

# 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

## 1. Maven 的概述和技术简介

### 1.1. 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\课前资料\参考图书				
名称	修改日期	类型	大小	
Maven权威指南.pdf	2013/3/28 7:45	Adobe Acrobat ...	6,746 KB	
Maven实战.pdf	2013/12/27 14:50	Adobe Acrobat ...	40,072 KB	

Maven 管理项目的周期:



## 2. Maven 的快速入门

### 2.1. Maven 的下载安装

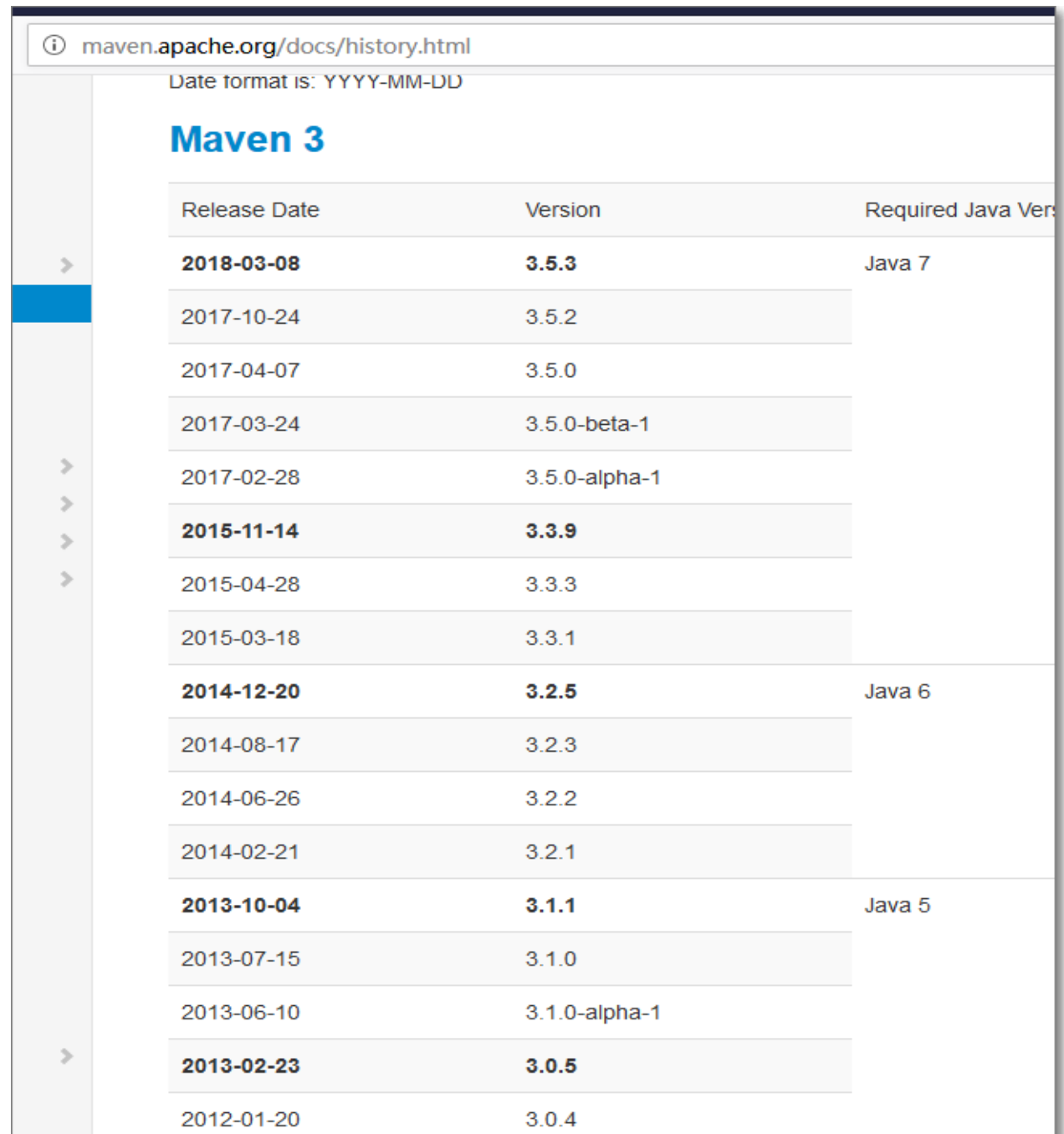
官方网站: <http://maven.apache.org/>

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

官网的系列版本:

maven.apache.org/download.cgi	
In order to guard against corrupted downloads/installations, it is highly recommended to verify the checksums of the downloaded files.	
	Link
Binary tar.gz archive	<a href="#">apache-maven-3.5.3-bin.tar.gz</a>
Binary zip archive	<a href="#">apache-maven-3.5.3-bin.zip</a>
Source tar.gz archive	<a href="#">apache-maven-3.5.3-src.tar.gz</a>
Source zip archive	<a href="#">apache-maven-3.5.3-src.zip</a>

版本选择问题:



① maven.apache.org/docs/history.html

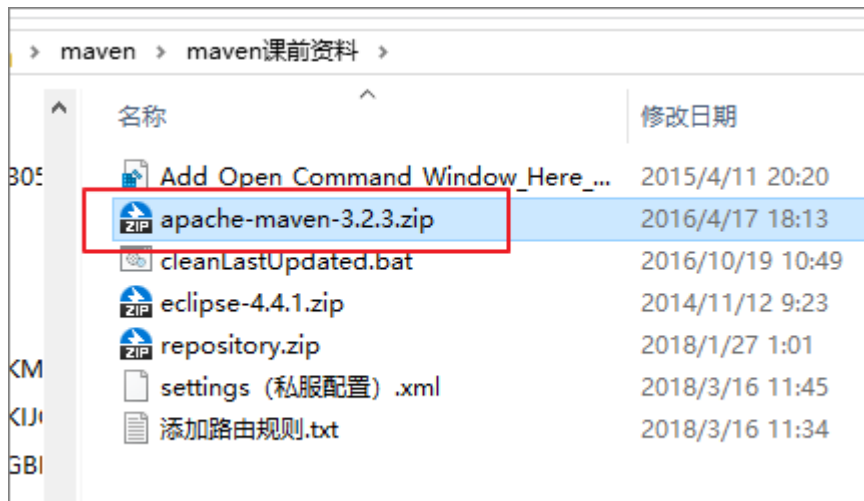
Date format is: YYYY-MM-DD

## Maven 3

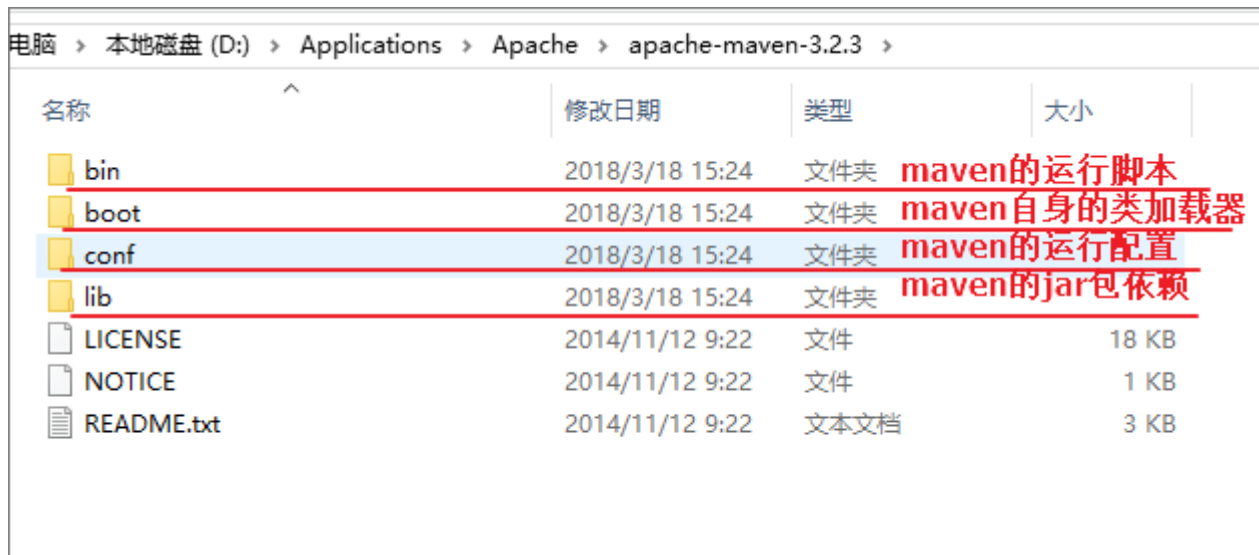
Release Date	Version	Required Java Version
<b>2018-03-08</b>	<b>3.5.3</b>	Java 7
2017-10-24	3.5.2	
2017-04-07	3.5.0	
2017-03-24	3.5.0-beta-1	
2017-02-28	3.5.0-alpha-1	
<b>2015-11-14</b>	<b>3.3.9</b>	
2015-04-28	3.3.3	
2015-03-18	3.3.1	
<b>2014-12-20</b>	<b>3.2.5</b>	Java 6
2014-08-17	3.2.3	
2014-06-26	3.2.2	
2014-02-21	3.2.1	
<b>2013-10-04</b>	<b>3.1.1</b>	Java 5
2013-07-15	3.1.0	
2013-06-10	3.1.0-alpha-1	
<b>2013-02-23</b>	<b>3.0.5</b>	
2012-01-20	3.0.4	

本课程选用 3.2.3

Windows 环境请下载 maven zip 包:



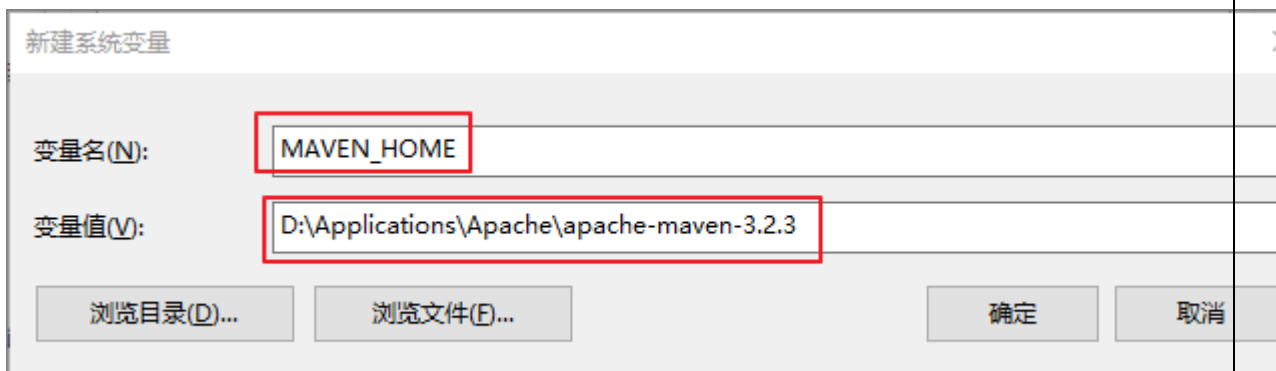
解压 maven 的 zip 包（注意解压的路径，建议不要有中文、空格、特 e 符）



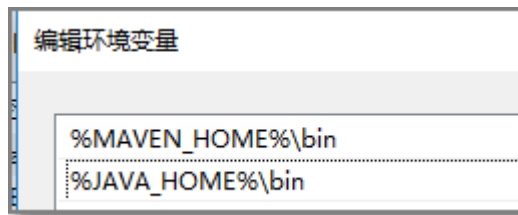
## 2.2. Maven 的配置

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：

```
C:\Users\Administrator>mvn -v
Apache Maven 3.2.3 (33f8c3e1027c3ddde99d3cdebad2656a31e8fdf4; 2014-
Maven home: D:\Applications\Apache\apache-maven-3.2.3\bin\..
Java version: 1.8.0_162, vendor: Oracle Corporation
Java home: D:\Applications\Java\JDK1.8\jdk1.8.0_162\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos
C:\Users\Administrator>
```

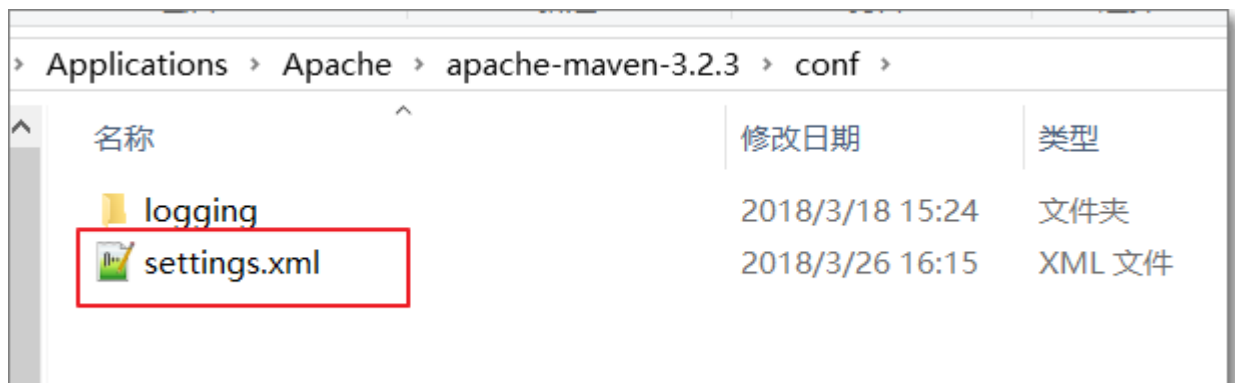
## 2.3. Maven 仓库配置

什么是仓库？

首先大家要清楚，jar 包或者 war 包其实就是一个 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：

```

49 <!-- localRepository
50 | The path to the local repository maven will use to store artifacts.
51 |
52 | Default: ${user.home}/.m2/repository
53 <localRepository>/path/to/local/repo</localRepository>
54 -->

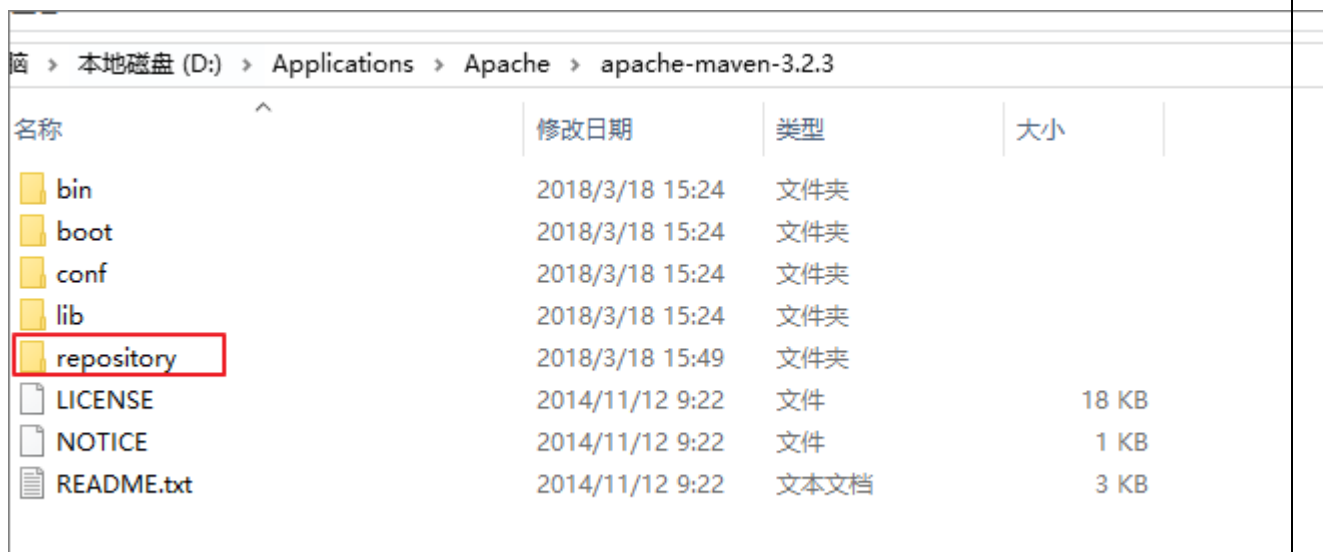
```

