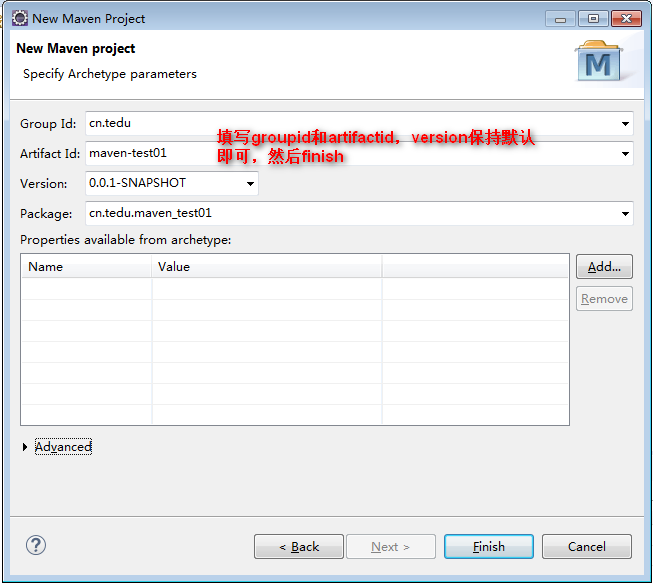


创建maven项目









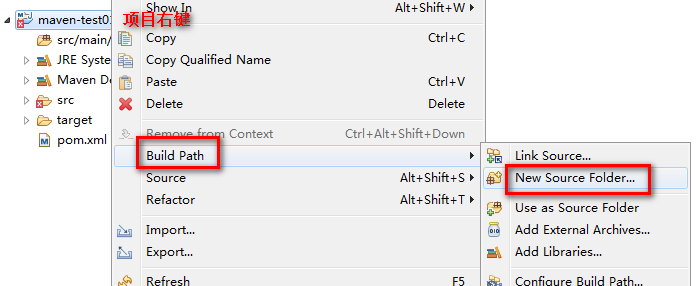
保证一个项目至少三个文件夹结构

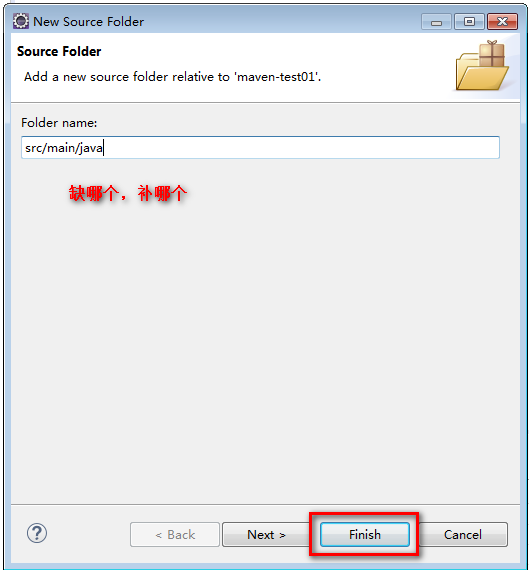
Src/main/java:编写源码

Src/main/resources：项目配置文件目录

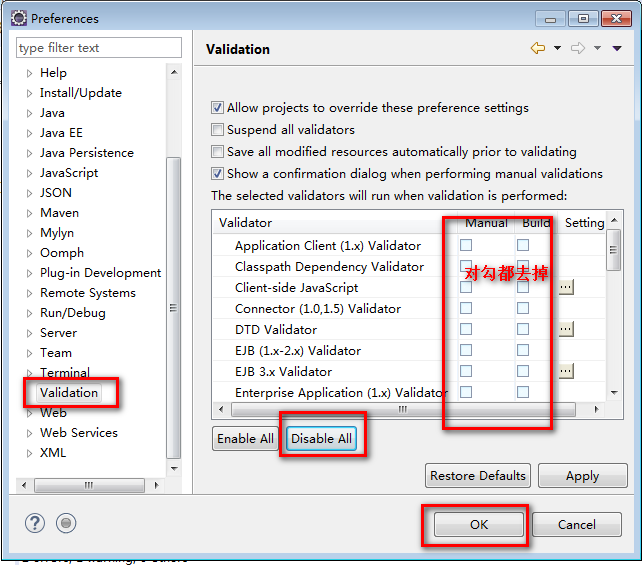
Src/test/java：测试代码位置

缺少任意一个，利用build path创建





报错问题：并非操作或代码问题，而是jsp校验报错，可以修改eclipse校验

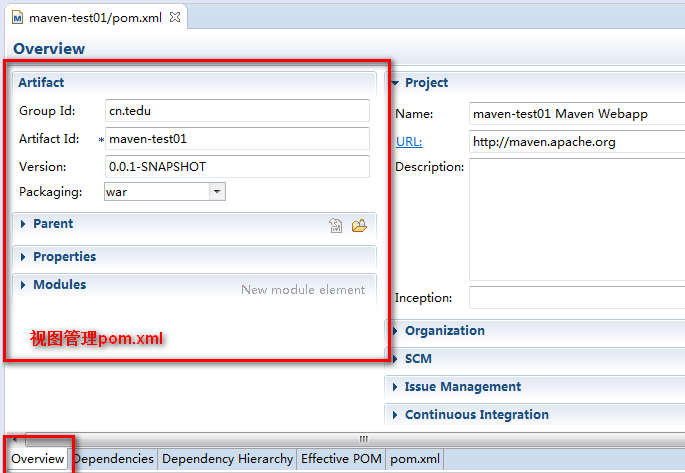


### 1.6.2.依赖资源的使用

Maven项目中的pom文件是管理项目的核心配置文件，<dependencies>标签保管的就是当前项目使用的所有依赖资源（jar包）

Eclipse中maven工程的pom文件管理

Overview:概括，总览，利用eclipse的插件对pom.xml文件解析后的视图

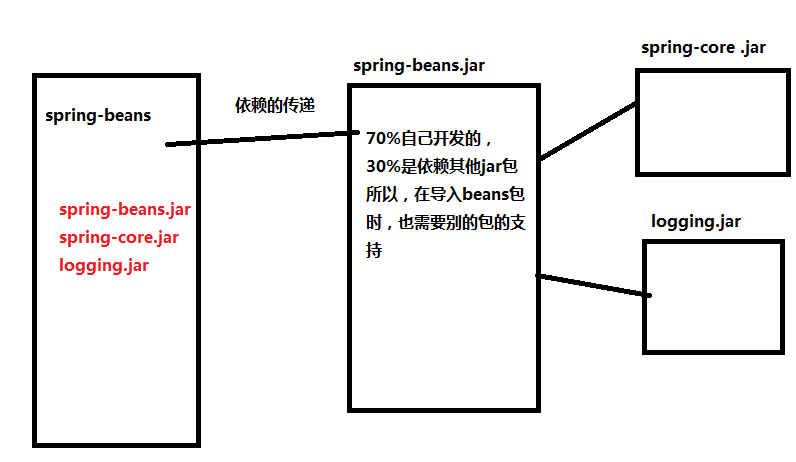


引入一个spring对应资源依赖spring-context

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>4.3.7.RELEASE</version>  </dependency> |

通过dependency的标签引入资源，可以在maven工程中直接使用依赖传入的所有jar包，例如，依赖spring-beans时，maven会自动将spring-core也传入进来

依赖的传递性



