Maven-Day01

教学内容：如何使用Maven创建项目、如何导入jar、管理jar、项目的生命周期、项目的分解和聚合

今天的主要安排：

1. Maven的概述（为什么要用Maven?Maven是什么?）
2. Maven快速入门,安装配置
3. Maven依赖的快速入门使用（配置、创建java项目和web项目）
4. Maven整合servlet
5. Maven传递依赖冲突

# Maven的概述和技术简介

## 什么是maven

Maven这个单词来自于意第绪语，意为知识的积累，用来简化项目构建管理过程。

Maven的作用：

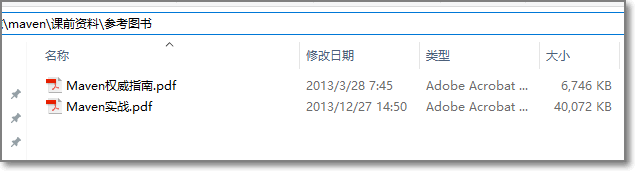
依赖管理:改变传统的jar包管理方式,一个或多个项目只需要配置一个文件即可实现jar包的依赖

在之前的项目开发中,我们搭建的一些架构通常都需要依赖框架或工具类的很多jar包.而且每个项目都需要重新copy一次

而现在我们只需要通过maven的pom.xml配置文件引入所需要依赖的包的坐标即可

项目构建:采用统一轻便的方式构建管理项目,通过简单的命令帮助完成复杂的清理,编译,测试运行等项目的生命周期管理

更多参考详见课前资料：



Maven管理项目的周期：



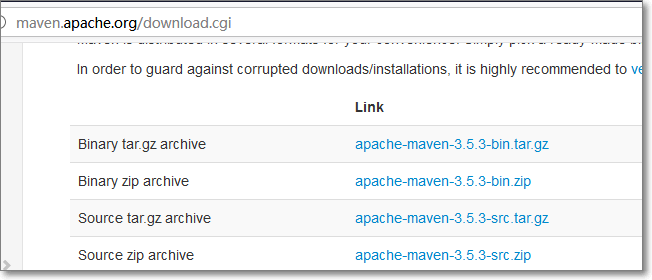
# Maven的快速入门

## Maven的下载安装

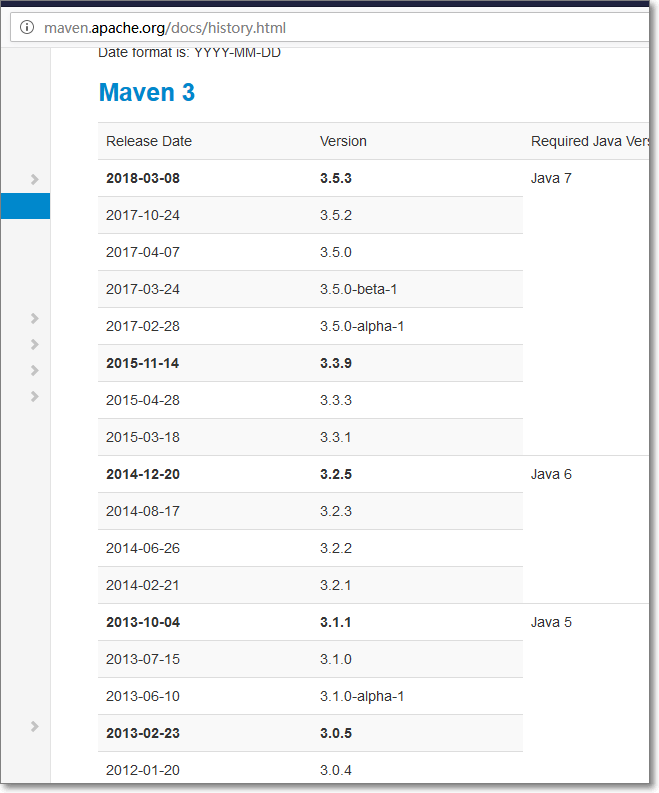
官方网站：<http://maven.apache.org/>

下载页面：<http://archive.apache.org/dist/maven/maven-3/>

官网的系列版本：

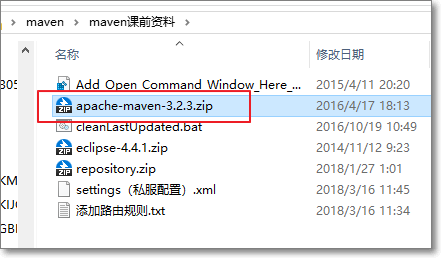


版本选择问题：

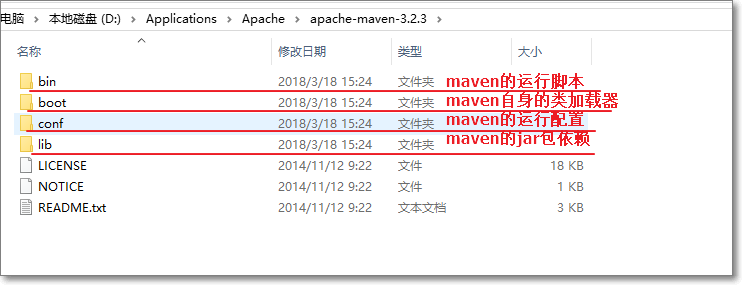


本课程选用3.2.3

Windows环境请下载maven zip 包：



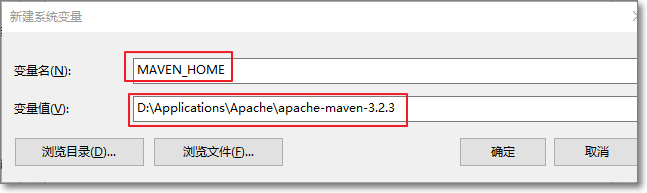
解压maven的zip包 （注意解压的路径，建议不要有中文、空格、特e符）



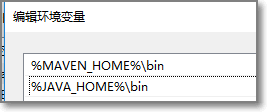
## Maven的配置

配置maven的环境变量：

* MAVEN\_HOME=maven压缩包解压目录。



* 在path环境变量中，增加 %MAVEN\_HOME%\bin



测试是否配置成功：

重新打开cmd窗口，运行：mvn -v

出现下图的信息，说明环境配置正确且生效了：



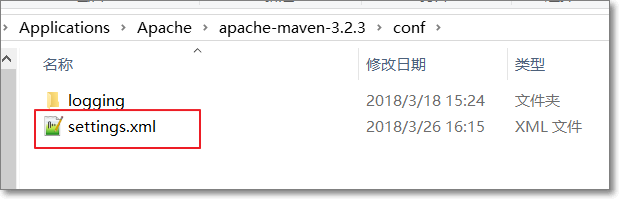
## Maven仓库配置

什么是仓库？

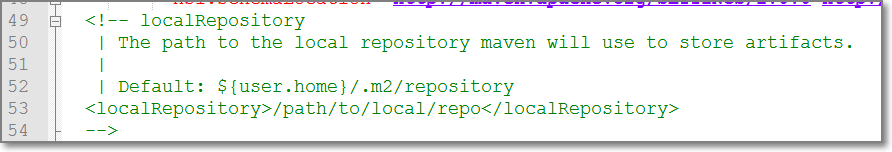
首先大家要清楚，jar包或者war包其实就是一个个java项目或web项目

Maven的仓库就是用来存放项目所需要的jar包和插件的 ，再简单的说，仓库就是为了存放管理jar包的。

仓库的位置是通过maven的核心配置文件（settings.xml）来配置的。



默认的仓库位置:

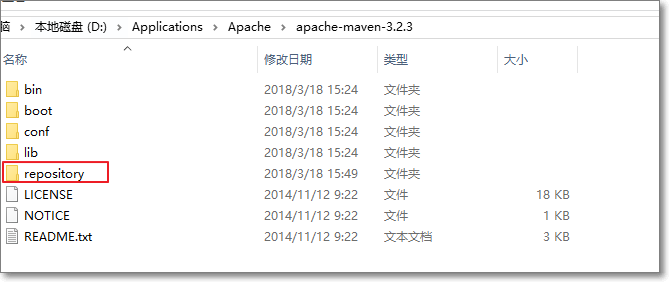


发现：默认的仓库位置是在当前登录用户的home目录下：

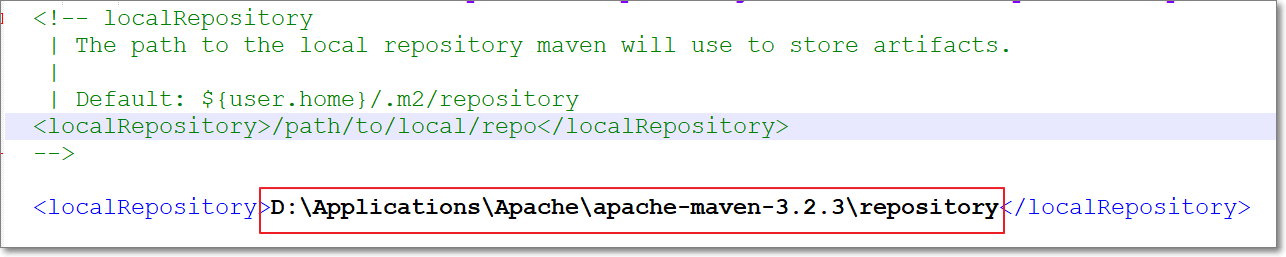


修改默认仓库位置为自定义的仓库位置：

新建repository文件夹：



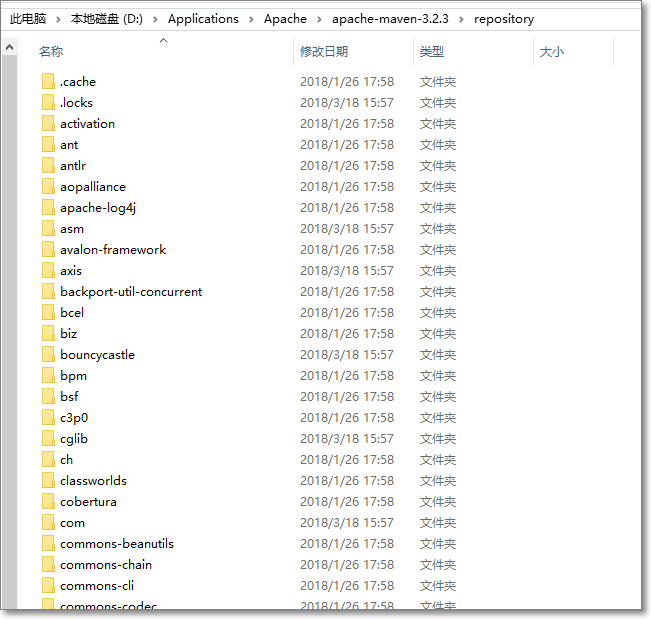
在核心配置文件中增加仓库的位置的配置：



## 本地仓库的建立

Maven要运行使用，需要一些插件（插件也是一些java项目）来支持，这些插件需要联网下载。maven会优先从本地仓库中寻找，如果没有则上网下载，将下载好的插件缓存到本地仓库中，下次运行时再需要这些插件时就会直接使用本地而不需要再次联网下载了，项目中需要导入的jar包也是同样的套路。