发现：默认的仓库位置是在当前登录用户的 home 目录下：

`<localRepository>` 仓库的存放位置 `</localRepository>`

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：

```

<!-- localRepository
| The path to the local repository maven will use to store artifacts.
|
| Default: ${user.home}/.m2/repository
<localRepository>/path/to/local/repo</localRepository>
-->

<localRepository>D:\Applications\Apache\apache-maven-3.2.3\repository</localRepository>

```

## 2.4. 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：

此电脑 > 本地磁盘 (D:) > Applications > Apache > apache-maven-3.2.3 > repository

名称	修改日期	类型	大小
.cache	2018/1/26 17:58	文件夹	
.locks	2018/3/18 15:57	文件夹	
activation	2018/1/26 17:58	文件夹	
ant	2018/1/26 17:58	文件夹	
antlr	2018/1/26 17:58	文件夹	
aopalliance	2018/1/26 17:58	文件夹	
apache-log4j	2018/1/26 17:58	文件夹	
asm	2018/3/18 15:57	文件夹	
avalon-framework	2018/1/26 17:58	文件夹	
axis	2018/3/18 15:57	文件夹	
backport-util-concurrent	2018/1/26 17:58	文件夹	
bcel	2018/1/26 17:58	文件夹	
biz	2018/1/26 17:58	文件夹	
bouncycastle	2018/3/18 15:57	文件夹	
bpm	2018/1/26 17:58	文件夹	
bsf	2018/1/26 17:58	文件夹	
c3p0	2018/1/26 17:58	文件夹	
cglib	2018/3/18 15:57	文件夹	
ch	2018/1/26 17:58	文件夹	
classworlds	2018/1/26 17:58	文件夹	
cobertura	2018/1/26 17:58	文件夹	
com	2018/3/18 15:57	文件夹	
commons-beanutils	2018/1/26 17:58	文件夹	
commons-chain	2018/1/26 17:58	文件夹	
commons-cli	2018/1/26 17:58	文件夹	
commons-coder	2018/1/26 17:58	文件夹	

注意：注意解压文件的目录层次！

## 2.5. 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:Project 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

## 2.6. 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)

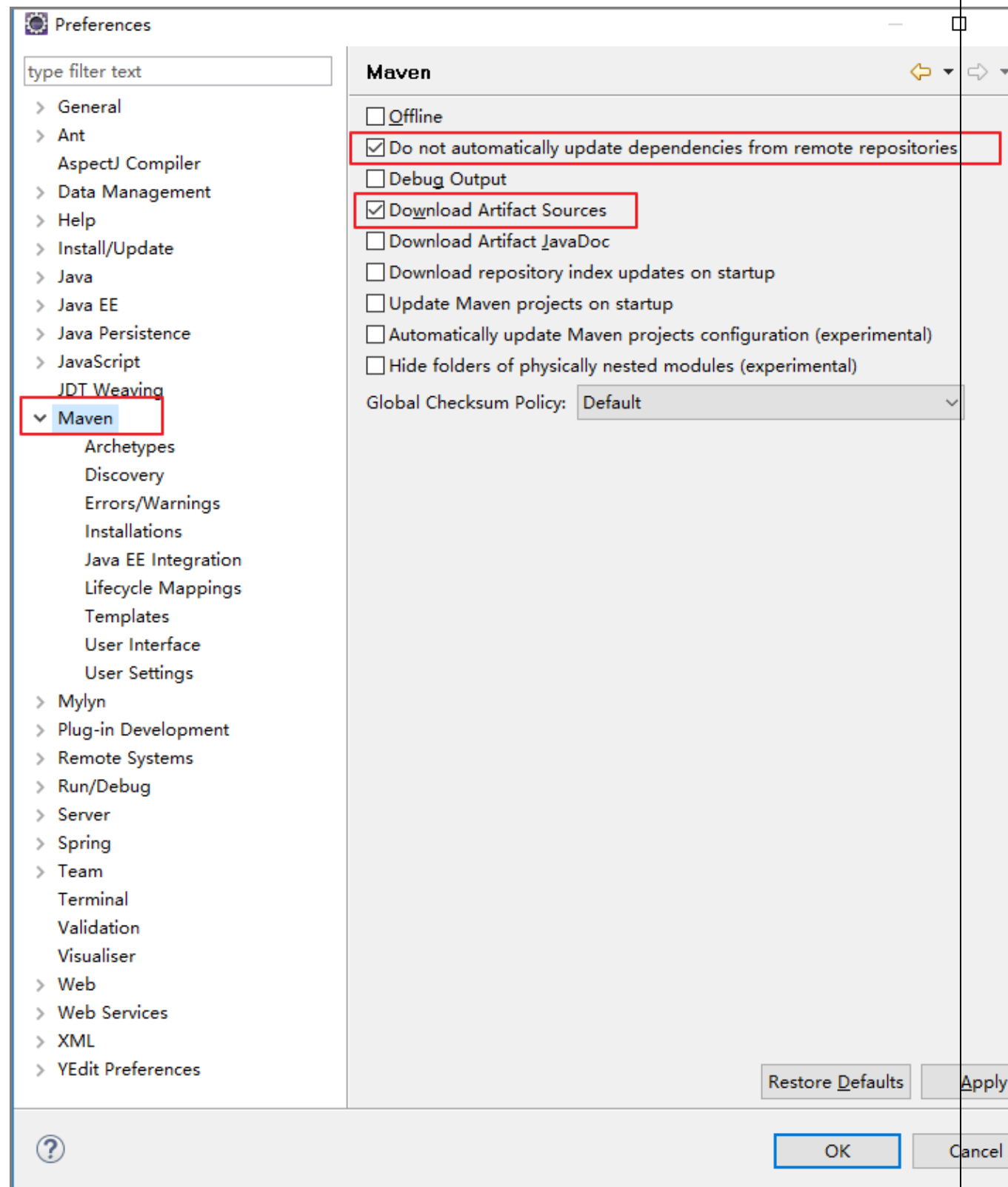
提示：生命周期从上到下执行，如果执行后面命令，自动执行之前项目构建步骤。

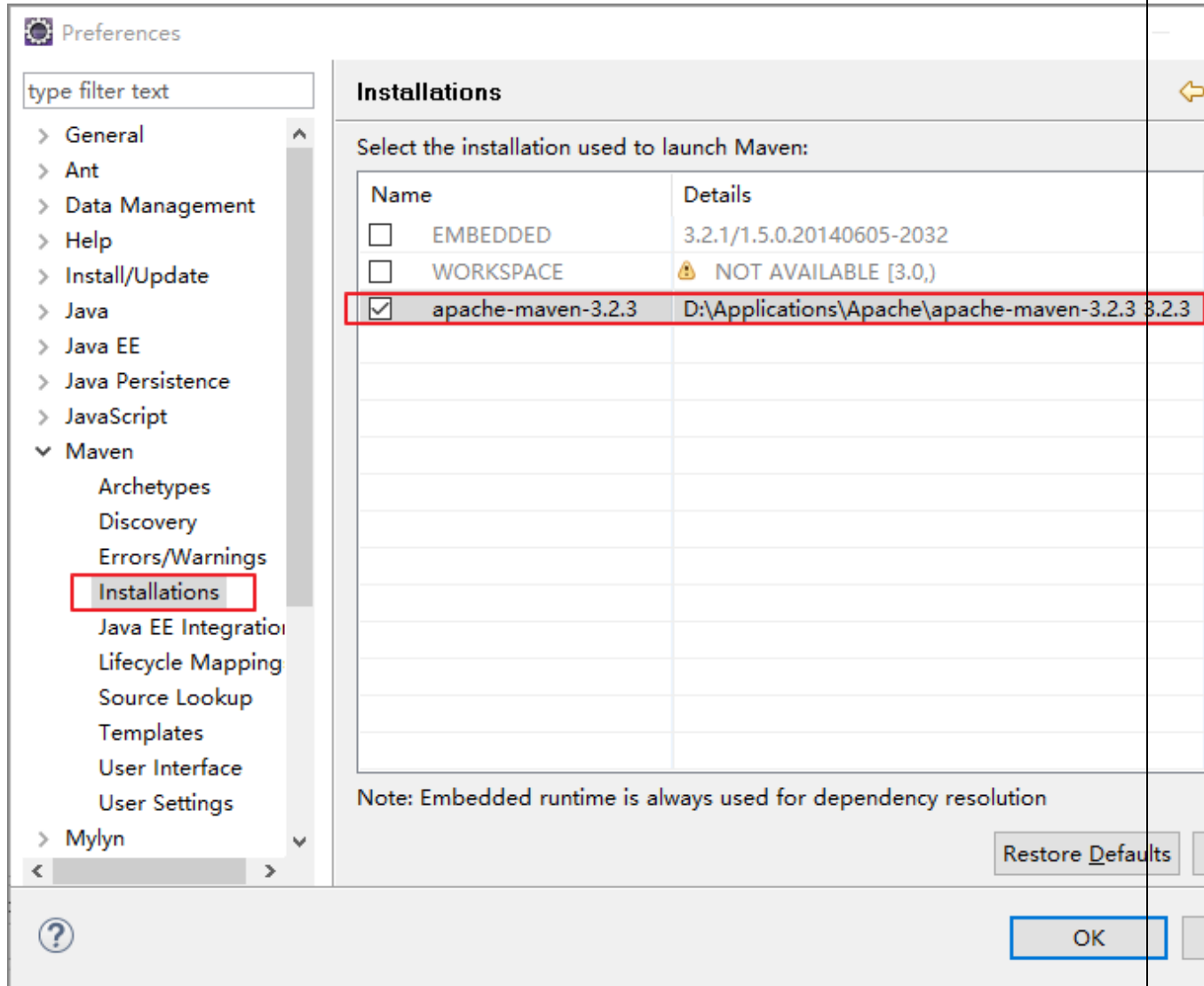
## 3. 通过 Eclipse 工具配置 MAVEN 构建工程项目

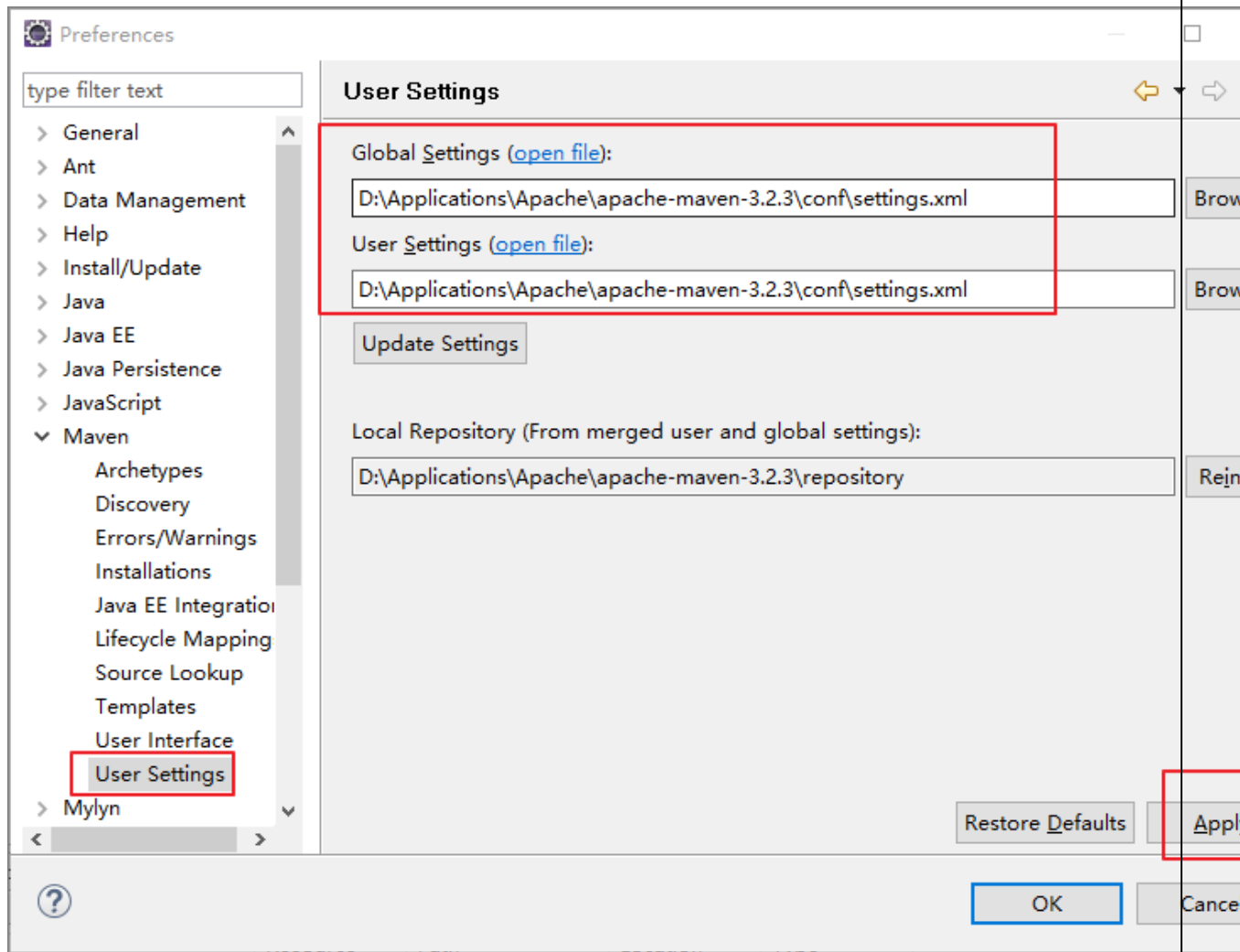
### 3.1. 在 Eclipse 工具集成 maven 配置

window -preferences:





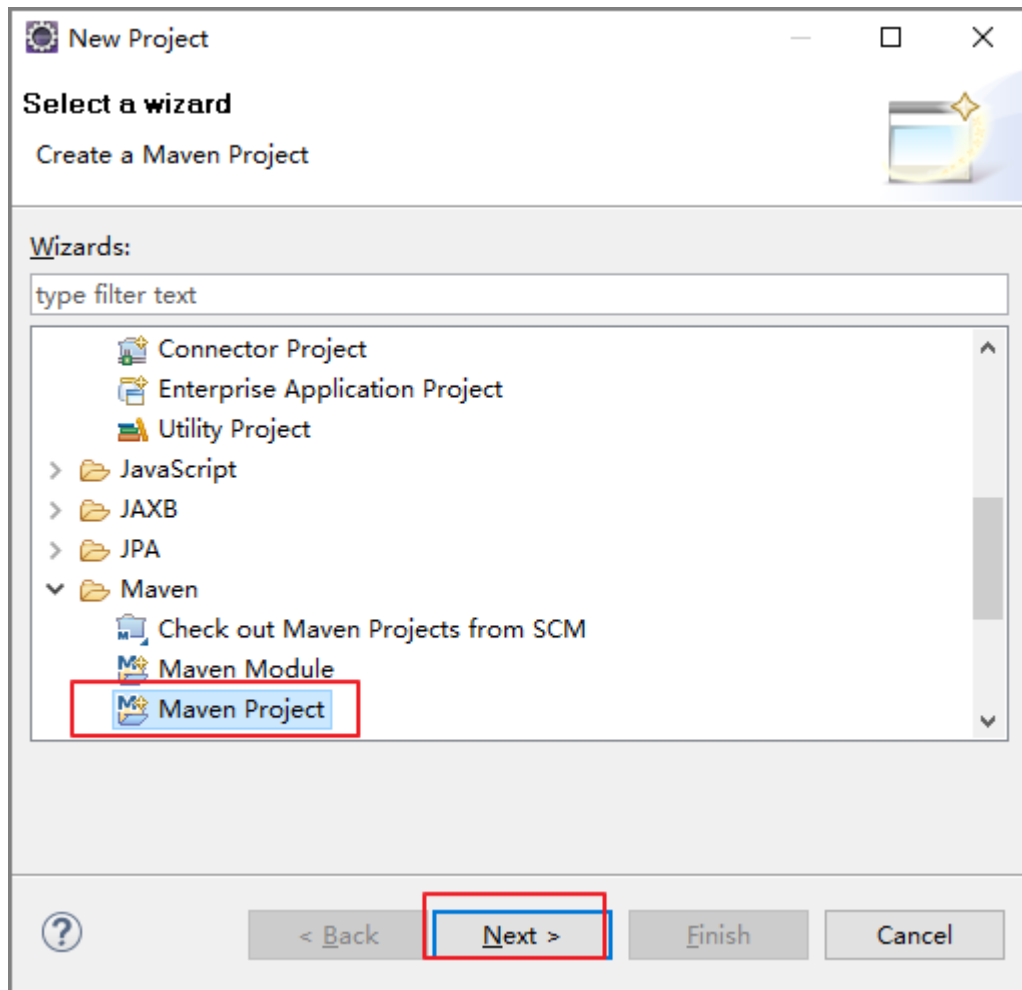




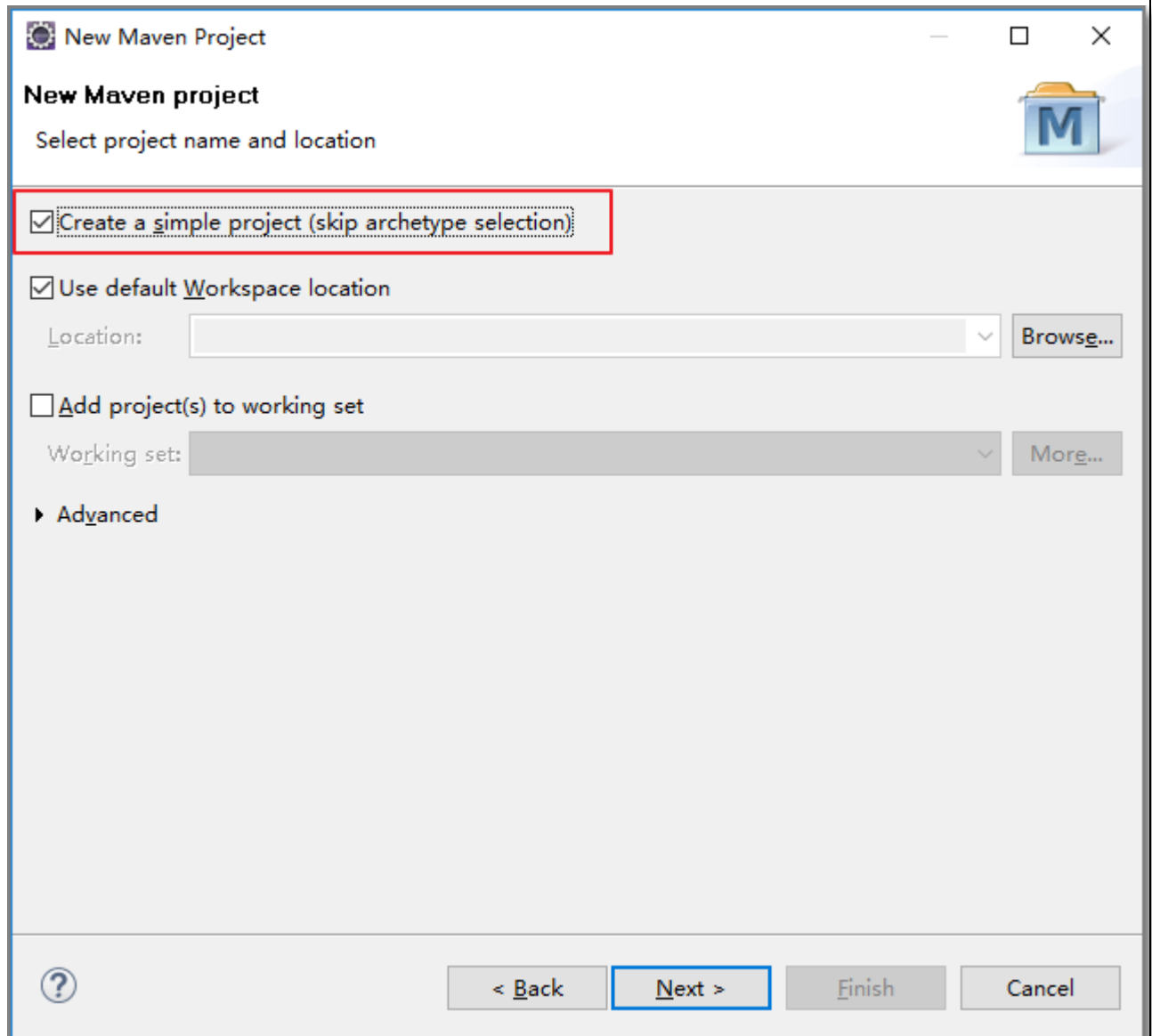
## 3.2. 构建工程项目

### 3.2.1. 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：

New Maven Project

New Maven project

Configure project

Artifact

Group Id: com.itheima.maven 公司名称

Artifact Id: mavendemo 项目名称

Version: 0.0.1-SNAPSHOT

Packaging: jar jar: java项目  
war: web项目  
pom: 父工程

Name: mavendemo

Description:

Parent Project

Group Id:

Artifact Id:

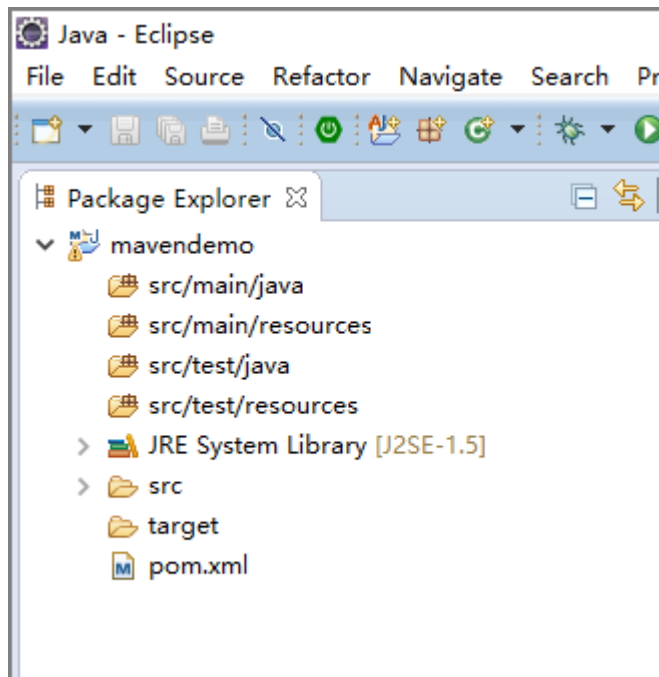
Version: Browse... Clear

Advanced

? < Back Next > Finish Cancel

点击 finish 完成工程创建。

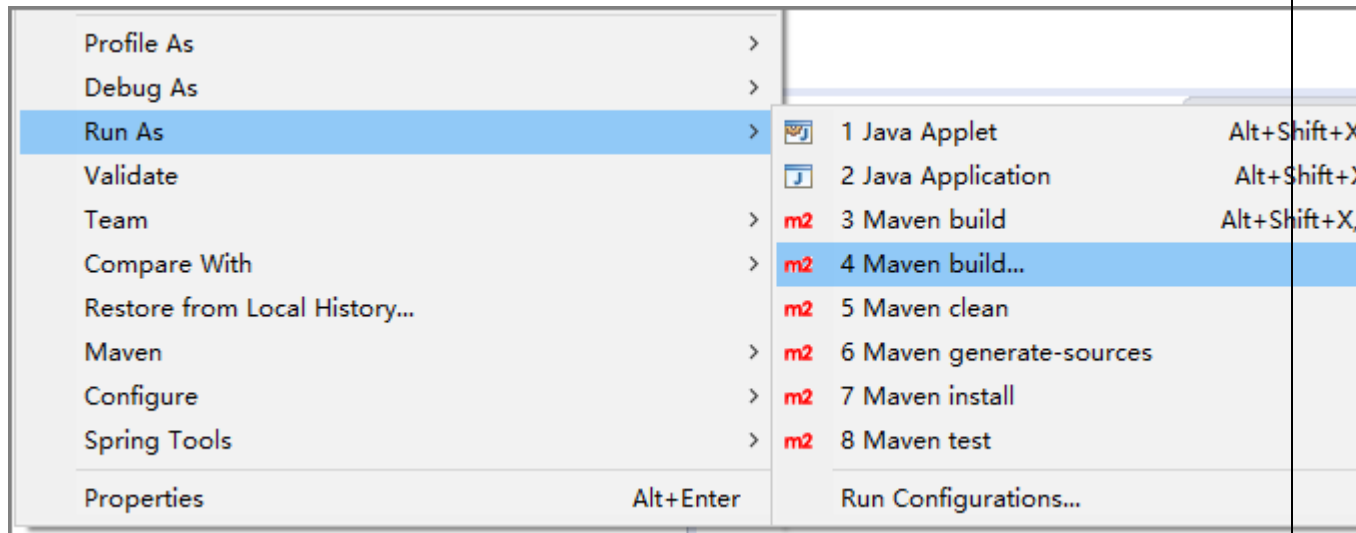
建立起来的工程目录结果—简单的 java 项目结构：



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

```
javademo/pom.xml  ⓘ
1 <project xmlns="http://maven.apache.org/P
2   <modelVersion>4.0.0</modelVersion>
3   <groupId>cn.itcast.maven</groupId>
4   <artifactId>javademo</artifactId>
5   <version>0.0.1-SNAPSHOT</version>
6 </project>
```