每当在dependency中添加一个依赖资源，本质上依赖资源都是创建maven工程编写代码，生成资源，放到库中使用，在开发工程中，如果依赖了其他的maven资源，可以传递给下一级；可以通过移除将不需要的依赖清除。

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>4.3.7.RELEASE</version>  <!-- 移除不需要的资源 -->  <exclusions>  <exclusion>  <!-- 需要通过坐标移除指定内容 -->  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  </exclusion>  </exclusions>  </dependency> |

依赖的资源使用范围

依赖中使用标签<scope></scope>

compile：编译范围，默认的范围，编译、打包、安装、发布的全部生命周期都存在的依赖资源

test：测试范围使用，打包、安装都不参加

runtime：运行时范围，和compile唯一的区别，就是不参加编译，例如jdbc相关依赖，在spring框架中并无相关代码，所以无需编译，但是底层是用的jdbc实现的，必须用到jdbc依赖

provided：和runtime正好相反，在编译时参加，别的地方都不用，官方给出一个列子，servlet-api。

system：系统范围，在当前项目的环境中存在需要使用的jar包资源，maven没有提供groupId、artifactid、version，可以使用suystem指定本地路径，

|  |
| --- |
| <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>1.1.0</version>  <scope>system</scope>  <systemPath>D:/software/maven-test/spring-beans-4.3.7.RELEASE.jar</systemPath>  </dependency> |

