自动构建工具：Maven基础Eclipse版

1. Maven的简介

目前的技术在开发中存在的问题

①一个项目就是一个工程

如果项目非常庞大，就不适合继续使用package来划分模块。最好是每一个模块对应一个工程，利于分工协作。

借助于Maven就可以将一个项目拆分成多个工程。

②项目中需要的jar包必须手动“复制”、“粘贴”到WEB-INF/lib目录下

带来的问题是：同样的jar包文件重复出现在不同的项目工程中，一方面浪费存储空间，另外也让工程比较臃肿。不同jar包的下载形式五花八门

有些技术的官网就是通过Maven或SVN等专门的工具来下载

借助Maven可以以规范的方式下载jar包

③jar包需要别人替我们准备好，或到官网下载

④一个jar包依赖的其他jar包需要自己手动加入到项目中

需要清除明确的依赖关系

清楚

1.1 Maven的概念

maven是一个Java语言编写的开源项目管理工具，是Apache软件基金会的顶级项目.主要用于项目构建，依赖管理，项目信息管理,是现今最流行的Java项目构建工具.

1.2 项目构建的概念

项目构建是一个项目从编写源代码到编译、测试、运行、打包、部署的过程。

.Maven是什么[What]

①Maven是一款服务于Java平台的自动化构建工具。

Make→Ant→Maven→Gradle

妹文、麦文

②构建

[1]概念：以“Java源文件”、“框架配置文件”、“JSP”、“HTML”、“图片”等资源为“原材料”，去“生产”一个可以运行的项目的过程。

- 编译
- 部署
- 搭建

[2]编译：Java源文件[User.java]→编译→Class字节码文件[User.class]→交给JVM去执行

[3]部署：一个BS项目最终运行的并不是动态Web工程本身，而是这个动态Web工程“编译的结果”

生的鸡→处理→熟的鸡

动态Web工程→编译、部署→编译结果

①纯 Java 代码

大家都知道，我们 Java 是一门编译型语言，.java 扩展名的源文件需要编译成.class 扩展名的字节码文件才能够执行。所以编写任何 Java 代码想要执行的话就必须经过编译得到对应的.class 文件。

②Web 工程

当我们需要通过浏览器访问 Java 程序时就必须将包含 Java 程序的 Web 工程编译的结果“拿”到服务器上的指定目录下，并启动服务器才行。这个“拿”的过程我们叫部署。

我们可以将未编译的 Web 工程比喻为一只生的鸡，编译好的 Web 工程是一只煮熟的鸡，编译部署的过程就是将鸡炖熟。

③实际项目

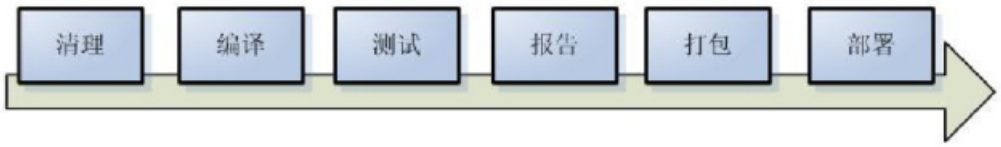
在实际项目中整合第三方框架，Web 工程中除了 Java 程序和 JSP 页面、图片等静态资源之外，还包括第三方框架的 jar 包以及各种各样的配置文件。所有这些资源都必须按照正确的目录结构部署到服务器上，项目才可以运行。

所以综上所述：构建就是以我们编写的 Java 代码、框架配置文件、国际化等其他资源文件、JSP 页面和图片等静态资源作为“原材料”，去“生产”出一个可以运行的项目的过程。

③构建过程中的各个环节

[1]清理：将以前编译得到的旧的class字节码文件删除，为下一次编译做准备
[2]编译：将Java源程序编程成class字节码文件
[3]测试：自动测试，自动调用junit程序
[4]报告：测试程序执行的结果
[5]打包：动态Web工程打war包，Java工程打jar包
[6]安装：Maven特定的概念——将打包得到的文件复制到“仓库”中的指定位置
[7]部署：将动态Web工程生成的war包复制到Servlet容器的指定目录下，使其可以运行

maven 将项目构建的过程进行标准化，每个阶段使用一个命令完成，下图展示了构建过程的一些阶段，后面章节详细介绍每个阶段，这里先大概了解下：



上图中部分阶段对应命令如下：

清理阶段对应 maven 的命令是 clean，清理输出的 class 文件

编译阶段对应 maven 的命令是 compile，将 java 代码编译成 class 文件。

打包阶段对应 maven 的命令是 package，java 工程可以打成 jar 包，web 包可以打成 war 包

2. Maven的安装配置及目录结构

首先，确保JAVA_HOME正确配置

然后解压Maven安装包到某个非中文路径下

配置Maven的环境变量MAVEN_HOME

验证：mvn -v

ications (D:) › apache-maven › apache-maven-3.3.9 ›

名称	修改日期	类型	大小
bin	2019.07.09 09:53	文件夹	
boot 运行要类加载器	2019.07.09 09:53	文件夹	
conf settings.xml 整个Maven工具核心配置文件	2019.07.09 09:53	文件夹	
lib	2019.07.09 09:53	文件夹	
LICENSE	2015.11.10 11:44	文件	19 KB
NOTICE	2015.11.10 11:44	文件	1 KB
README.txt	2015.11.10 11:38	文本文档	3 KB