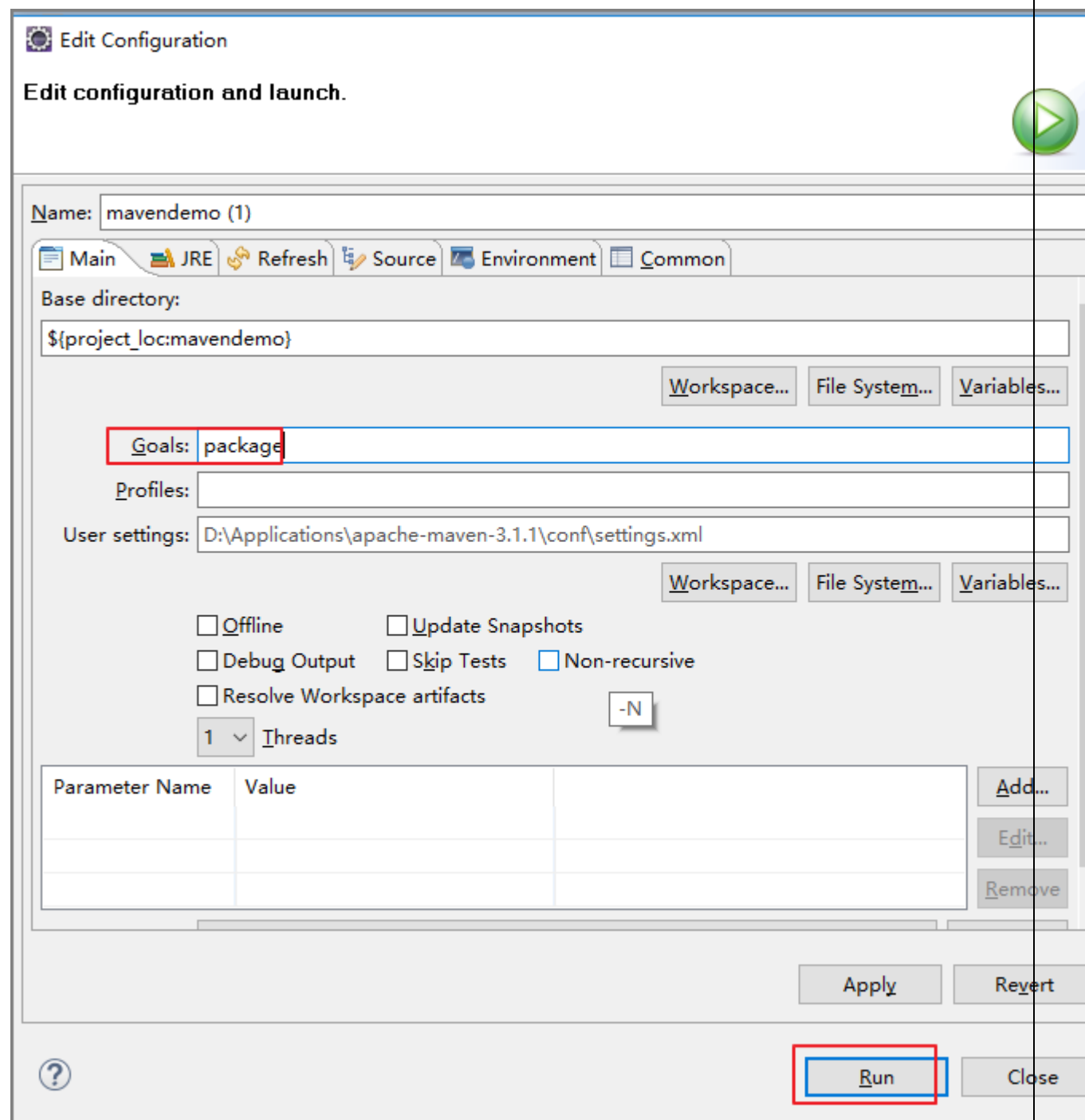
右键项目 run as 查看能够使用 maven 命令



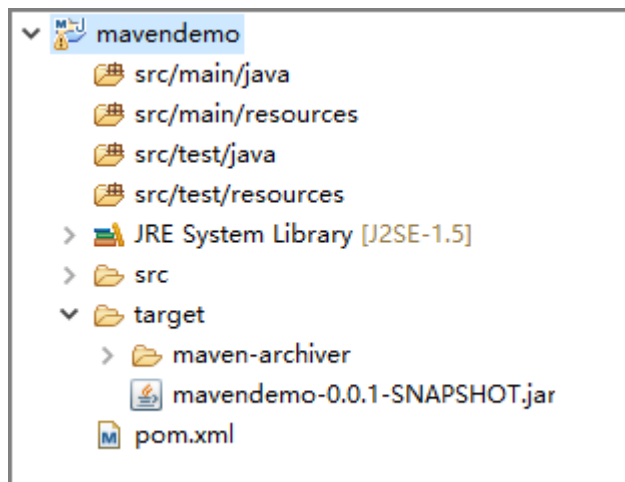
通过手动输入命令，来执行 `mvn package` 命令：

`Package` 命令会将当前项目打成 `jar` 包放置在 `target` 目录下

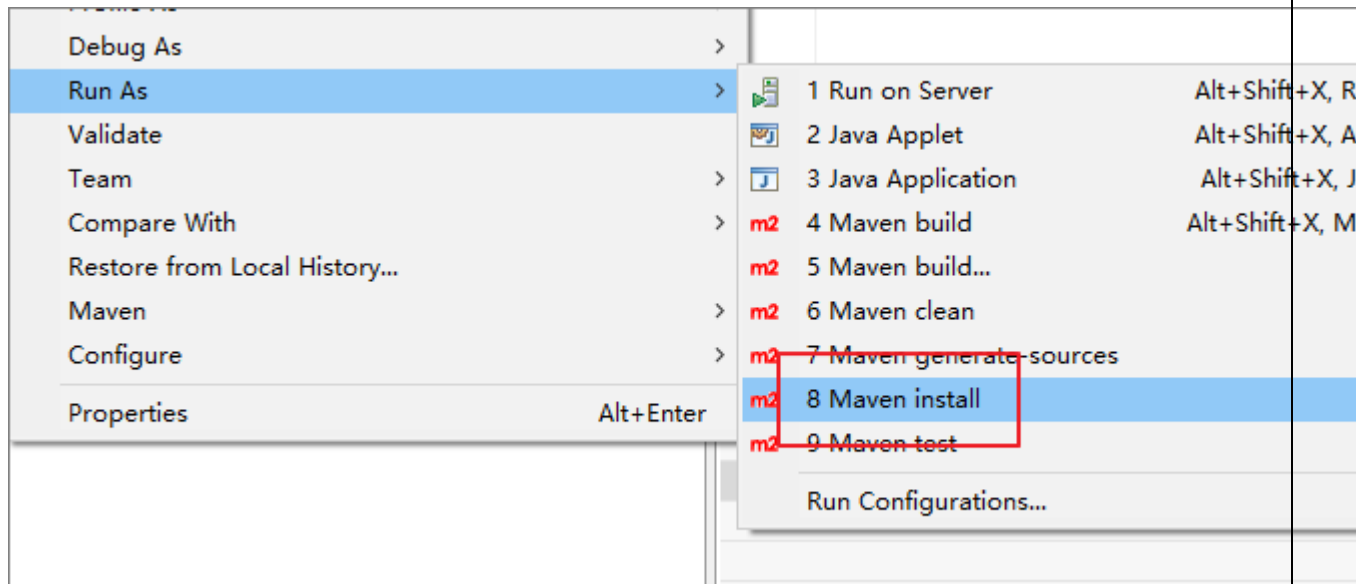




查看打出来的 jar:



通过 `install` 命令 将打好包的项目安装到本地仓库中：



```

[INFO] --- maven-compiler-plugin:3.1:testCompile (de
[INFO] Nothing to compile - all classes are up to da
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @
[INFO] Building jar: D:\JavaWorkspace\LunaWorkspace\
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default
[INFO] Installing D:\JavaWorkspace\LunaWorkspace\Awo
[INFO] Installing D:\JavaWorkspace\LunaWorkspace\Awo
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.008 s
[INFO] Finished at: 2018-03-18T16:33:35+08:00
[INFO] Final Memory: 10M/155M
[INFO] -----

```

che-maven-3.2.3 > repository > cn > itcast > maven > javademo > 0.0.1-SNAPSHOT				
名称	G	修改日期	A 类型	· V 大小
_remote.repositories		2018/3/18 16:33	REPOSITORIES ...	1 KB
javademo-0.0.1-SNAPSHOT.jar		2018/3/18 16:33	Executable Jar File	2 KB
javademo-0.0.1-SNAPSHOT.pom		2018/3/18 16:23	POM 文件	1 KB
maven-metadata-local.xml		2018/3/18 16:33	XML 文档	1 KB

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

### 3.2.2. 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

New Maven Project

New Maven project

Configure project

Artifact

Group Id: com.itheima.maven

Artifact Id: webdemo

Version: 0.0.1-SNAPSHOT

Packaging: war

Name:

Description:

Parent Project

Group Id:

Artifact Id:

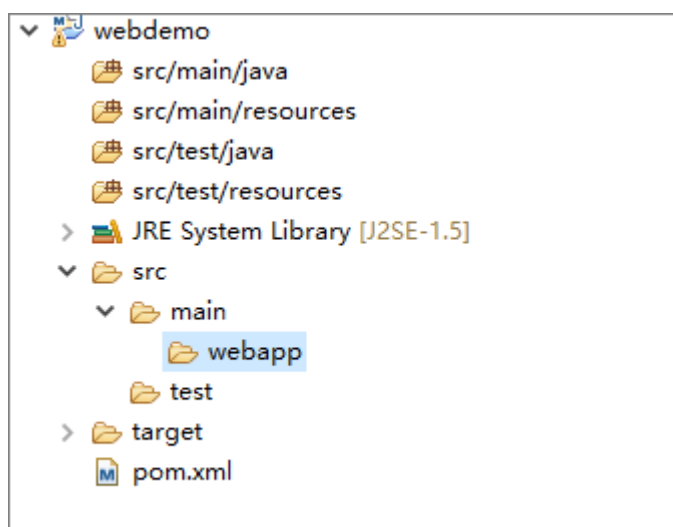
Version:

Browse... Clear

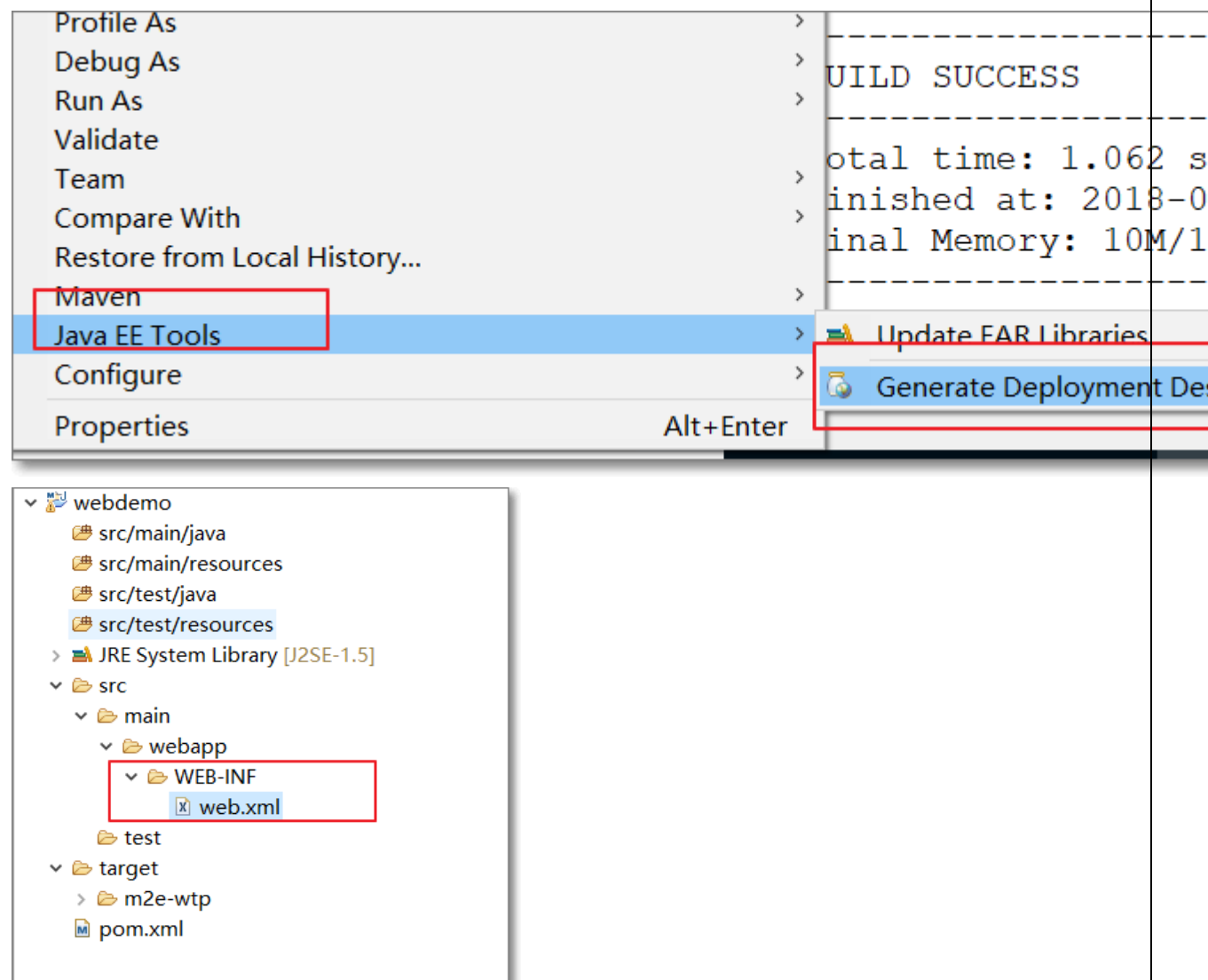
Advanced

? < Back Next > Finish Cancel

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



## 4. 通过 maven 工程去引入依赖

通过 maven 引入依赖的两种方式:

### 4.1. 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/>

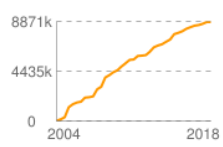
访问 maven 的中央仓库:

← → ↻ 🏠 mvnrepository.com

# MVNREPOSITORY

Search for groups, artifacts, categories


**Indexed Artifacts (8.88M)**





**Popular Categories**


- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities

## What's New in Maven

**React4j Core Binding**  
[org.realityforge.react4j](#) » [react4j-core](#) » 0.52  
React4j core binding  
Last Release on Mar 18, 2018

**WildFly Elytron**  
[org.wildfly.security](#) » [wildfly-elytron](#) » 1.1.9.Final  
WildFly Security SPIs  
Last Release on Mar 16, 2018

**WildFly Elytron**  
[org.wildfly.security](#) » [wildfly-elytron](#) » 1.2.4.Final  
WildFly Security SPIs  
Last Release on Mar 16, 2018

**WildFly Common**  
[org.wildfly.common](#) » [wildfly-common](#) » 1.3.1.Final  
WildFly Common


搜索想要的 jar 包依赖:


mvnrepository.com/search?q=servlet


servlet Search

**Found 1765 results**

Sort: **relevance** | popular | newest

**1. Java Servlet API**  
[javax.servlet](#) » [javax.servlet-api](#)  
Java Servlet API  
Last Release on Apr 20, 2018

**2. JavaServlet(TM) Specification**  
[javax.servlet](#) » [servlet-api](#)  
JavaServlet(TM) Specification  
Last Release on Apr 17, 2008

**3. Jetty :: Servlet Handling**  
[org.eclipse.jetty](#) » [jetty-servlet](#)  
Jetty Servlet Container  
Last Release on May 3, 2018

选择需要版本并拷贝 Dependency:

Home » javax.servlet » servlet-api » 2.5

**Note:** There is a new version for this artifact

New Version	3.0-alpha-1
-------------	-------------

 **JavaServlet(TM) Specification » 2.5**  
JavaServlet(TM) Specification

License	CDDL GPL 2.0
Categories	Java Specifications
Date	(Jul 17, 2006)
Files	<a href="#">pom (157 bytes)</a> <a href="#">jar (102 KB)</a> <a href="#">View All</a>
Repositories	<a href="#">Central</a> <a href="#">Adobe</a> <a href="#">Java.net Releases</a> <a href="#">Redhat GA</a>
Used By	9,286 artifacts

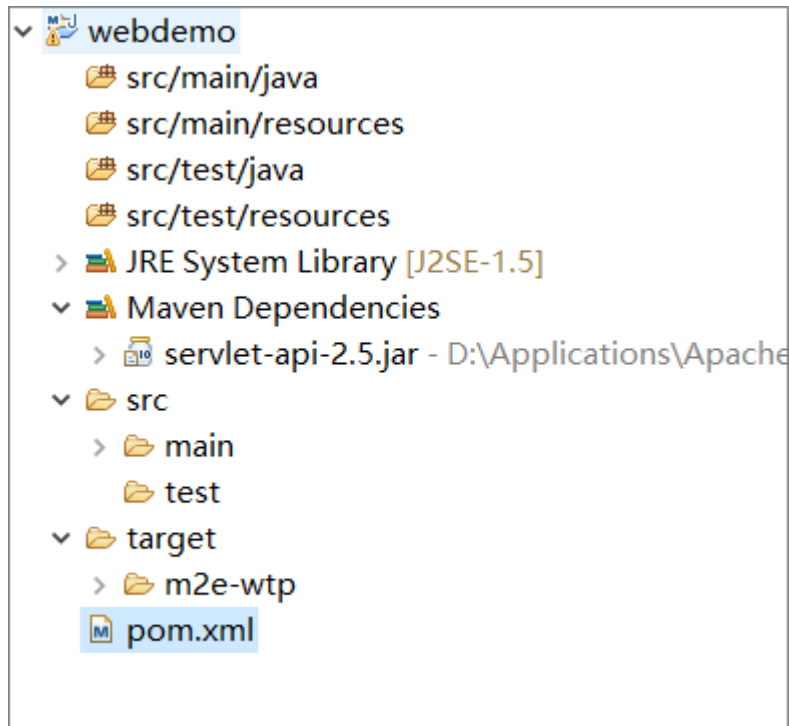
[Maven](#) [Gradle](#) [SBT](#) [Ivy](#) [Grape](#) [Leiningen](#) [Buildr](#)

```
<!-- 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>
```