将课前资料中“本地仓库”中的已经下载好的仓库内容，解压到仓库文件夹：



注意：注意解压文件的目录层次！

## Maven相关名词解释（预备知识）

* Project:Maven创建的工程项目,这些工程被定义为工程对象模型(POM:Poject Object Model). 一个工程可以依赖其他的工程,可以由多个子工程构成.
* POM:POM(pom.xml)是Maven项目的核心配置文件,它是指示Maven如何工作的元数据文件,比如该项目都依赖了哪些其他项目,POM文件位于每个工程的根目录中.
* GroupId:是一个工程在全局中唯一的标识符,好比是一家公司的名称，是工程完整的一个包路径，如: cn.itcast.maven,用以区分不同的工程.
* ArtifactId:创建的工程,比如：demo ,每个artifact都由groupId（cn.itcast.maven）和artifactId(demo)组成的唯一标识如(cn.itcast.maven.demo)
* Version:是当前生产的工程版本信息
* Dependency:一个典型的Java工程会依赖其他的包,在Maven中,这些被依赖的包就被称为dependency, dependency一般都是其他工程的GAV.
* Plug-in:Maven是由插件组织的,它的每一个功能都是由插件提供. 插件根据pom文件中的配置的dependency和相关配置完成工作.

**相关概念的关系：**

新建maven工程称之为project， 每个maven工程都包含 pom.xml 描述文件（maven工程的核心配置文件）

一个工程区分其它工程的方式，是通过 groupId（组）和 ArtifactId（唯一标识）来区分的。

一个工程可以通过 dependency（依赖的jar包） 配置依赖其它工程

## 常用的Maven命令

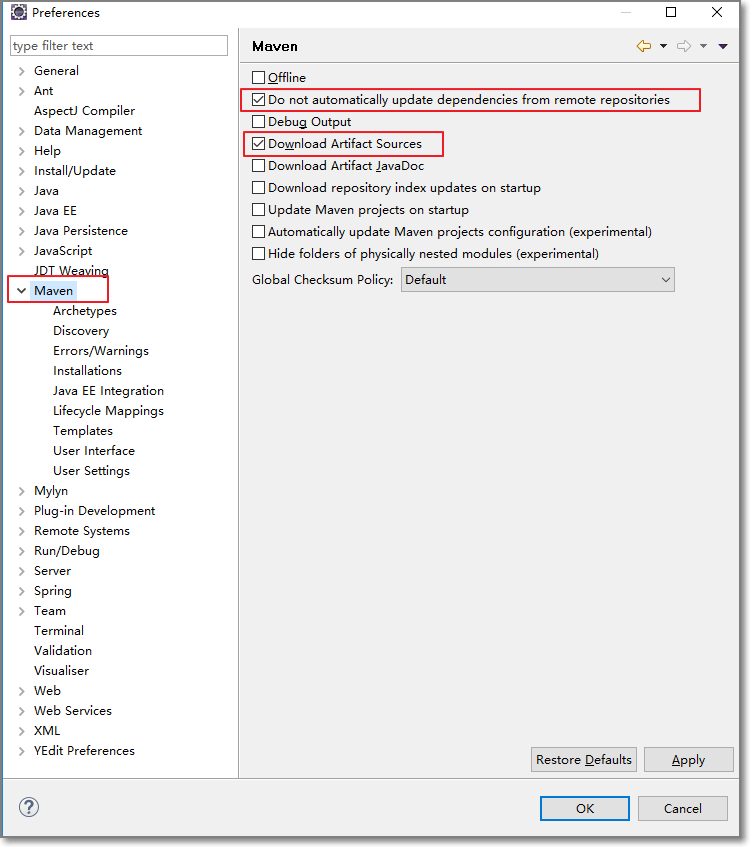
* **编译**  
   compile 将项目下的java文件编程成class文件
* **测试**  
   test 执行目录下的单元测试类
* **清除**  
   clean:清理项目中target目录
* **打包**  
   package 将项目打包到target目录下
* **安装**  
   install 将项目打包成构件安装到本地仓库
* **发布**  
   deploy 发布到本地仓库或服务器(如: tomcat,JBoss)

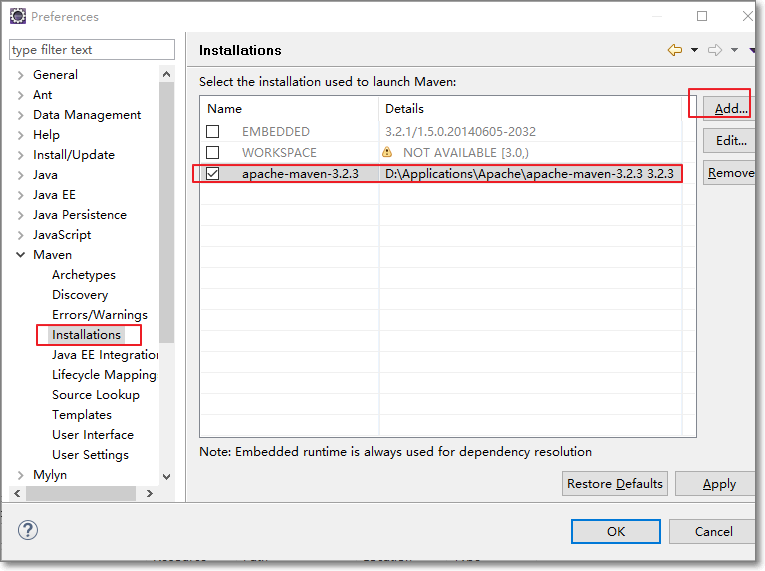
提示：生命周期从上到下执行，如果执行后面命令，自动执行之前项目构建步骤。

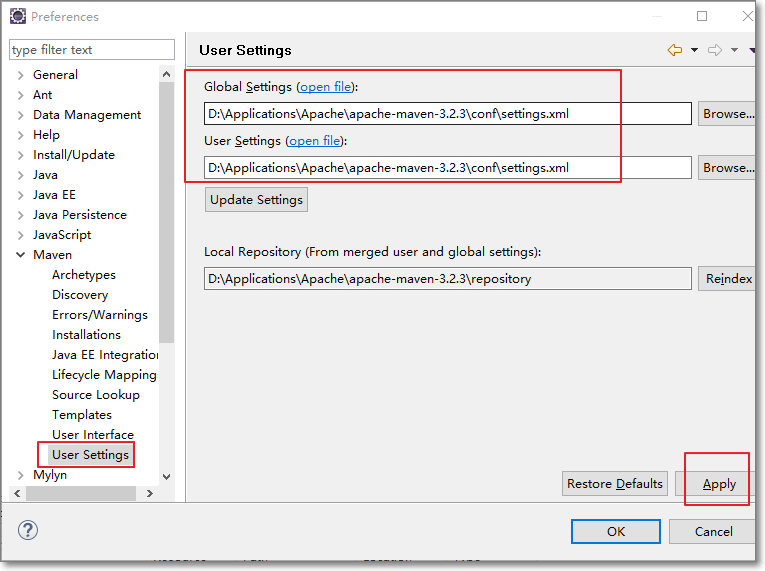
# 通过Eclipse工具配置MAVEN构建工程项目

## 在Eclipse工具集成maven配置

window –preferences:



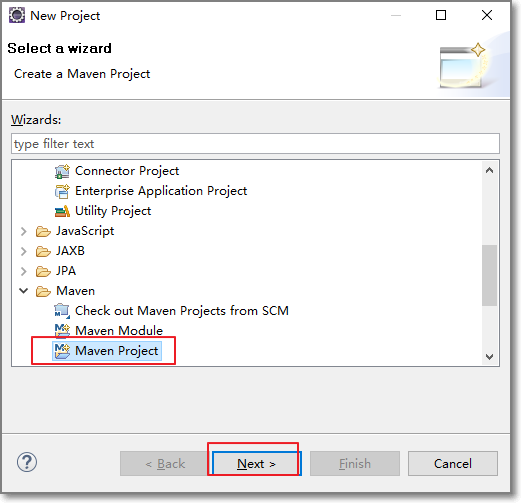




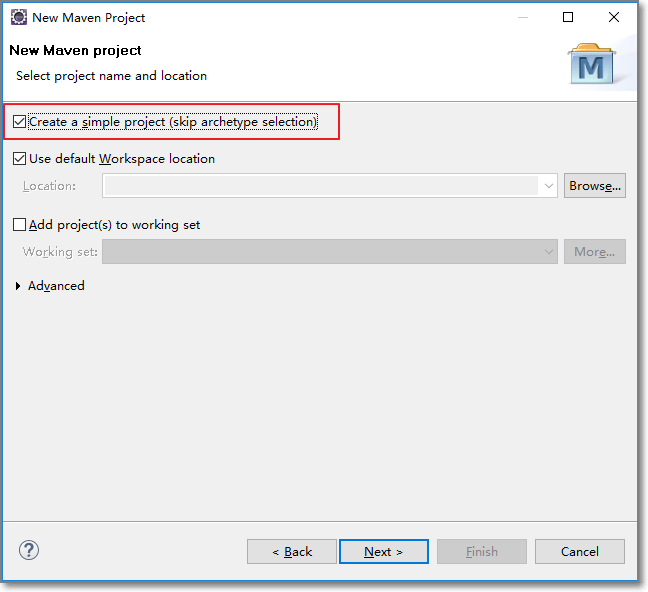
## 构建工程项目

### 使用maven创建java项目

新建一个maven java项目：



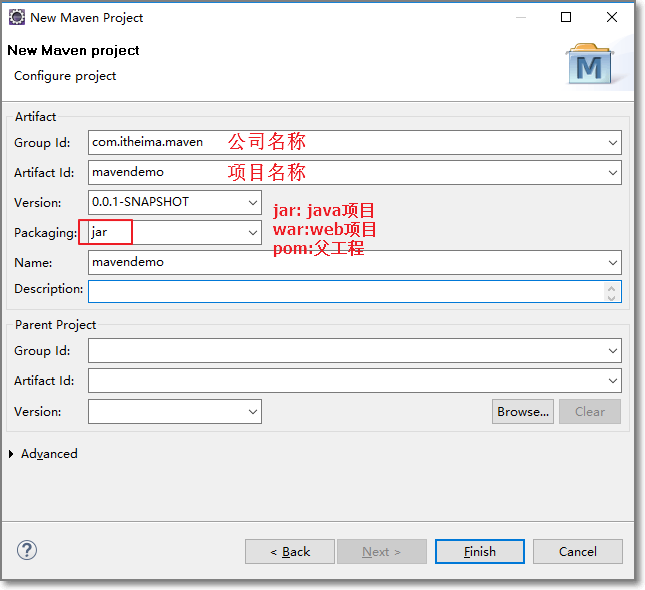
下一步：跳过骨架选择创建



**什么是骨架（archetype）?**：创建项目的模板。

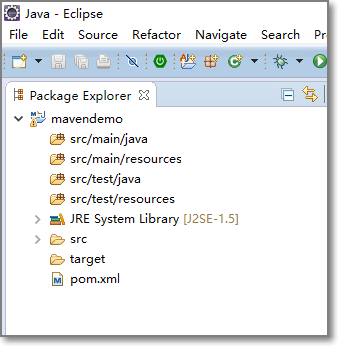
提示:跳过骨架选择的原因是通过骨架创建的项目结构不完整,需要额外配置.

下一步：配置maven工程的参数：

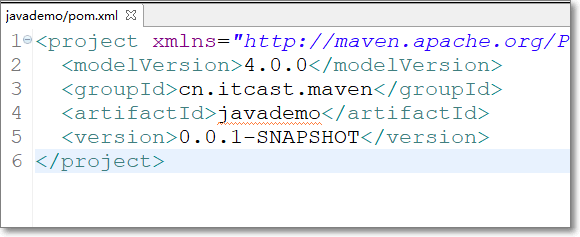


点击finish完成工程创建。

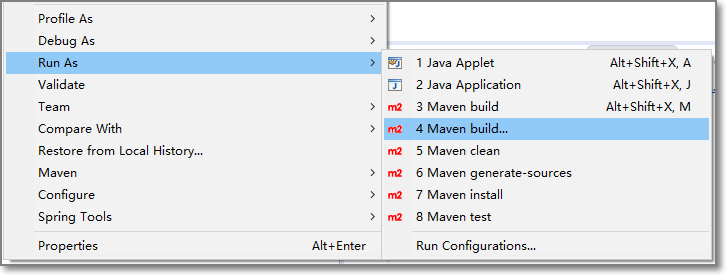
建立起来的工程目录结果—简单的java项目结构：



在项目的pom.xml中有关于项目的自身描述信息，后期我们依赖的其他项目也会出现在这里

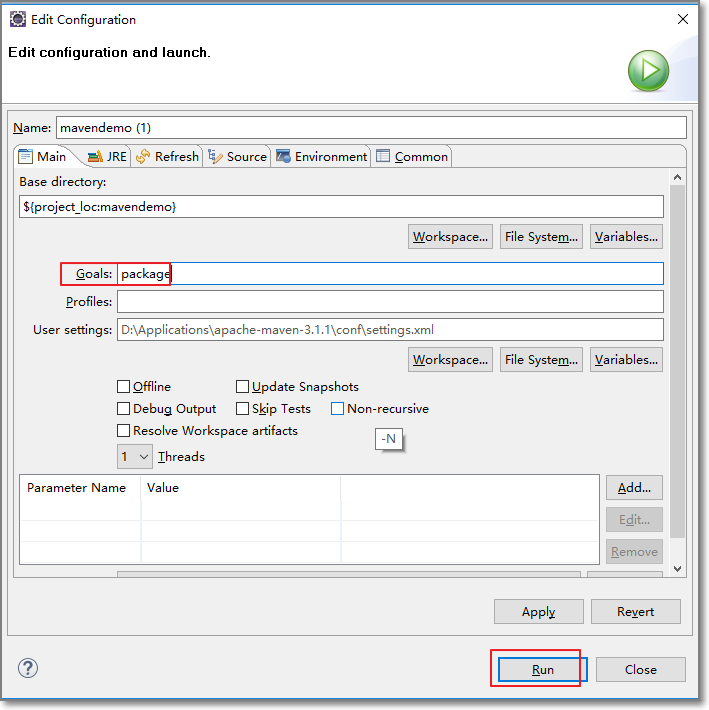


右键项目run as 查看能够使用maven 命令

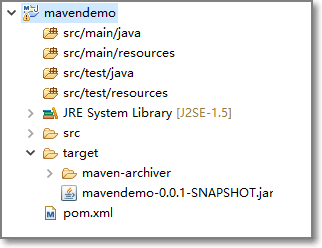


通过手动输入命令，来执行mvn package命令：

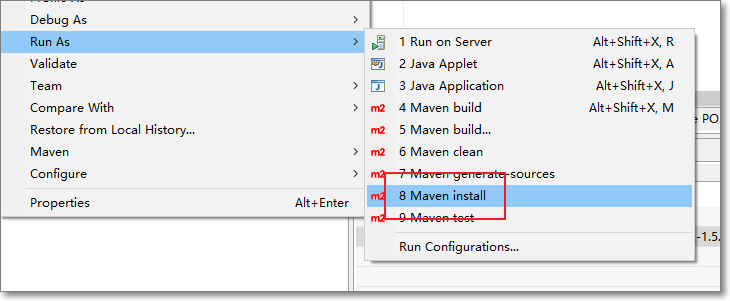
Package命令会将当前项目打成jar包放置在target目录下

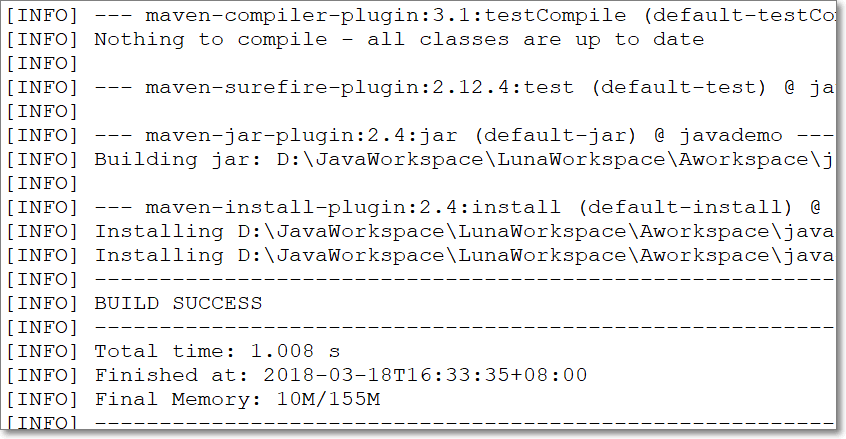


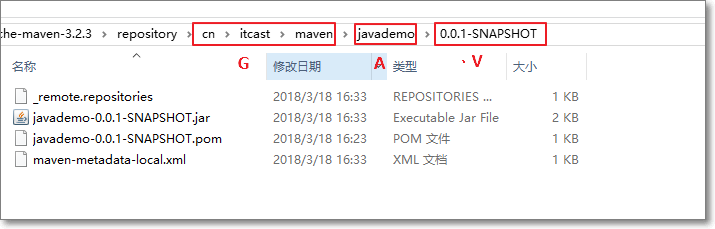
查看打出来的jar：



通过install 命令 将打好包的项目安装到本地仓库中：





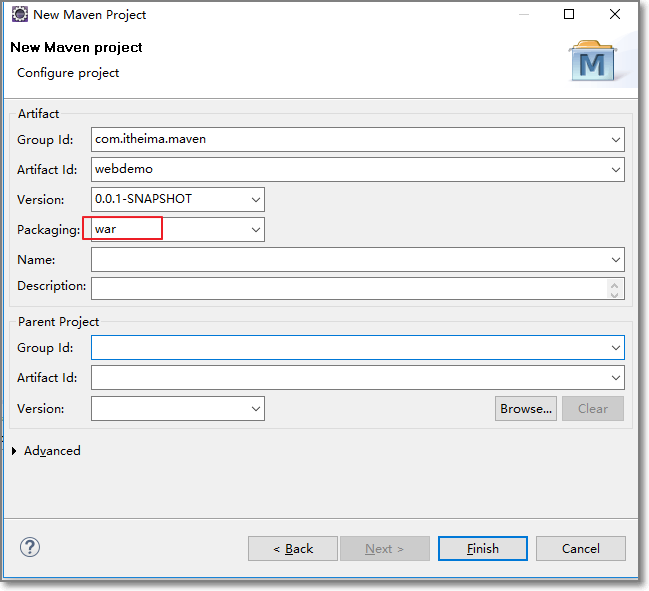


在本地仓库中看到我们的自己创建的项目打成的jar包了，如果在其他项目中依赖我们的项目，需要在poml.xml中配置

该项目的GAV坐标

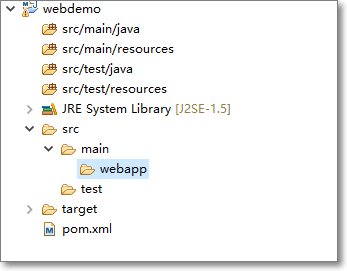
### 使用maven创建web项目

仍然跳过骨架创建web项目:packaging属性选择war 表示为一个web项目

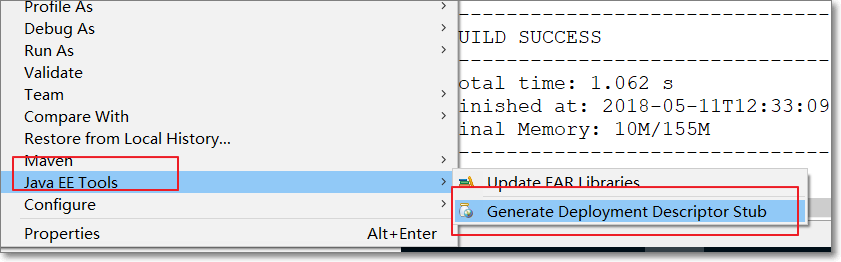


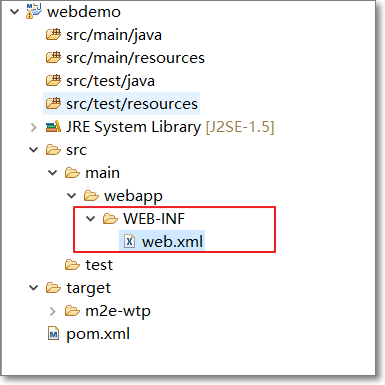
其中webapp为web相关配置的根目录，等同于通过eclipse创建的webContent文件夹，