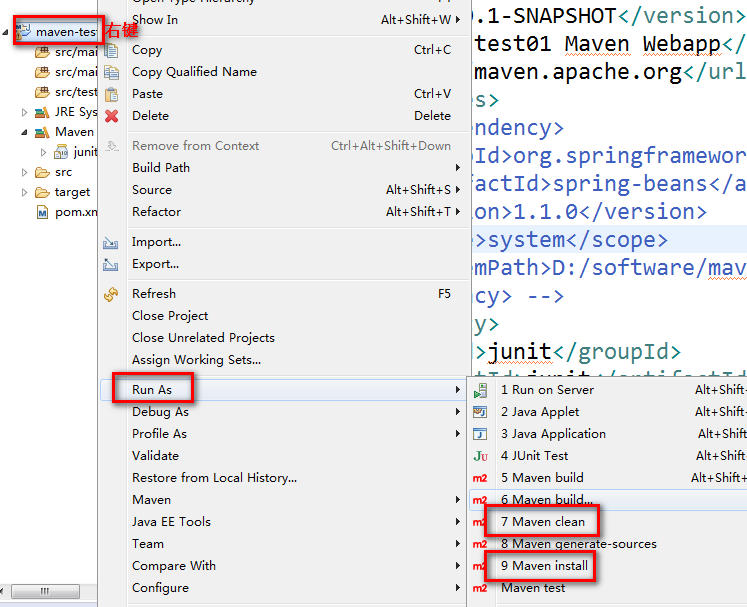
### 1.6.3.eclipse调用mvn各种命令

Eclipse中自动生成2个命令执行的包装操作

mvn clean（相当于mvn clean）

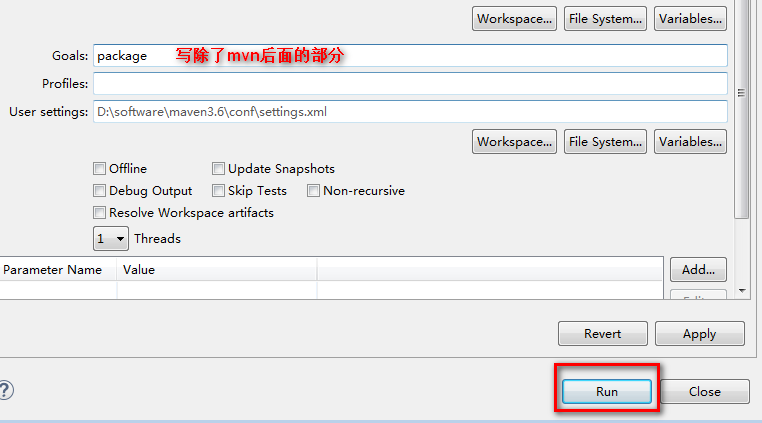
mvn install（不完全等于mvn install 底层运行其他命令的封装，例如对依赖包的管理等）

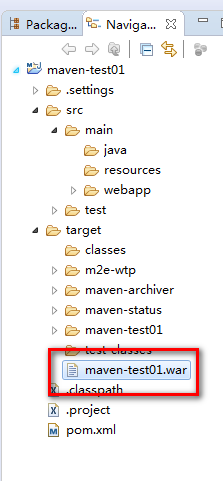
mvn test（不完全等于mvn test命令，生成报告文件以外的一些操作也封装了）



自定义执行命令

执行maven build





将需要的命令写到goals中，例如mvn package，只需要写出了mvn的部分即可

**整合问题**

1.maven环境问题

Downloading...

原因：

Maven环境没有调试正确（没有指向本地库）

注意：MAVEN\_HOME、path中有没有引入正确的变量

Maven根目录的conf/settings.xml中55行左右，指向正确的本地库路径

还有可能是本地库中没有此资源