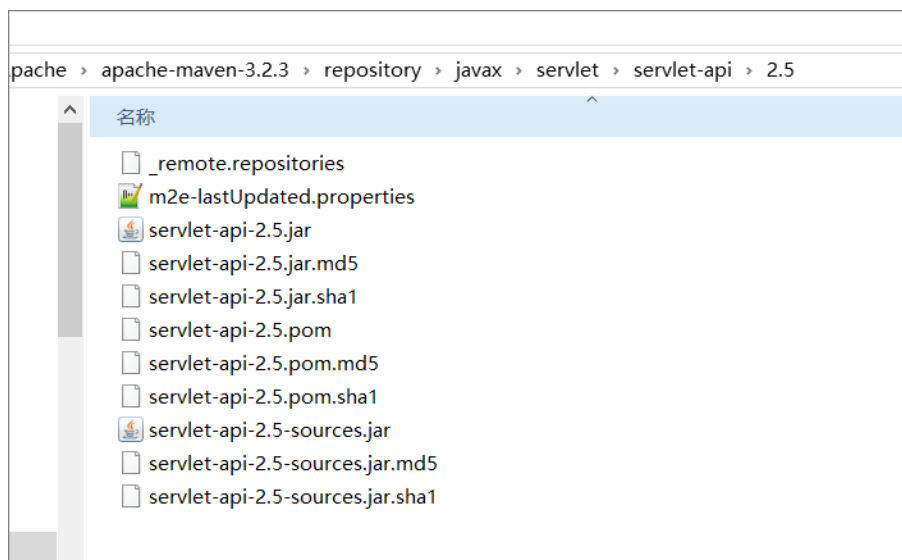
在本地项目的 pom.xml 文件中添加 dependencies 标签并将拷贝的 Dependency 粘贴进来:

```
<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：



## 4.2. 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/>

### 4.2.1. 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

```
D:\Desktop>route add 192.168.50.0 mask 255.255.255.0 192.168.15.1 -p
操作完成!
```

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)

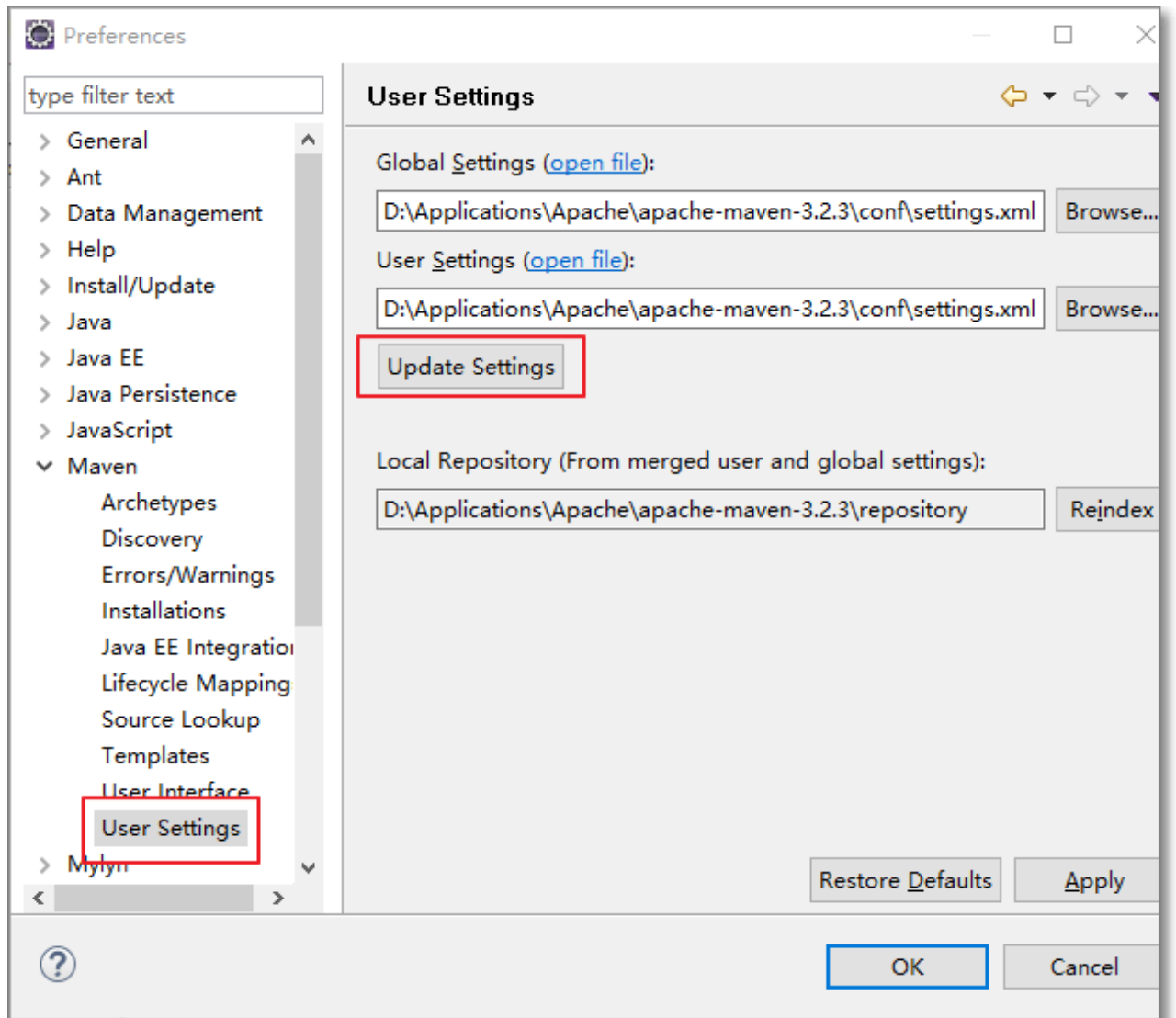
### 4.2.2. settings.xml 文件添加私服配置

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)

```
<mirrors>
  <!-- mirror
  | Specifies a repository mirror site to use instead of a given repository
  | this mirror serves has an ID that matches the mirrorOf element of this
  | for inheritance and direct lookup purposes, and must be unique across
  | the mirrors.
  |-->
  <mirror>
    <id>mirrorId</id>
    <mirrorOf>repositoryId</mirrorOf>
    <name>Human Readable Name for this Mirror.</name>
    <url>http://my.repository.com/repo/path</url>
  </mirror>
  -->

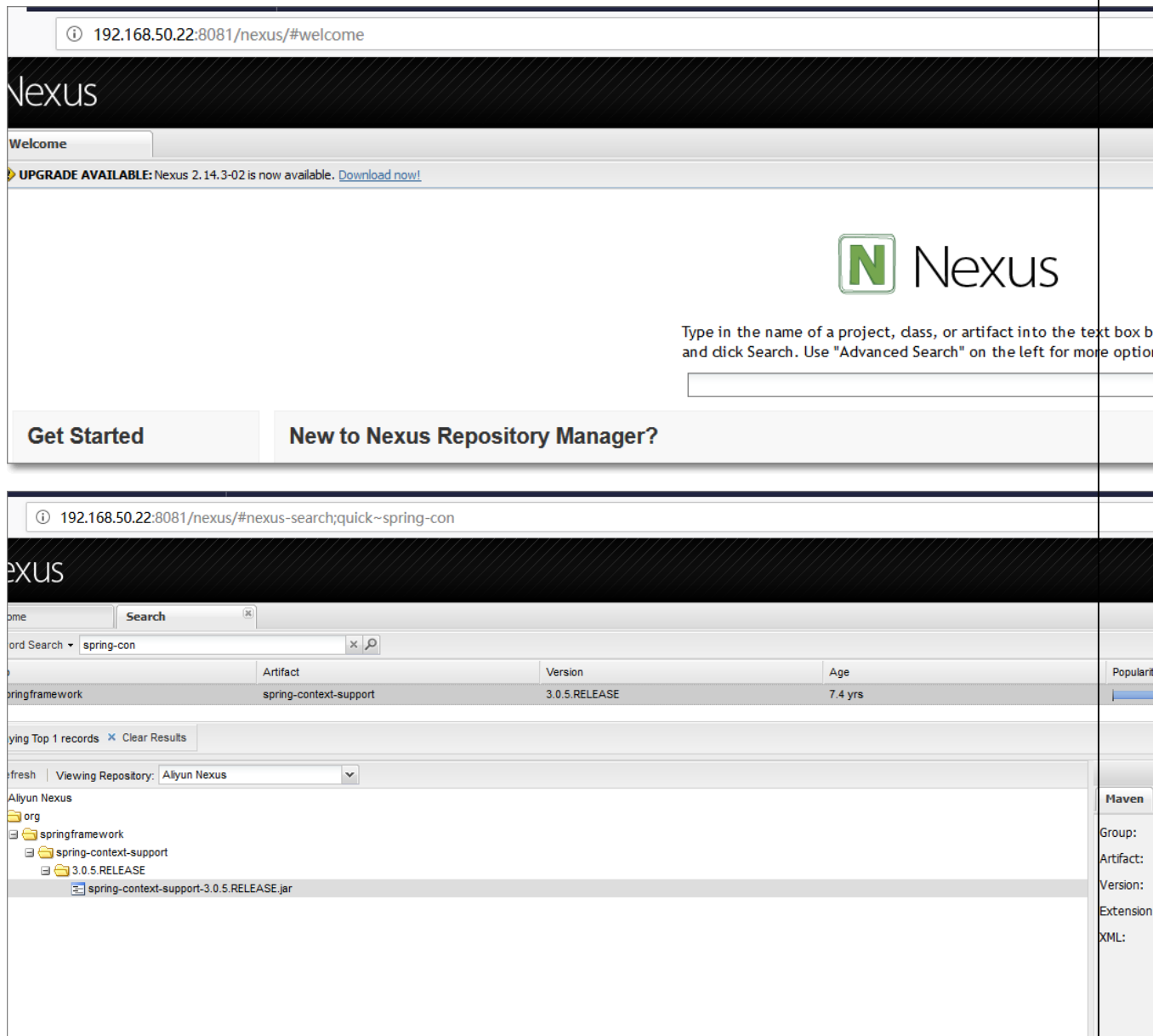
  <mirror>
    <id>my</id>
    <mirrorOf>*</mirrorOf>
    <name>Human Readable Name for this Mirror.</name>
    <url>http://192.168.50.22:8081/nexus/content/groups/public/</url>
  </mirror>
</mirrors>
```

设置完之后在 Eclipse 中对文件进行 Update:

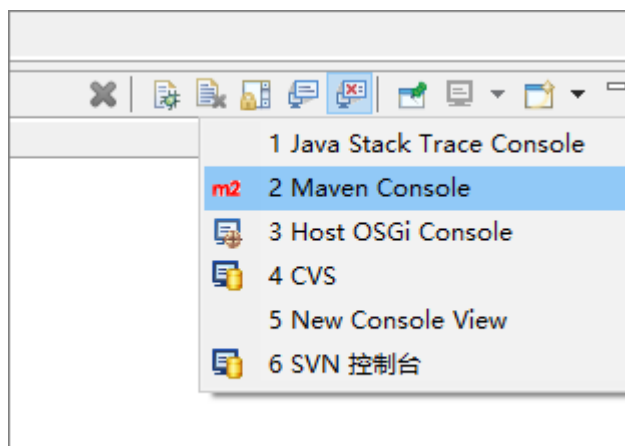


至此，我们以后使用的 `jar` 包就全部来自学校的私服仓库了，实际工作时大家使用的会是自己公司的私服

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：



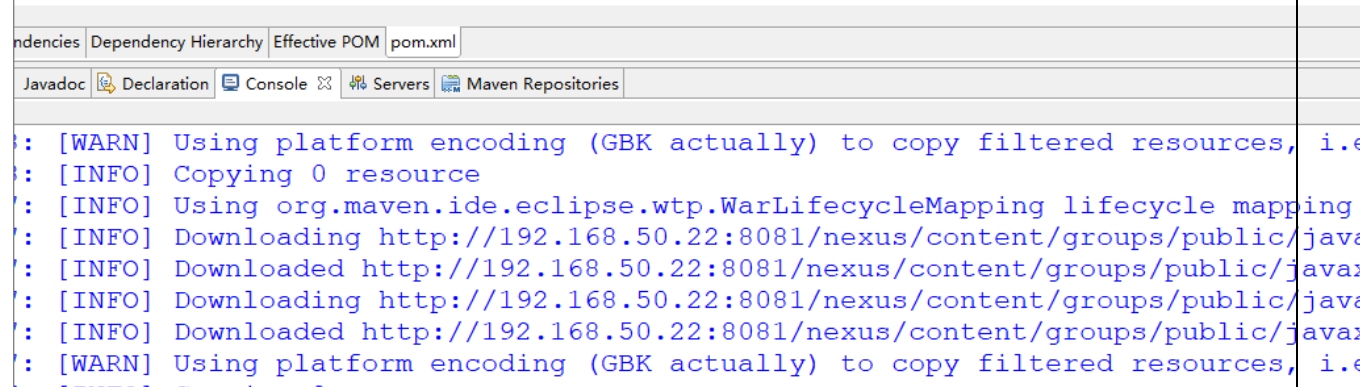
将控制台窗口修改为 Maven Console:



更新本地 `servlet` 的版本:

```
<dependencies>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.0</version>
    <scope>provided</scope>
  </dependency>
</dependencies>

</project>
```



Dependencies | Dependency Hierarchy | Effective POM | pom.xml

Javadoc | Declaration | Console | Servers | Maven Repositories

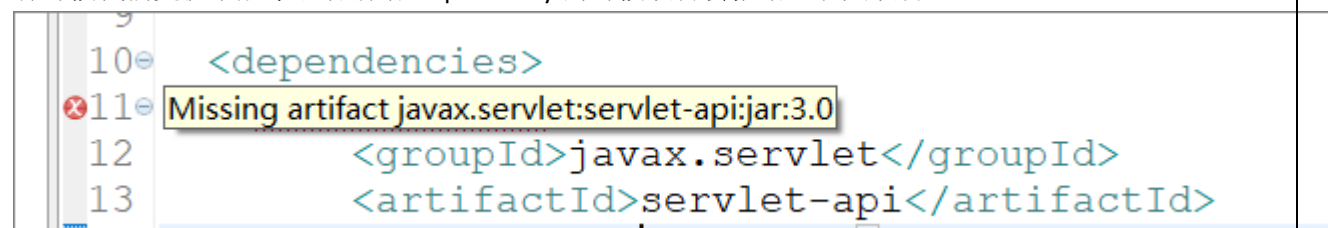
```
: [WARN] Using platform encoding (GBK actually) to copy filtered resources, i.e.
: [INFO] Copying 0 resource
: [INFO] Using org.maven.ide.eclipse.wtp.WarLifecycleMapping lifecycle mapping
: [INFO] Downloading http://192.168.50.22:8081/nexus/content/groups/public/javax
: [INFO] Downloaded http://192.168.50.22:8081/nexus/content/groups/public/javax
: [INFO] Downloading http://192.168.50.22:8081/nexus/content/groups/public/javax
: [INFO] Downloaded http://192.168.50.22:8081/nexus/content/groups/public/javax
: [WARN] Using platform encoding (GBK actually) to copy filtered resources, i.e.
```

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。

### 4.2.3. 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