通过这种方式创建的web项目不会帮我们自动生成WEB-INF目录和web.xml文件，我们需要自己将项目补全



通过工具补全web项目:右击项目





# 通过maven工程去引入依赖

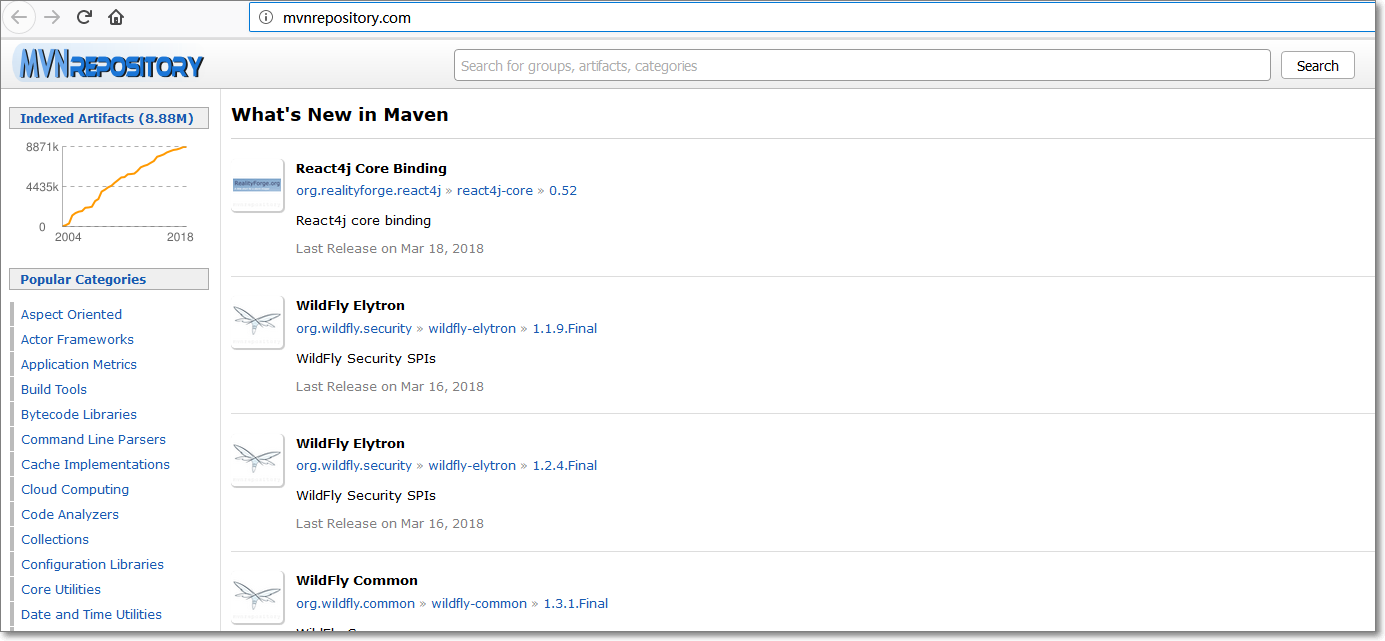
通过maven引入依赖的两种方式：

## 通过maven中央仓库下载jar包到本地仓库进行依赖

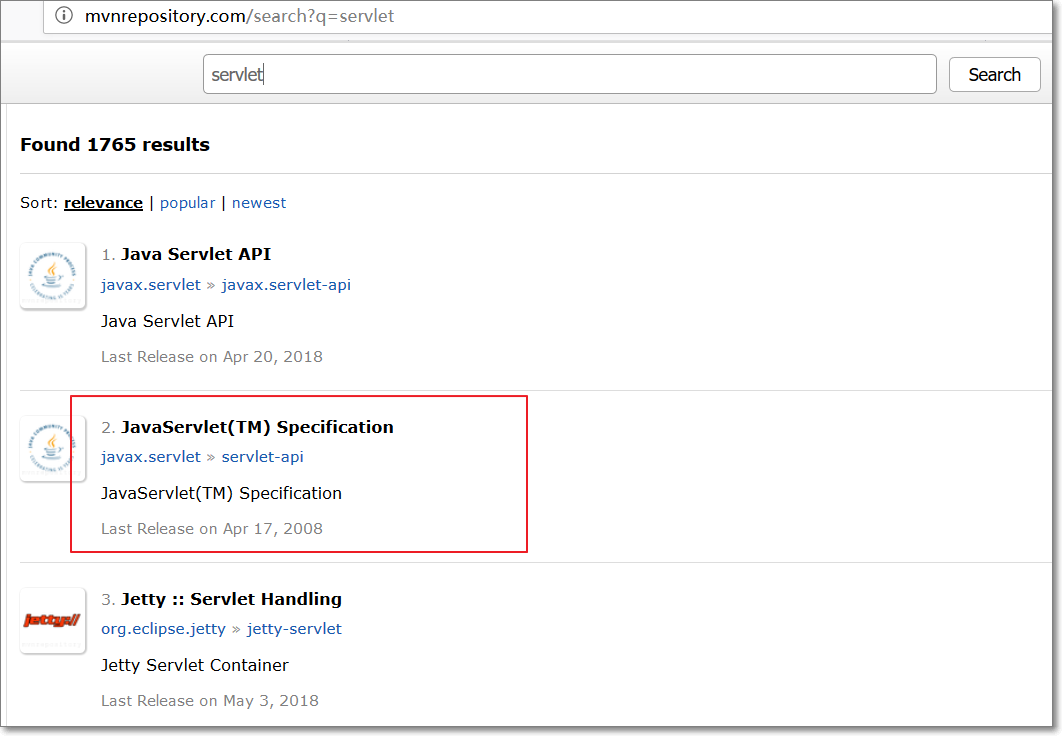
当项目中引用的依赖本地仓库中不存在的时候， maven会自动从中央仓库直接进行下载并存放至本地。

Maven官方的中央仓库地址：<http://mvnrepository.com/>

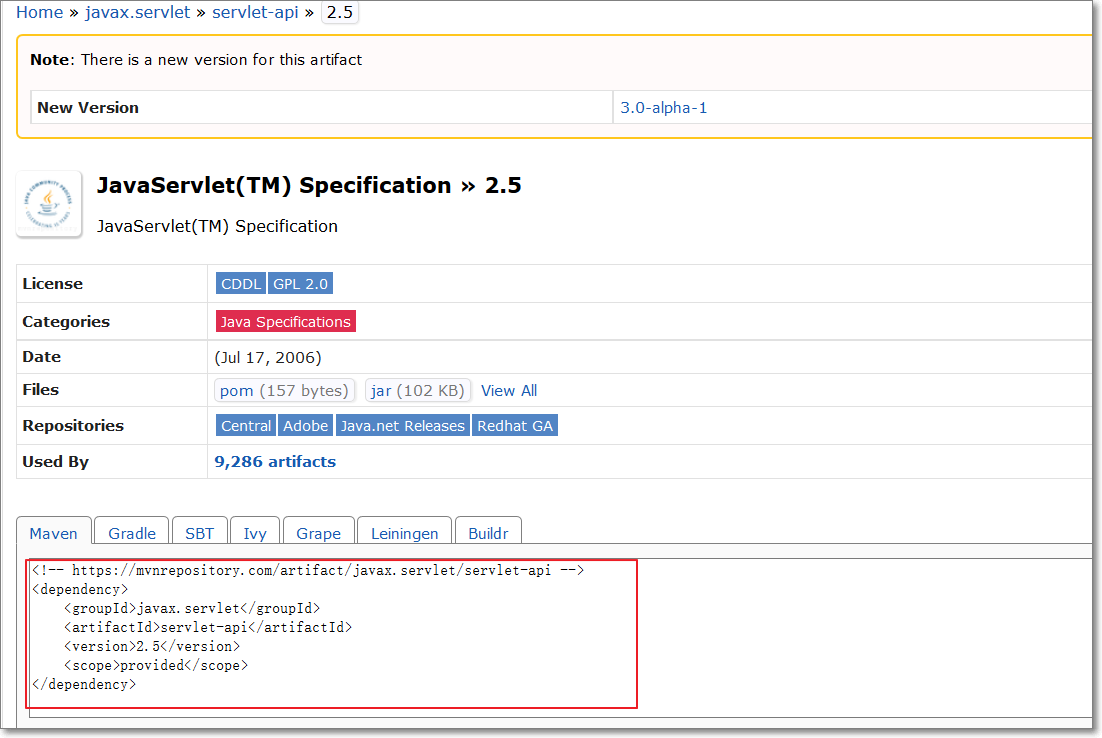
访问maven的中央仓库：



搜索想要的jar包依赖:



选择需要版本并拷贝 Dependency：



在本地项目的pom.xml文件中添加dependencies 标签并将拷贝的Dependency粘贴进来：

<dependencies>

<!-- https://mvnrepository.com/artifact/javax.servlet/servlet-api -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