```
10 <dependencies>
11 Missing artifact javax.servlet:servlet-api:jar:3.0
12   <groupId>javax.servlet</groupId>
13   <artifactId>servlet-api</artifactId>
```

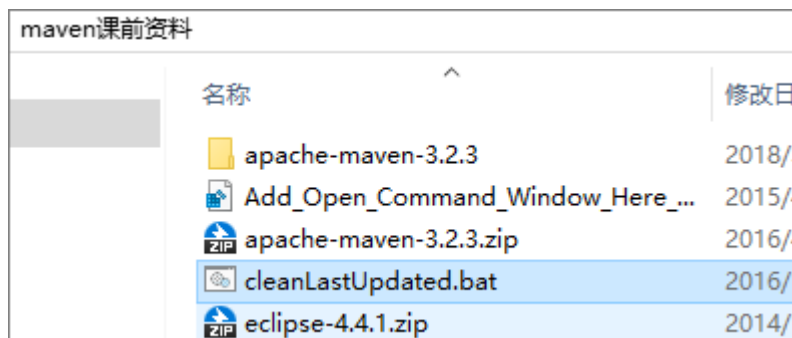
产生错误的原因大致有两种可能:

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

手动删除文件太辛苦:

在课前资料中提供了删除 `lastUpdated` 的批处理文件:



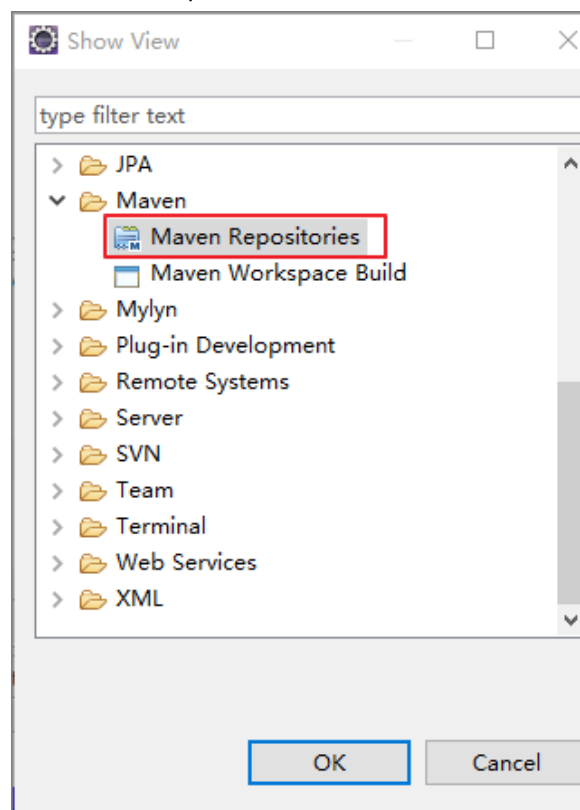
笔记本打开后将仓库地址改为自己的仓库路径

```
cleanLastUpdated.bat
1 set REPOSITORY_PATH=D:\Applications\Apache\apache-maven-3.2.3\repository
2 rem 正在搜索...
3 for /f "delims=" %%i in ('dir /b /s "%REPOSITORY_PATH%\*lastUpdated"')
4     del /s /q %%i
5 )
6 rem 搜索完毕
7 pause
```

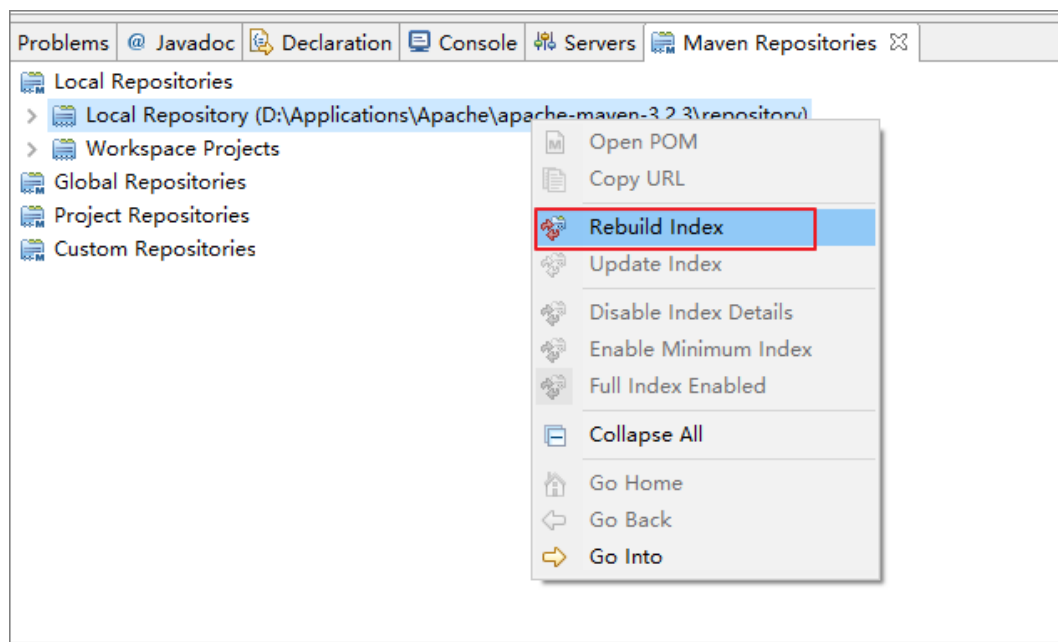
它能帮助我们删除仓库中所有的 lastUpdated 文件

#### 4.2.4. Eclipse 工具快捷添加依赖

打开 Maven Repositories 视图:



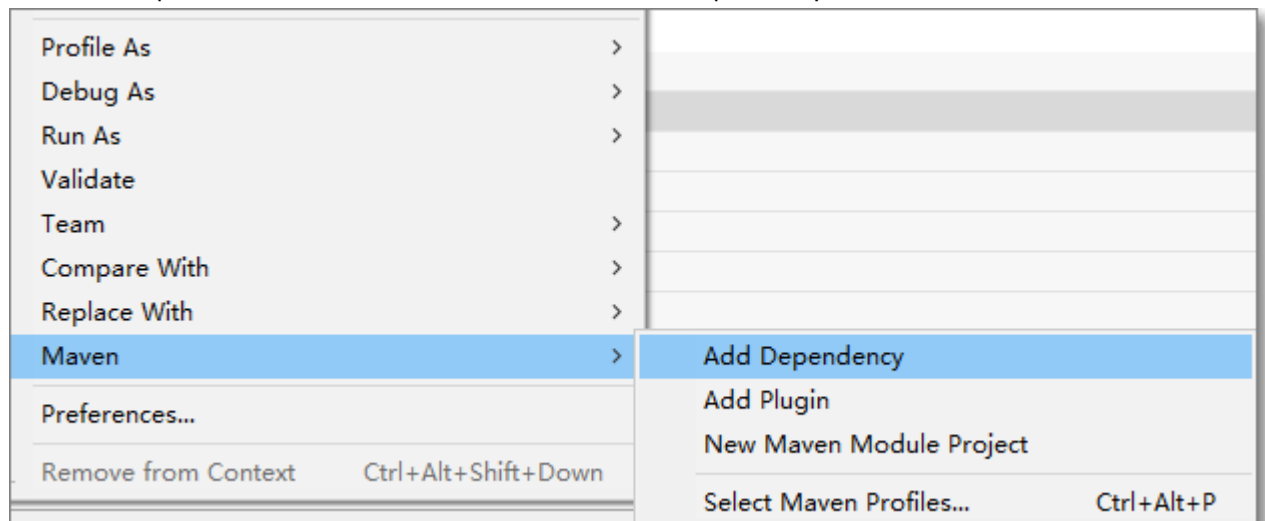
重建本地仓库的索引：



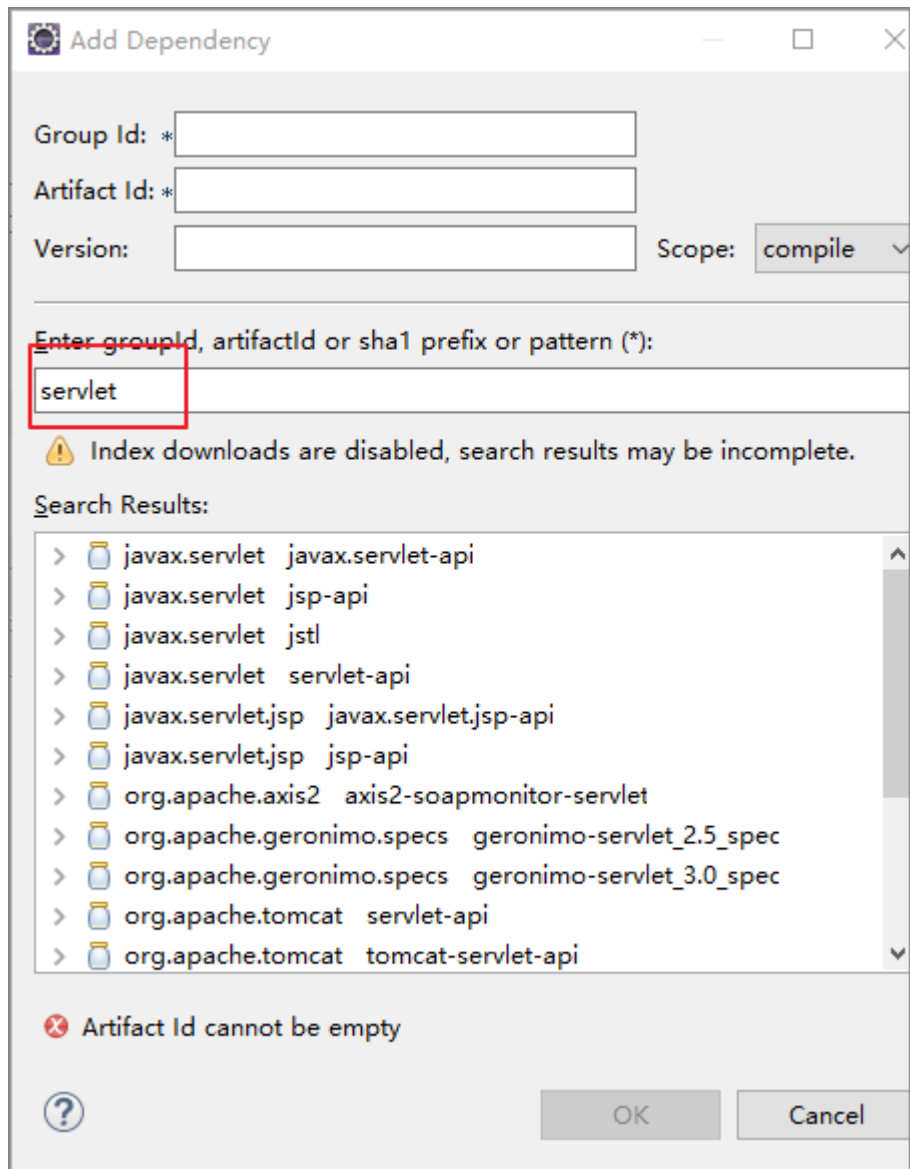
这样做的目的是为了让 eclipse 工具知道本地仓库中已经拥有哪些 jar 包

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

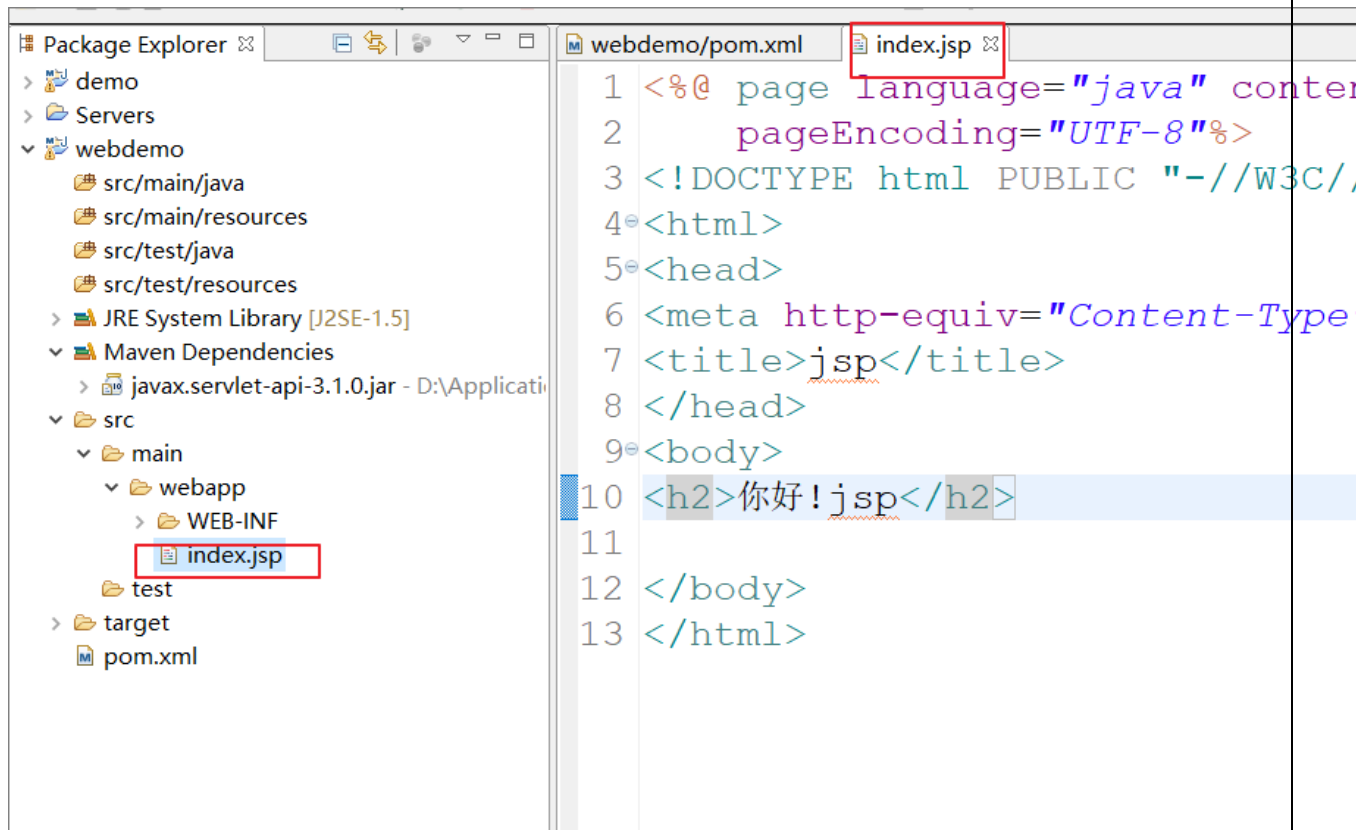


在输入框中搜索想要引入的项目：

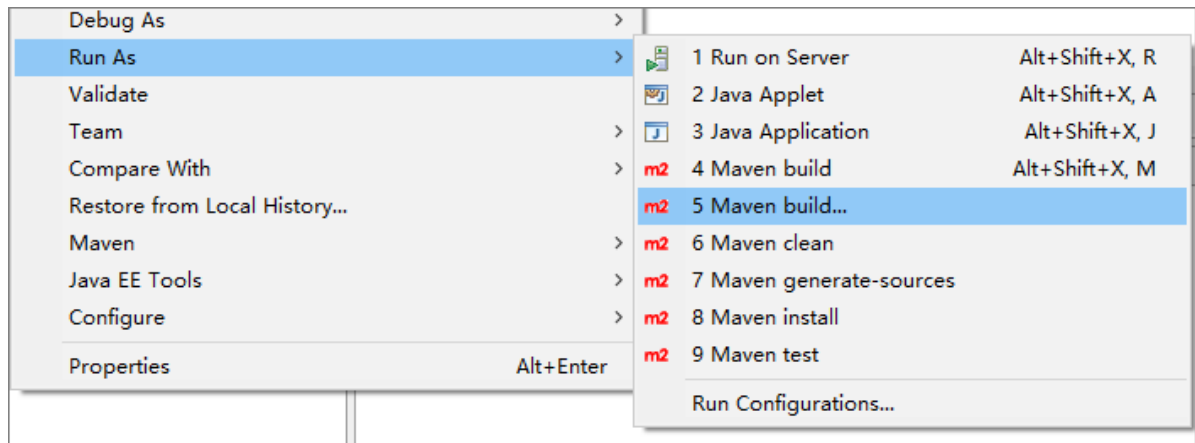