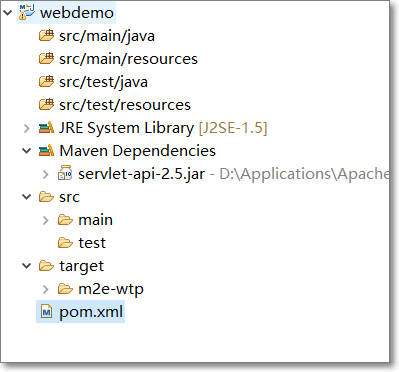
<version>2.5</version>

<scope>provided</scope>

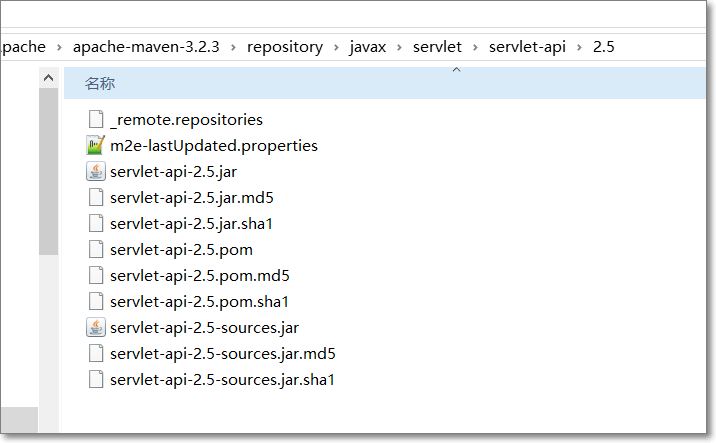
</dependency>

</dependencies>

观察项目中的Maven Dependencies，已经依赖了servlet对应jar包了



同时servlet也会被下载存储到本地仓库中：



## 通过公司私服仓库下载jar包到本地仓库进行依赖

私服仓库的使用一般是在公司内部，相当于是一个局域网内的大一点的本地仓库，私服中的jar包是由员工上传或者从中央仓库中下载存储下来的

具体流程为：当本地仓库没有需要的jar包时，会首先从公司内部的私服仓库中下载到本地仓库并进行依赖， 如果公司私服中也不存在的话，私服也会从 maven的中央仓库中进行下载，存储到私服仓库和本地仓库。再次依赖同样的jar包时就会从私服中下载到本地仓库了。

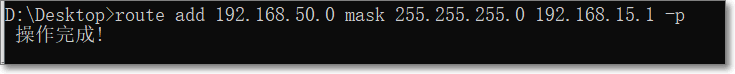
学校私服地址：<http://192.168.50.22:8081/nexus/content/groups/public/>

### 配置本地网卡路由规则

访问私服仓库前我们需要设置本地网卡的路由规则:管理员身份打开cmd：

**route add 192.168.50.0 mask 255.255.255.0 192.168.x.1 -p**

其中 x 对应当前局域网的网关。例如：192.168.15.36, x 就是 15

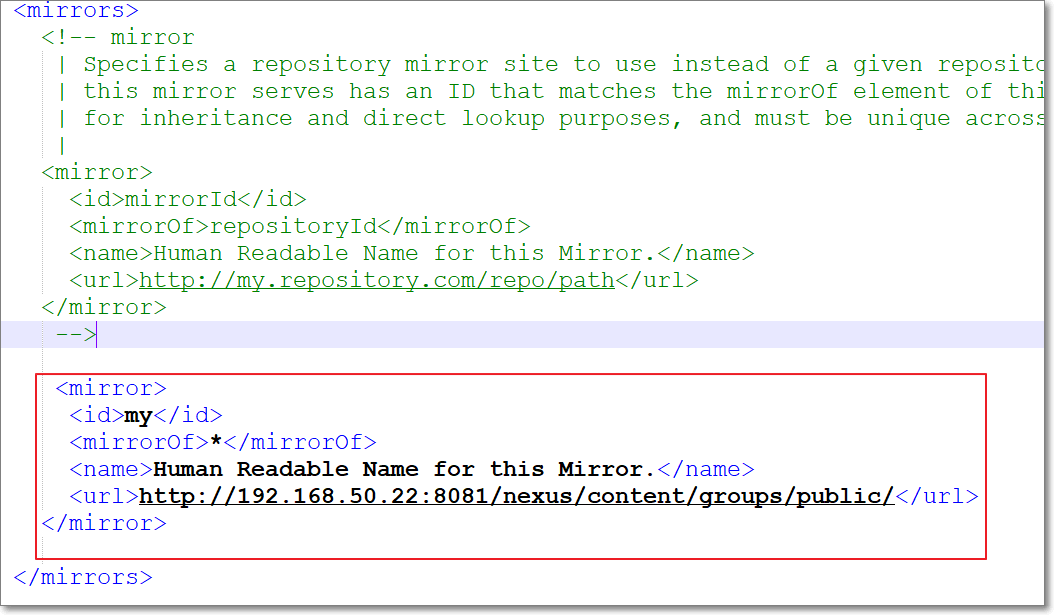


配置路由规则的目的是让我们既能访问外网也能访问本地私服。

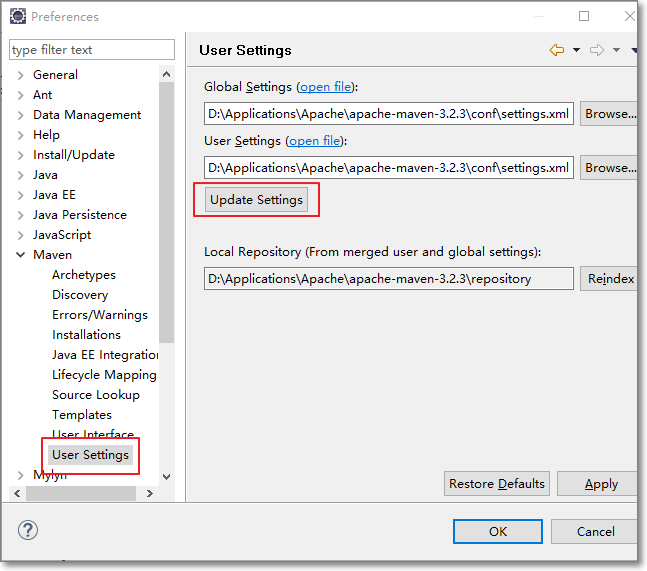
删除路由规则：route delete 192.168.50.0 （不删除没影响）

### settings.xml文件添加私服配置

配置Maven的settings.xml 文件，更改maven的默认下载地址(由中央仓库变为私服仓库)

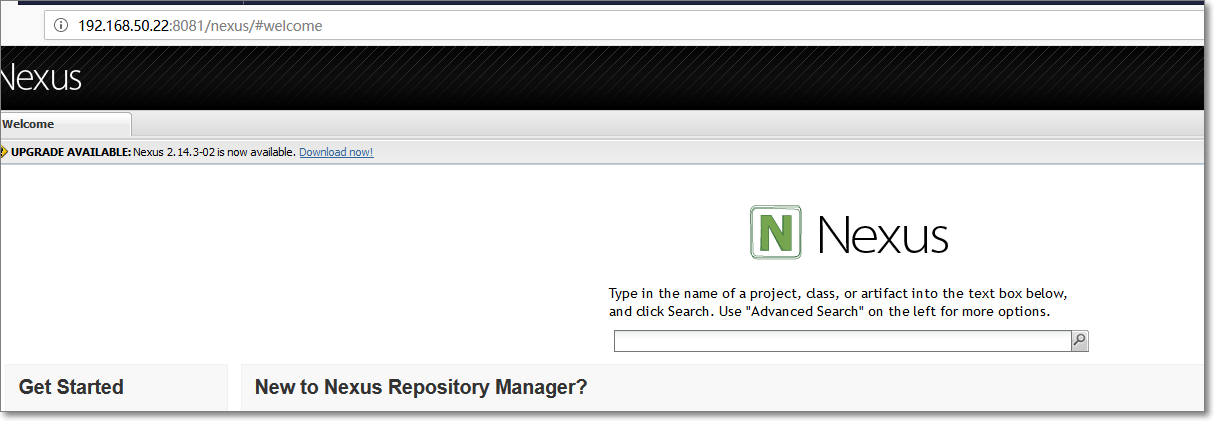


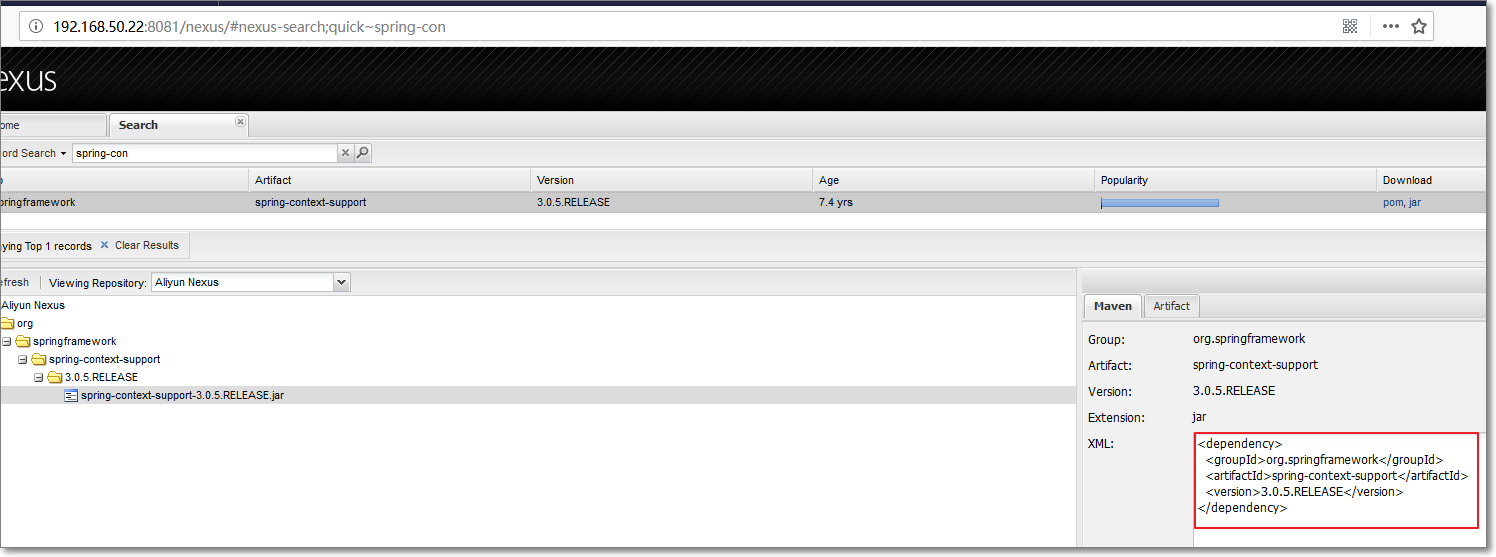
设置完之后在Eclipse中对文件进行Update：



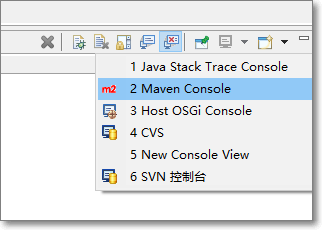
至此，我们以后使用的jar包就全部来自学校的私服仓库了，实际工作时大家使用的会是自己公司的私服