```
<dependency>  
  <groupId>javax.servlet</groupId>  
  <artifactId>servlet-api</artifactId>  
  <version>2.5</version>  
  <scope>provided</scope>  
</dependency>
```

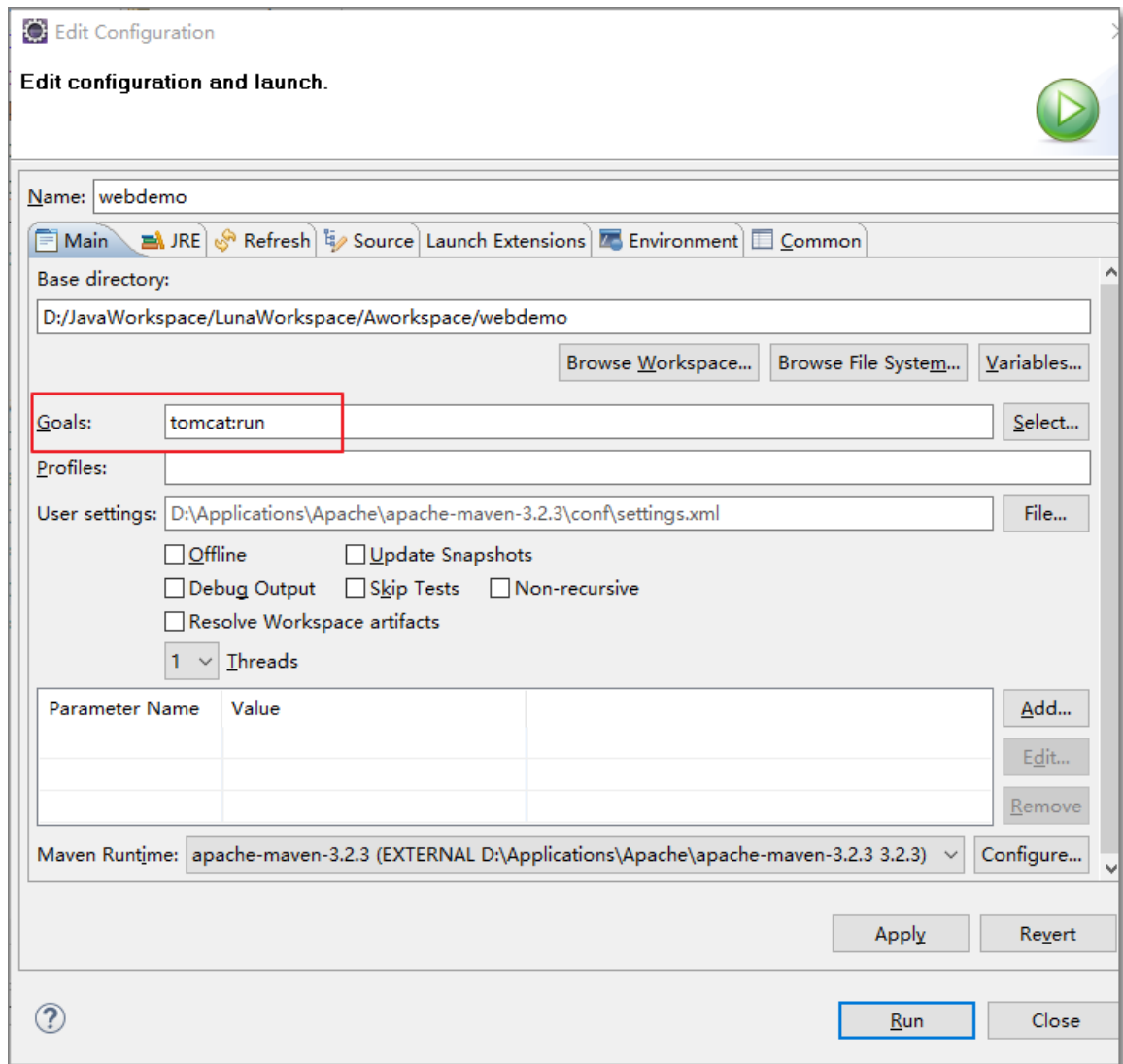
创建 jsp 页面:



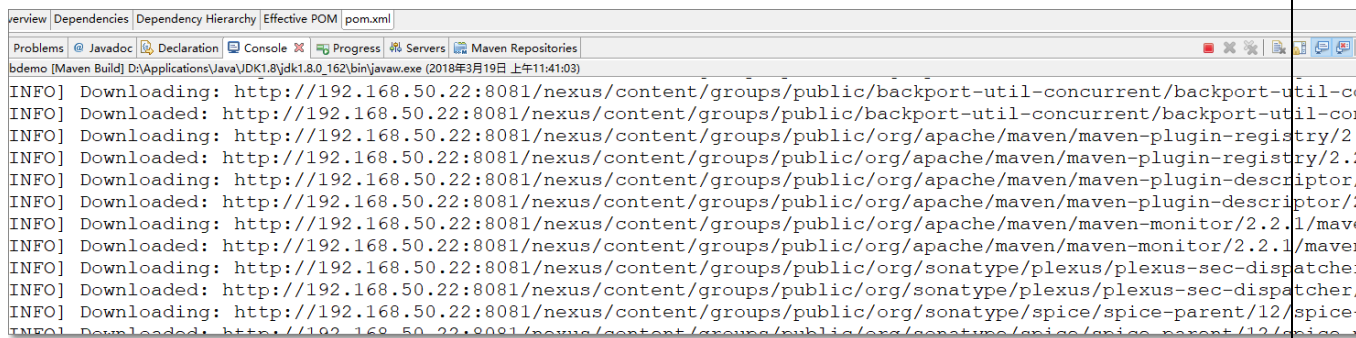
通过 Maven 内置的 Tomcat 插件运行项目：



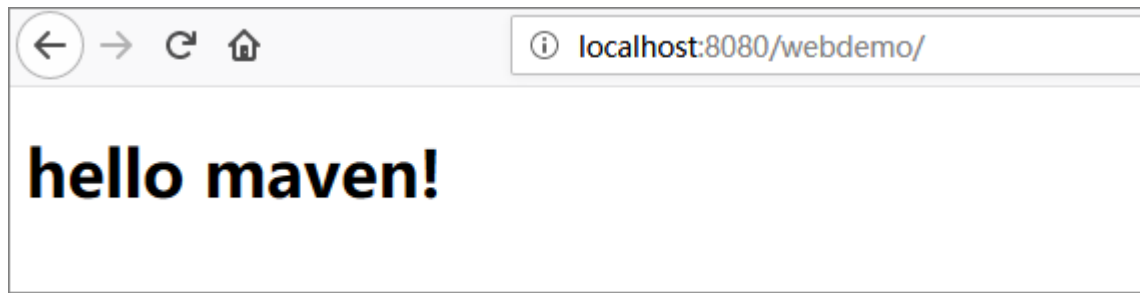




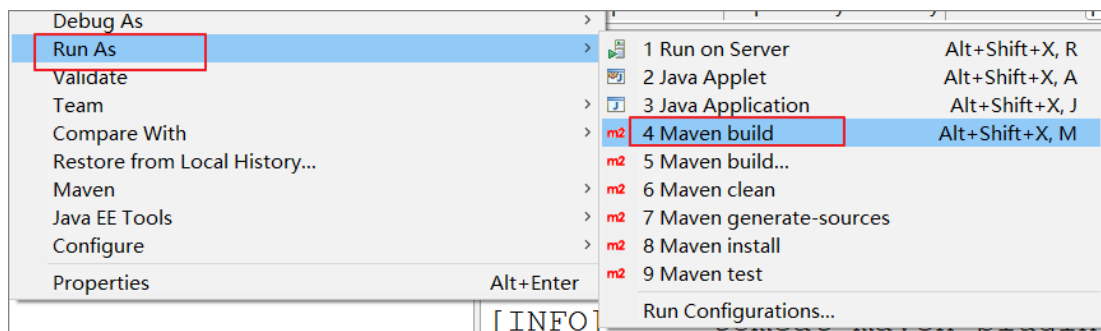
提示:插件也是需要联网下载的,本地没有也会从私服或者中央仓库下载:



访问项目



重新运行项目或者执行 maven 的上一次命令:



## 5. 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。

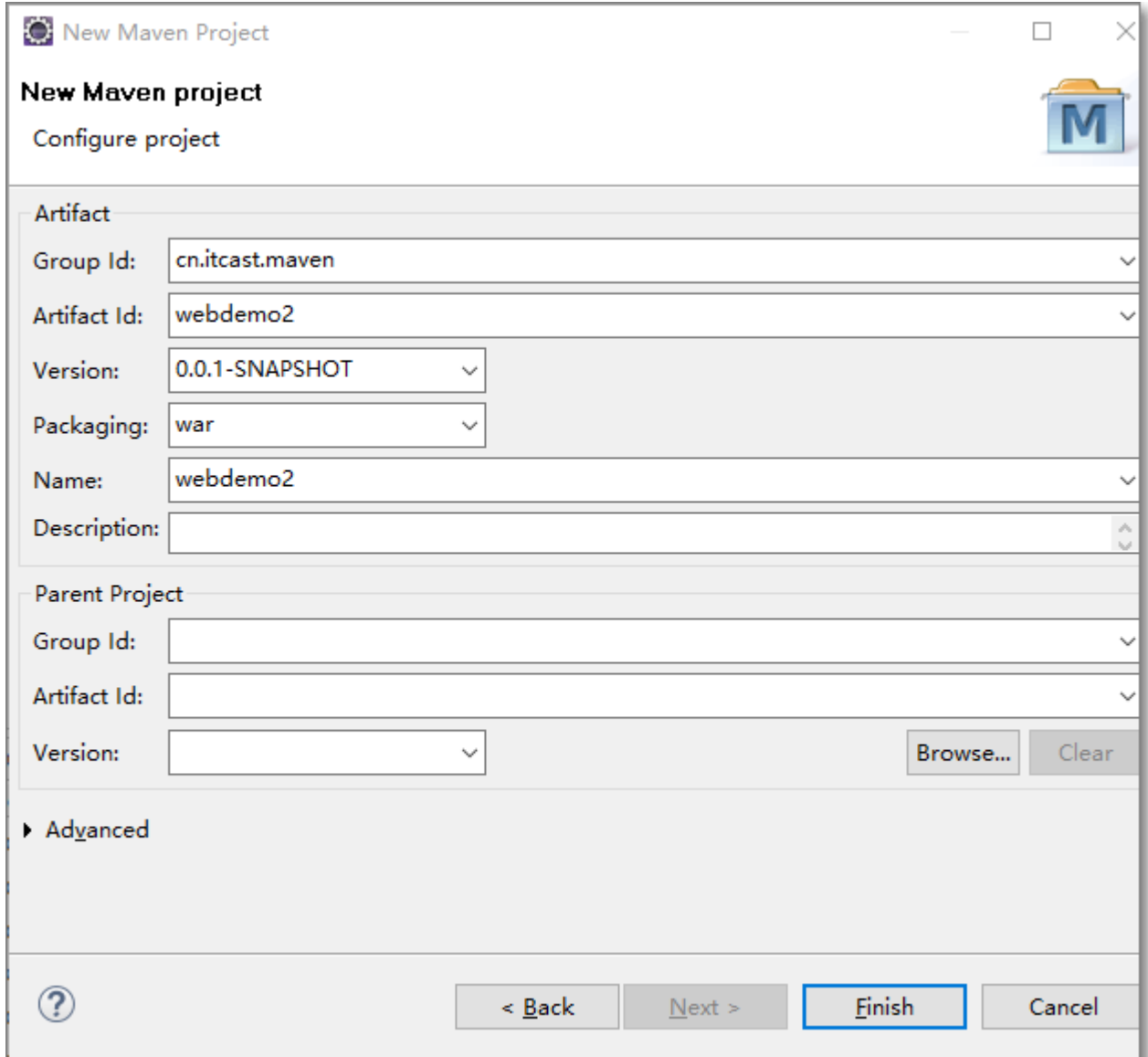
提示：导入 servlet-api-xxx.jar,jsp-api-xx.jar 依赖范围选择：**provided**

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>servlet-api</artifactId>
  <version>2.5</version>
  <scope>provided</scope>
</dependency>
```

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。

## 6. Maven 传递依赖冲突

新建一个 Maven web 项目：



**New Maven Project**

**New Maven project**  
Configure project

**Artifact**

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

**Parent Project**

Group Id:

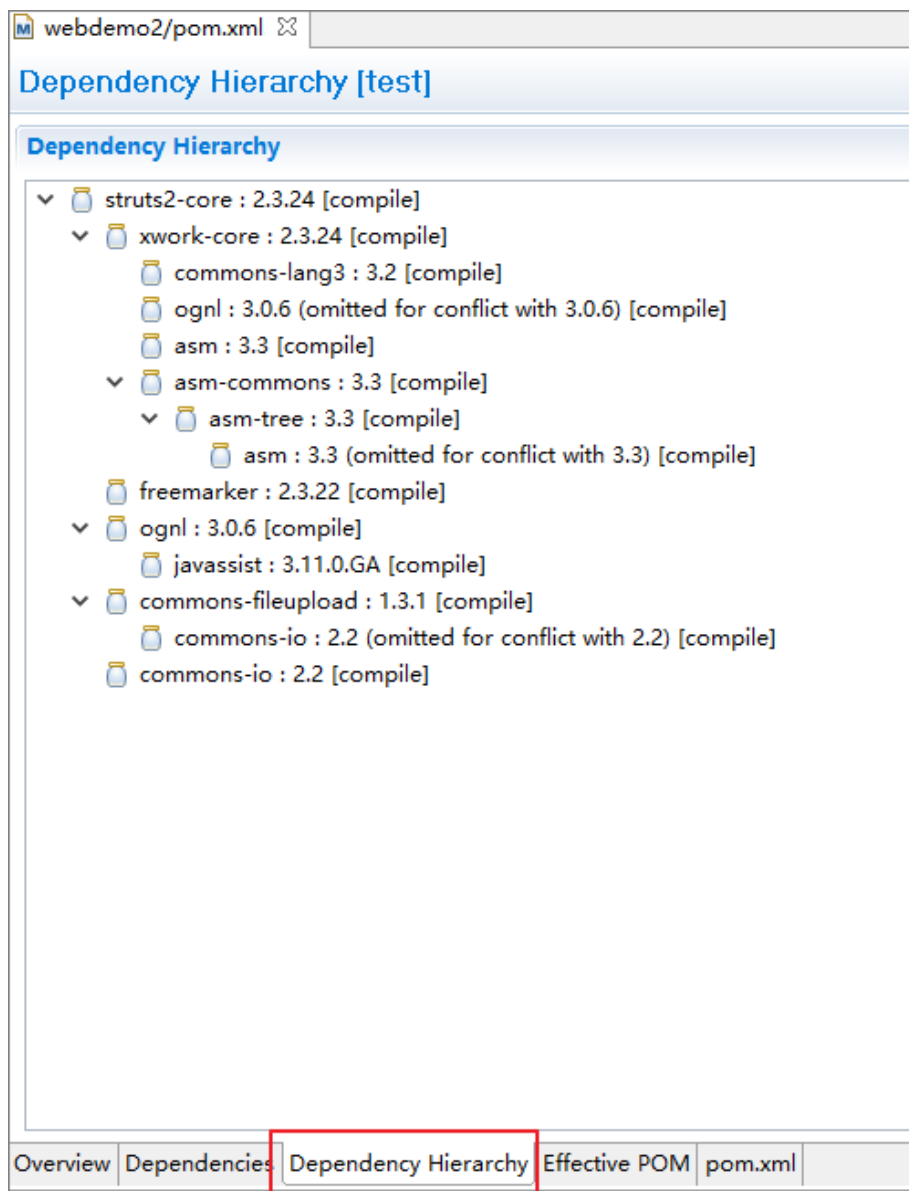
Artifact Id:

Version:

▶ **Advanced**

### 6.1. 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:

```

<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>4.2.8.RELEASE</version>
</dependency>

```

多个依赖传递中出现了相同的依赖时,冲突就产生了:

webdemo2/pom.xml

### Dependency Hierarchy [test]

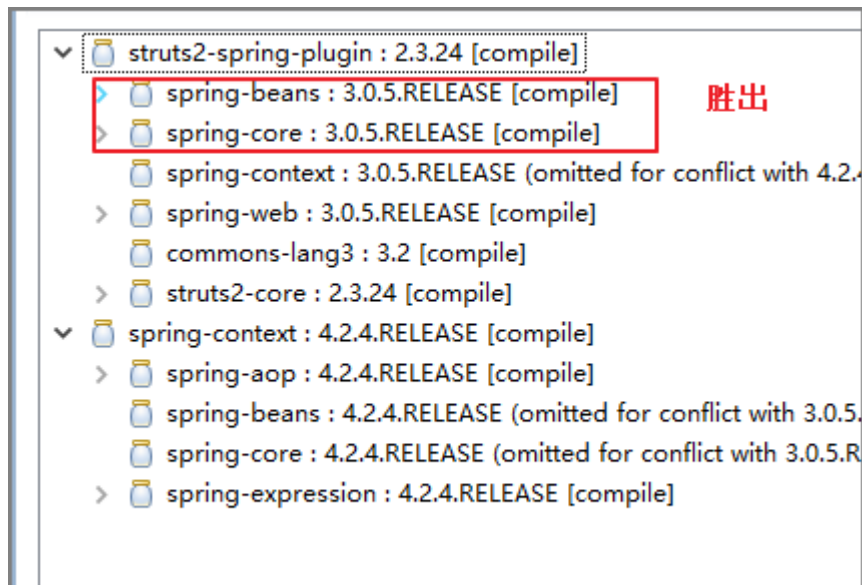
#### Dependency Hierarchy

- ▼ struts2-spring-plugin : 2.3.24 [compile]
  - > spring-beans : 3.0.5.RELEASE [compile]
  - > spring-core : 3.0.5.RELEASE [compile]
    - spring-context : 3.0.5.RELEASE (omitted for conflict with 4.2.8.RELEASE) [compile]
  - > spring-web : 3.0.5.RELEASE [compile]
    - commons-lang3 : 3.2 [compile]
  - > struts2-core : 2.3.24 [compile]
- ▼ spring-context : 4.2.8.RELEASE [compile]
  - > spring-aop : 4.2.8.RELEASE [compile]
    - spring-beans : 4.2.8.RELEASE (omitted for conflict with 3.0.5.RELEASE) [compile]
    - spring-core : 4.2.8.RELEASE (omitted for conflict with 3.0.5.RELEASE) [compile]
  - ▼ spring-expression : 4.2.8.RELEASE [compile]
    - spring-core : 4.2.8.RELEASE (omitted for conflict with 3.0.5.RELEASE) [compile]

## 6.2. Maven 自身的冲突调节原则

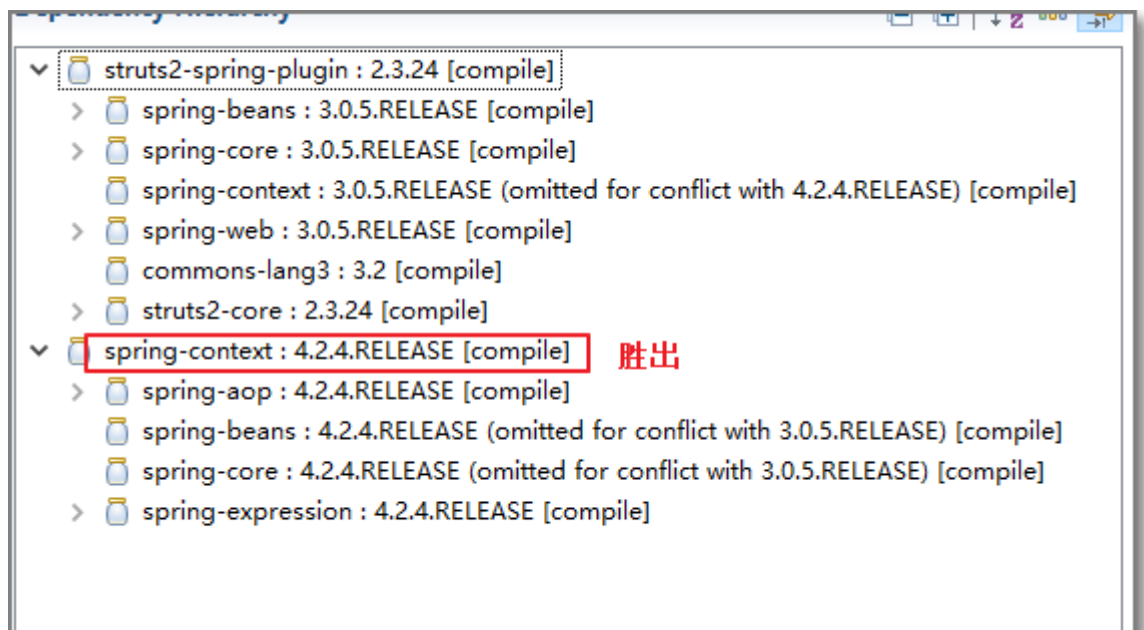
### 6.2.1. 第一声明者优先原则

根据依赖的配置顺序,谁先声明就用谁的依赖:



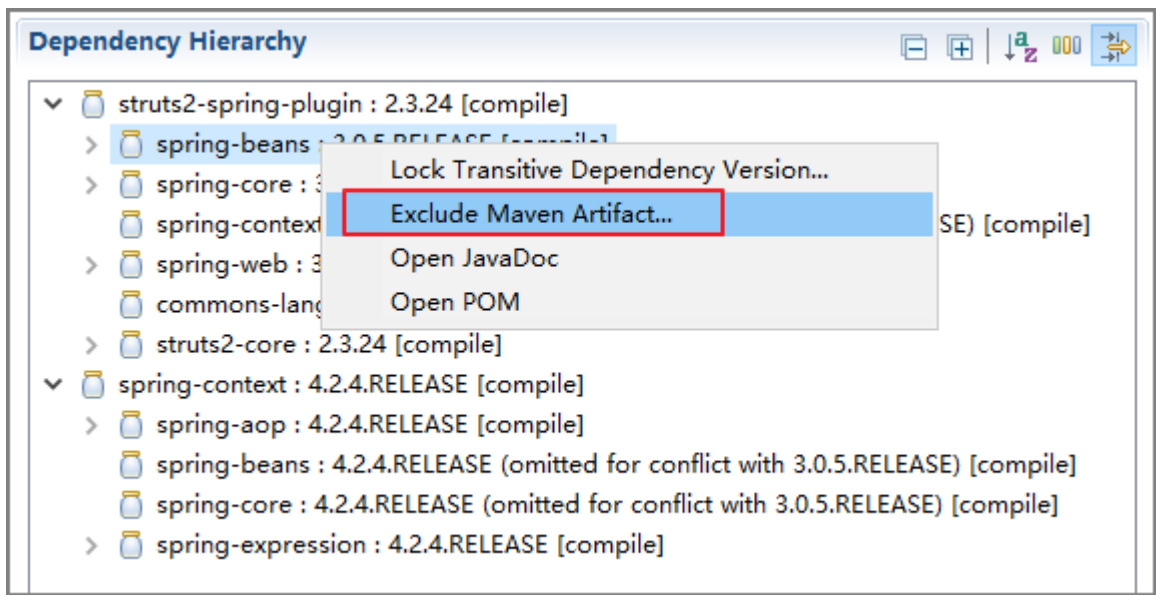
### 6.2.2. 路径近者优先原则

直接依赖的级别要比传递依赖优先:



### 6.3. 排除依赖

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



```
<dependency>
  <groupId>org.apache.struts</groupId>
  <artifactId>struts2-spring-plugin</artifactId>
  <version>2.3.24</version>
  <exclusions>
    <exclusion>
      <artifactId>spring-beans</artifactId>
      <groupId>org.springframework</groupId>
    </exclusion>
  </exclusions>
</dependency>
```

## 6.4. 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>

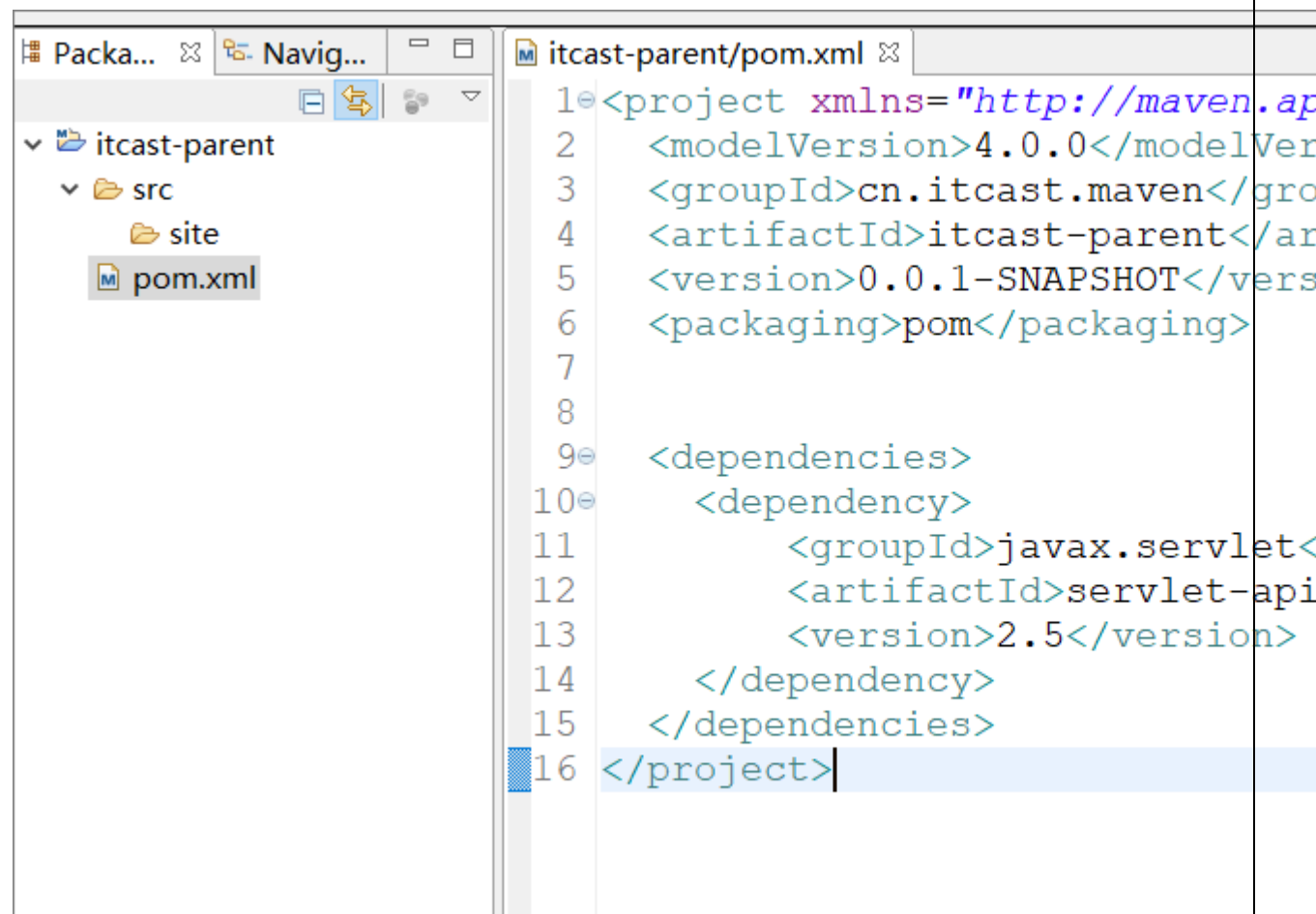
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-web</artifactId>
  <version>${spring-version}</version>
</dependency>
</dependencies>
</dependencyManagement>
```

## 7. 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:

 New Maven Project

New Maven project

Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:


Parent Project

Group Id:

Artifact Id:

Version:

Advanced



## 8. 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