打开私服主页，也可以像操作中央仓库一样去搜索本地私服中的资源了：

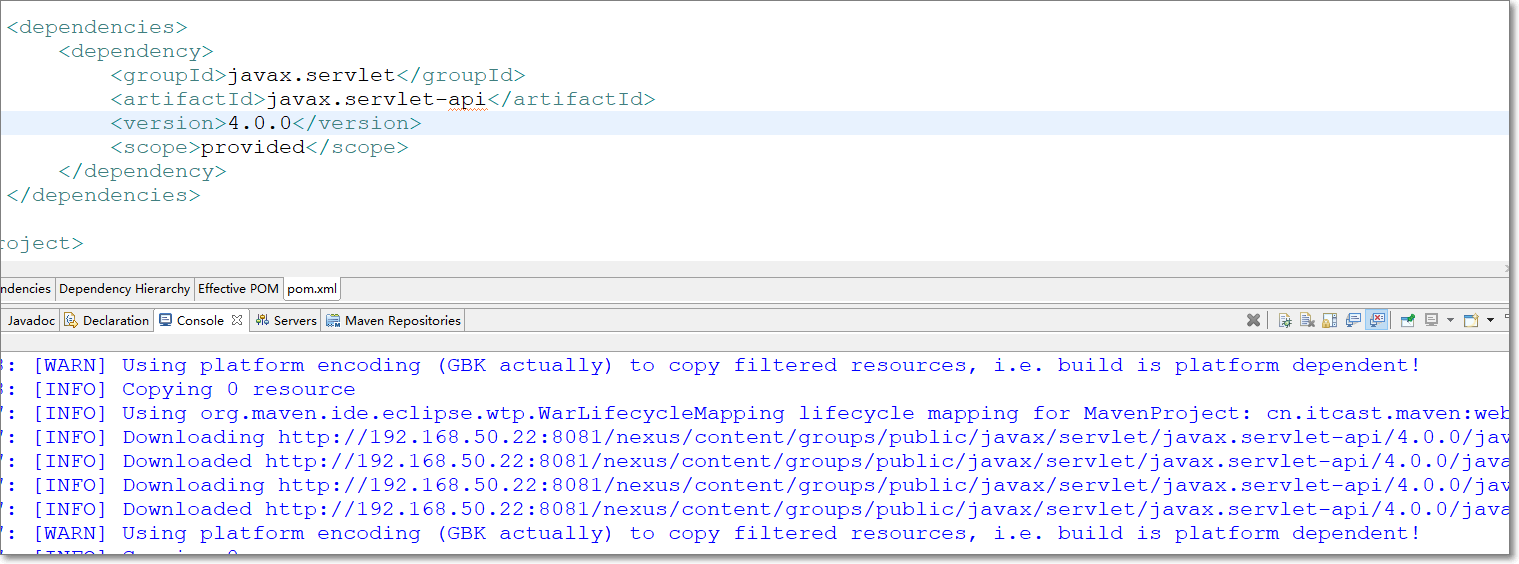




将控制台窗口修改为Maven Console：



更新本地servlet的版本：

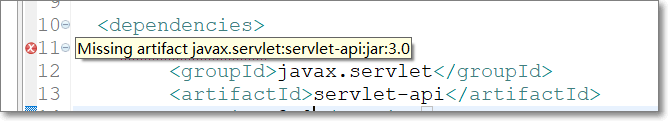


下载地址变成了私服地址。

一旦maven配置了私服,那么以后下载jar包就只会从私服中下载.

### 仓库lastUpdated文件清理（会用即可）

有时候我们更换了版本或者新增Dependency的时候项目会报错：缺失项目



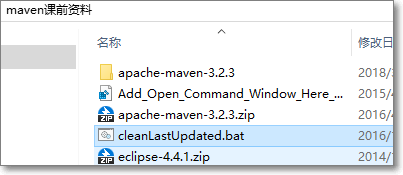
产生错误的原因大致有两种可能:

* + - 1. 中央仓库或者私服仓库中不存在对应版本的项目
      2. 存在项目，但在下载过程中出现网络中断，导致下载了不完整的项目：

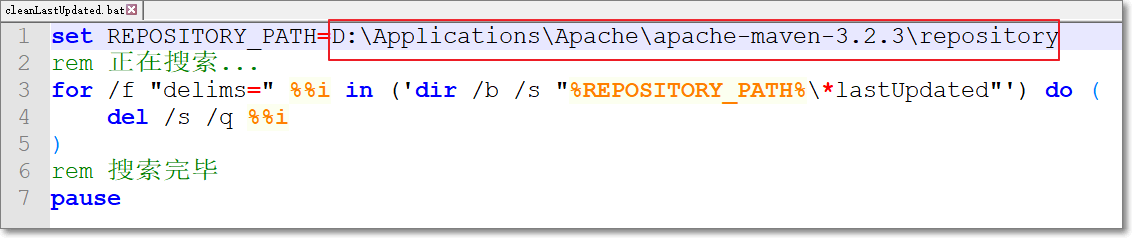
我们找到本地仓库中对应的项目将lastUpdated文件删除后更新项目 maven会自动重新从中央仓库或私服中下载

手动删除文件太幸苦：

在课前资料中提供了删除lastUpdated的批处理文件：



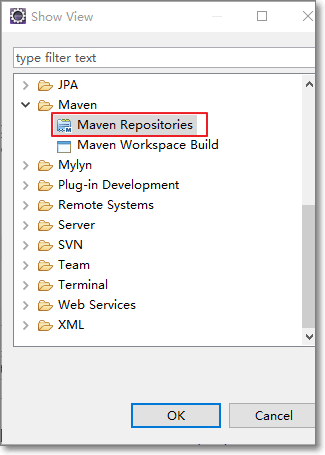
笔记本打开后将仓库地址改为自己的仓库路径



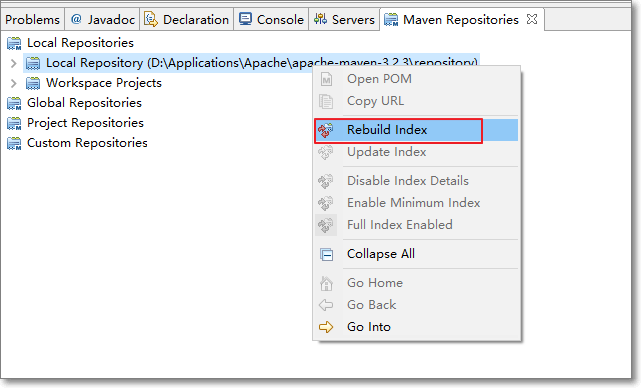
它能帮助我们删除仓库中所有的lastUpdated文件

### Eclipse工具快捷添加依赖

打开Maven Repositories视图：



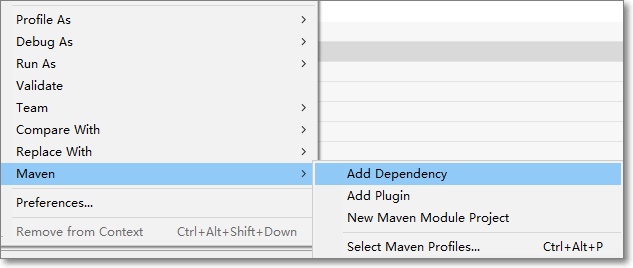
重建本地仓库的索引：



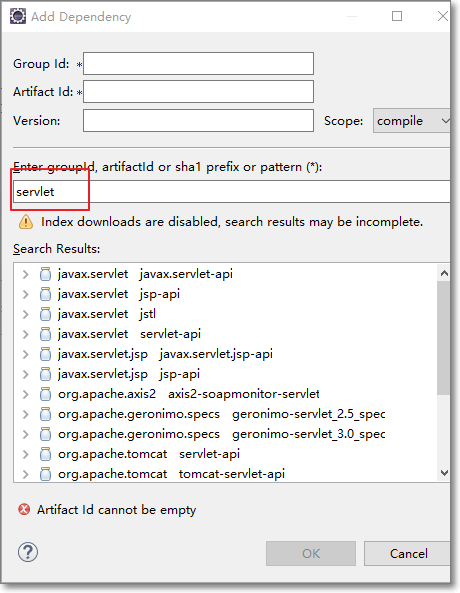
这样做的目的是为了让eclipse工具知道本地仓库中已经拥有哪些jar包

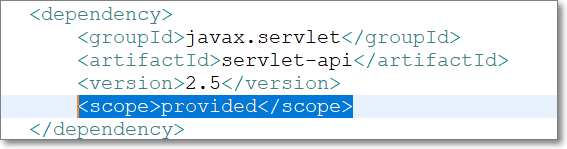
之前我们是手动添加Dependency，在Eclipse中可以通过工具帮我们添加依赖

右击项目的pom.xml空白处或者项目右击：Maven-Add Dependency

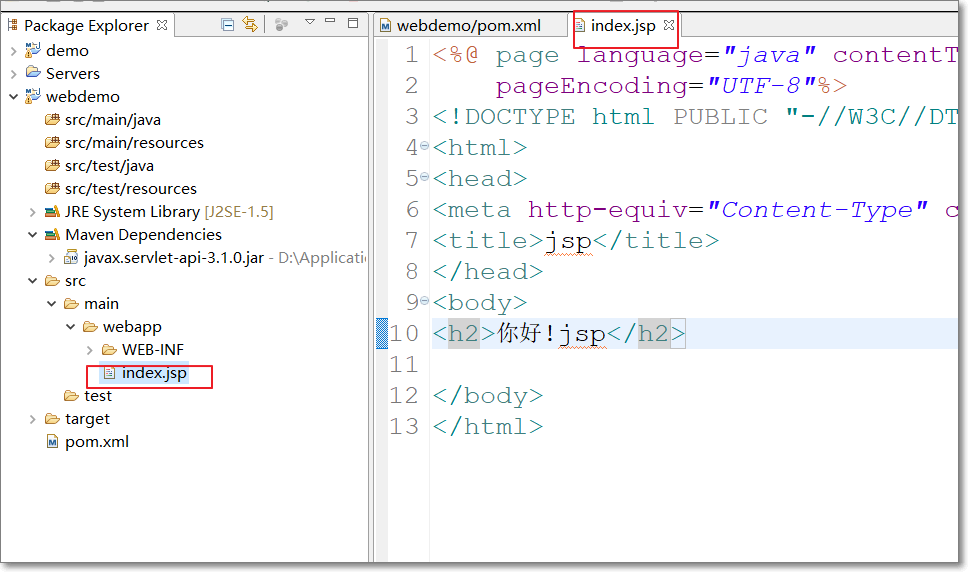


在输入框中搜索想要引入的项目：

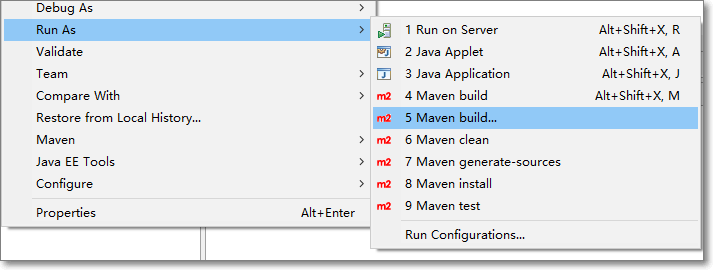


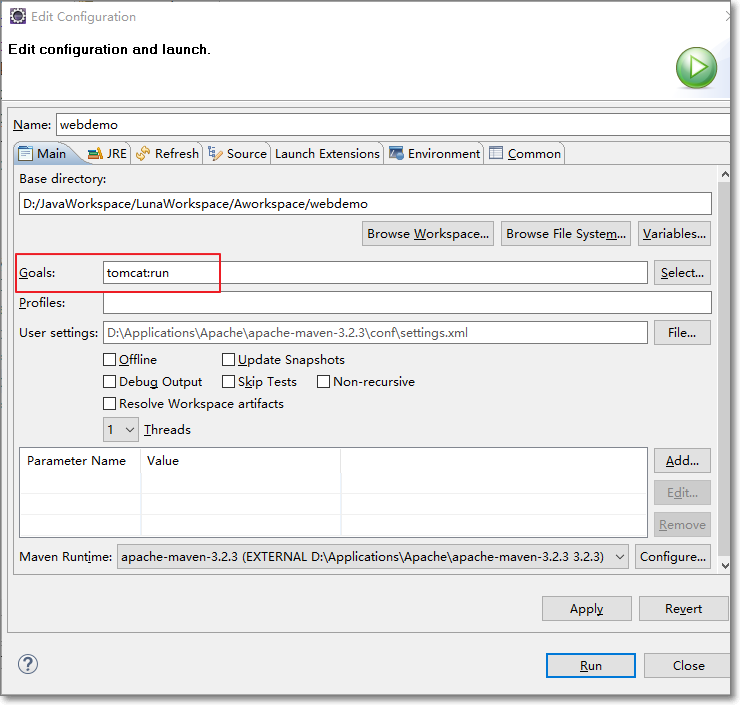


创建jsp页面:

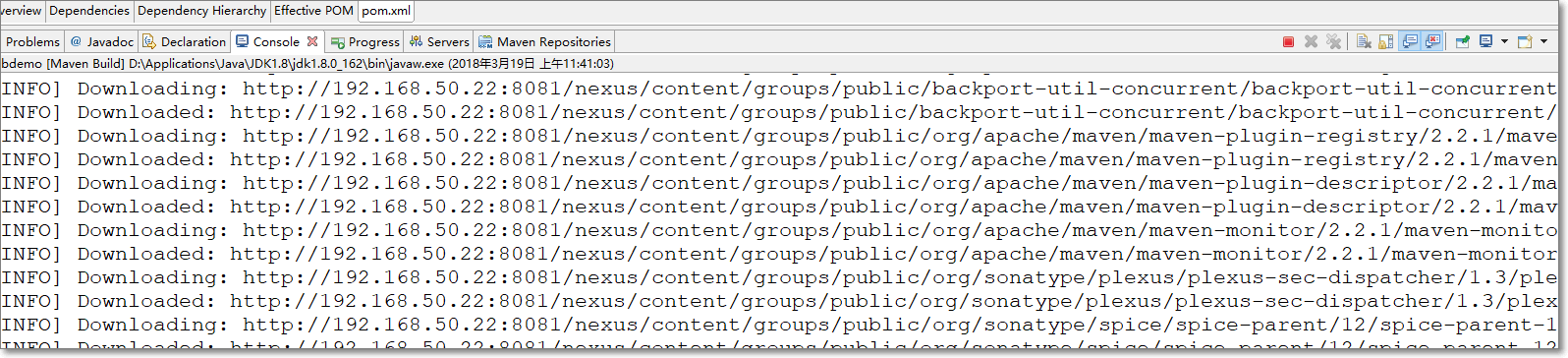


通过Maven内置的Tomcat插件运行项目：

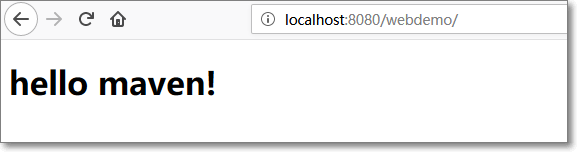




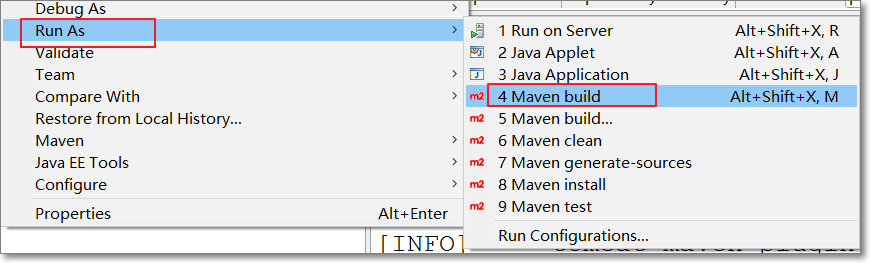
提示:插件也是需要联网下载的，本地没有也会从私服或者中央仓库下载：



访问项目



重新运行项目或者执行maven的上一次命令:



# Jar包的依赖范围(了解)

**compile**：编译依赖范围（默认），使用此依赖范围对于编译、测试、运行三种 classpath 都有效，即在编译、测试和运行的时候都要使用该依赖jar包；

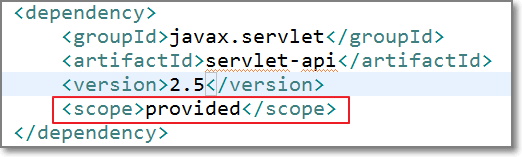
**test**：测试依赖范围，从字面意思就可以知道此依赖范围只能用于测试classpath，而在编译和运行项目时无法使用此类依赖，典型的是JUnit，它只用于编译测试代码和运行测试代码的时候才需要；

**provided**：此依赖范围，对于编译和测试classpath有效，而对运行时无效；

**runtime**：运行时依赖范围，对于测试和运行classpath有效，但是在编译主代码时无效，比如：JDBC驱动；

**system**：系统依赖范围，使用system范围的依赖时必须通过systemPath元素显示地指定依赖文件的路径，不依赖Maven仓库解析，所以可能会造成建构的不可移植，谨慎使用。

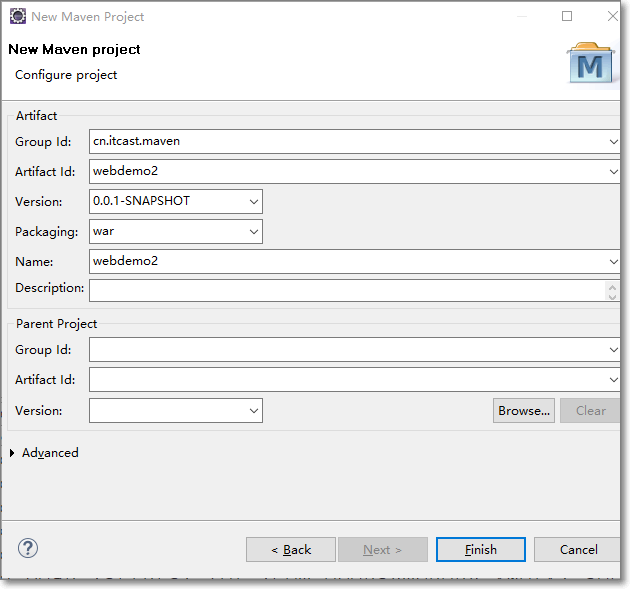
提示：导入servlet-api-xxx.jar,jsp-api-xx.jar依赖范围选择：**provided**



如果设置为compile,在tomcat运行时候需要这些jar , tomcat目录下也有这个jar包，最终出现jar包冲突问题。

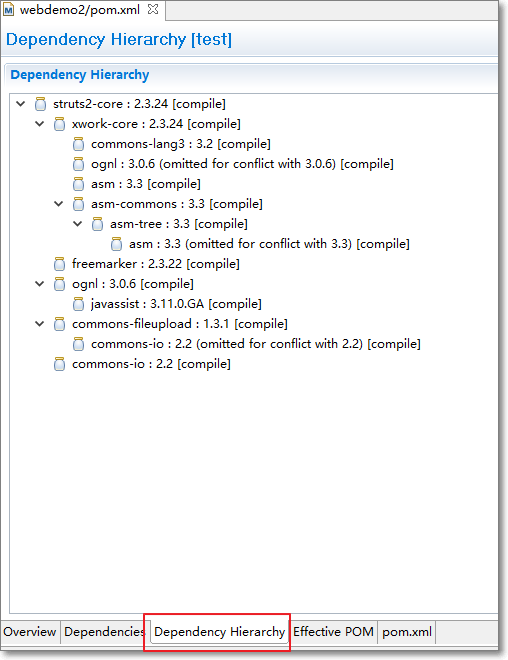
# Maven传递依赖冲突

新建一个Maven web项目：



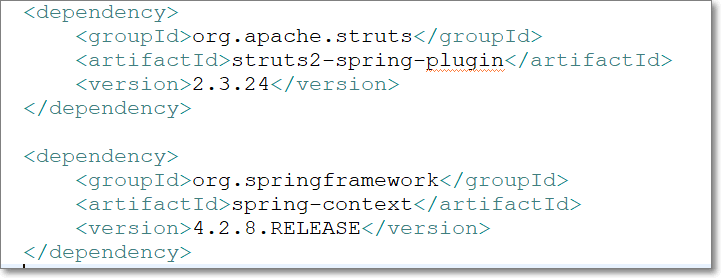
## 依赖传递问题的产生

在pom.xml中我们只配置了一个struts-core的jar包,它会自动将struts2-core包的其他相关依赖jar包进行关联,这种关联就称为依赖传递

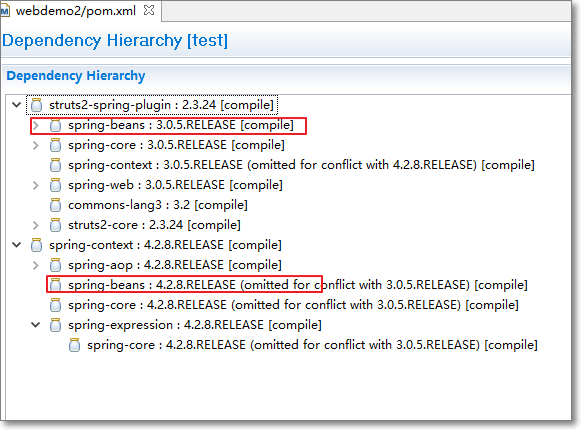


通俗一点来讲， 当前项目A的创建是需要依赖B(struts2-core)项目的，而B项目的创建时需要依赖C(xwork-core)项目的，A项目也就间接的依赖了C项目了

在整合框架时需要配置多个依赖比如:



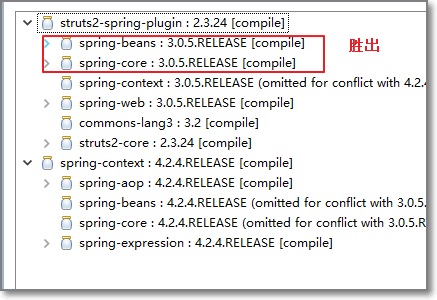
多个依赖传递中出现了相同的依赖时,冲突就产生了:



## Maven自身的冲突调节原则

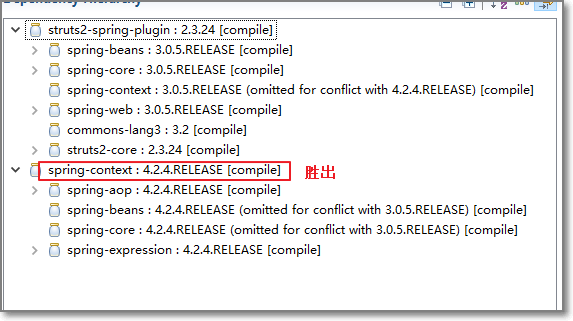
### 第一声明者优先原则

根据依赖的配置顺序,谁先声明就用谁的依赖:



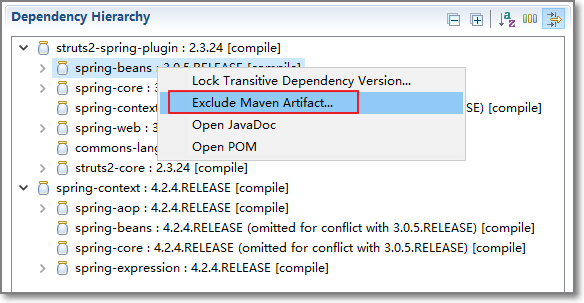
### 路径近者优先原则

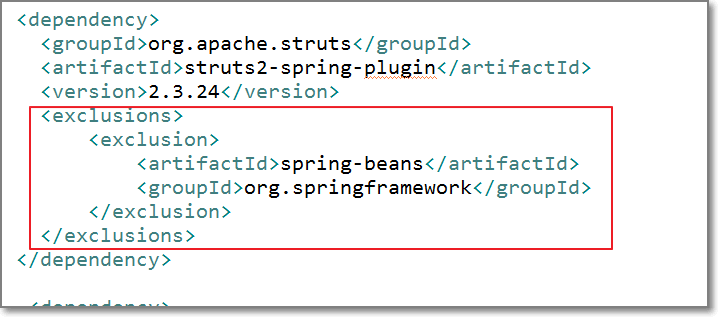
直接依赖的级别要比传递依赖优先:



## 排除依赖

将重复的依赖通过exclude Maven Artfact... 排除解决,只保留唯一依赖:





## 版本绑定(推荐)

通过配置dependencyManagement标签指定使用依赖的版本:

<!-- 版本锁定 -->

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-spring-plugin</artifactId>

<version>2.3.24</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

</dependencies>

</dependencyManagement>

注意: 版本锁定只是锁定了依赖的版本,并没有导入jar包.

使用常量管理依赖的版本信息,方便框架库的版本更替:

<!-- 版本信息统一管理 -->

<properties>

<spring-version>4.2.4.RELEASE</spring-version>

</properties>

<!-- 版本锁定 -->

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.apache.struts</groupId>

<artifactId>struts2-spring-plugin</artifactId>

<version>2.3.24</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring-version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>${spring-version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring-version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring-version}</version>

</dependency>

</dependencies>

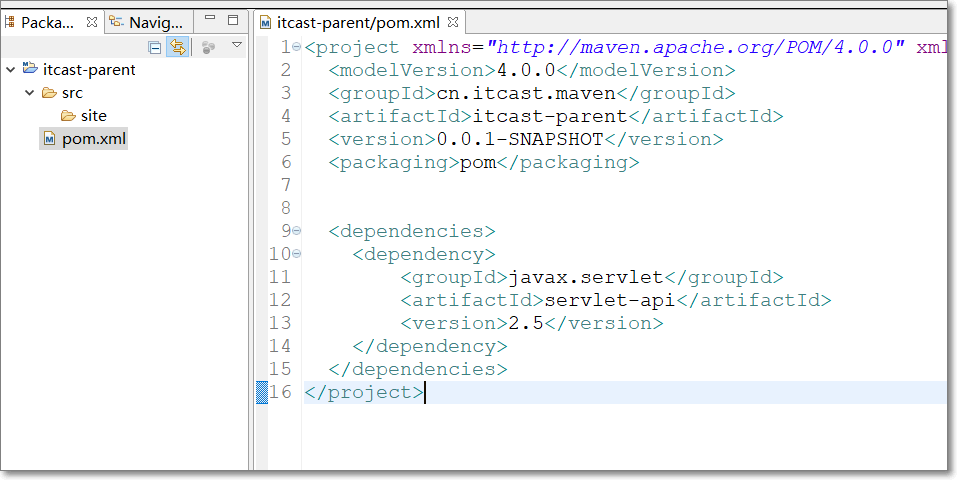
</dependencyManagement>

# Maven项目的继承

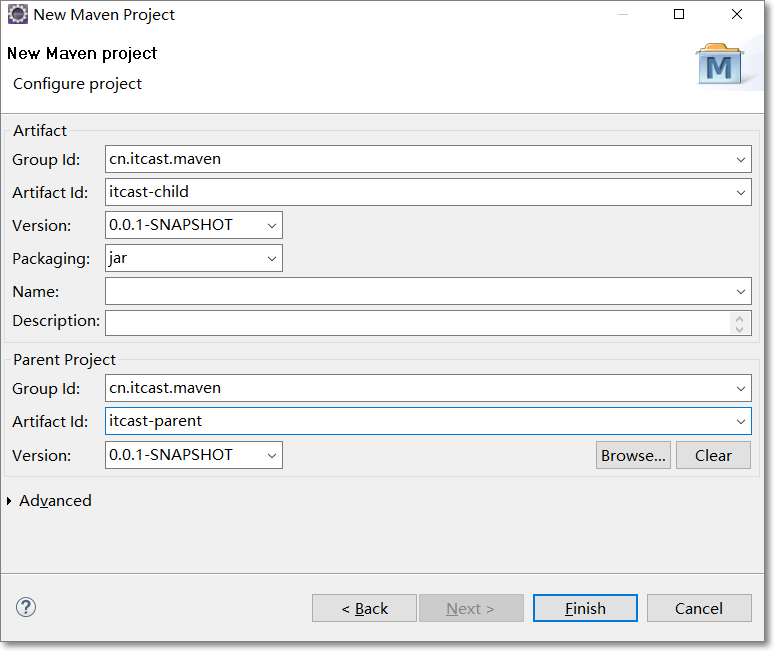
大家都知道,java语言中,当多个类中代码存在重复时,可以通过抽取共性编写一个父类,由子类继承后获取父类的功能

实际上,maven工程与maven工程之间也可以实现这样的关系

第一步: 创建父工程并添加依赖



第二步:创建maven工程并填入父工程GAV:



# 总结

1. Maven的配置安装
2. 通过maven创建项目
3. 理解项目的生命周期
4. Maven的常用命令(记忆)
5. 掌握通过pom引入jar依赖
6. 了解jar依赖范围
7. 掌握传递依赖冲突的解决和版本绑定