3. Maven的仓库

三种仓库

1、本地仓库 自己维护

本地仓库的配置只需要修改 settings.xml 文件就可以

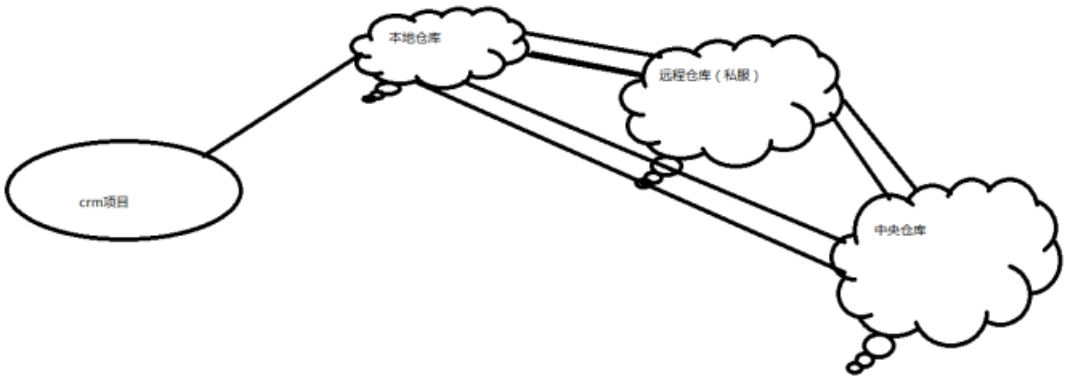
```
46 <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
47           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
48           xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apa
49   <!-- localRepository
50    | The path to the local repository maven will use to store artifacts.
51    |
52    | Default: ${user.home}/.m2/repository
53   <localRepository>/path/to/local/repo</localRepository>
54   -->
55 <localRepository>F:\class32\repository</localRepository>
56   <!-- interactiveMode
57    | This will determine whether maven prompts you when it needs input. If set to false,
58    | maven will use a sensible default value, perhaps based on some other setting, for
59    | the parameter in question.
60    |
61    | Default: true
62   <interactiveMode>true</interactiveMode>
63   -->
```

自己解压的本地仓库的路径

2、远程仓库（私服） 公司维护

3、中央仓库 maven 团队维护 两个亿 的jar包

三种仓库的关系如下：



4. Maven工程的目录结构

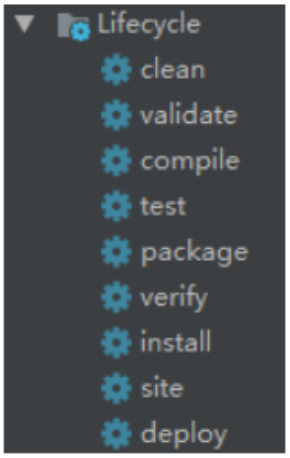

```
Project
  |-src
  |    |-main
  |    |    |-java          —— 存放项目的.java 文件
  |    |    |-resources     —— 存放项目资源文件，如 spring, hibernate 配置文件
  |    |         |-webapp        —— webapp 目录是 web 工程的主目录
  |    |              |-WEB-INF
  |    |                   |-web.xml
  |    |-test
  |         |-java          ——存放所有测试.java 文件，如 JUnit 测试类
  |         |-resources     —— 测试资源文件
  |-target                  —— 目标文件输出位置例如.class、.jar、.war 文件
  |-pom.xml                 ——maven 项目核心配置文件
```

5. Maven的常用命令

```
1、mvn compile 编译,将Java 源程序编译成 class 字节码文件。
2、mvn test 测试，并生成测试报告
3、mvn clean 将以前编译得到的旧的 class 字节码文件删除
4、mvn pakage 打包,动态 web工程打 war包，Java工程打 jar 包。
5、mvn install 将项目生成 jar 包放在仓库中，以便别的模块调用
```

6. Maven的生命周期(其功能由插件完成)

6.1 clean生命周期



pre-clean（执行一些清理前需要完成的工作）
clean （清理上一次构建生成的文件，最常用）
post-clean（执行一些清理后需要完成的工作）

6.2 default生命周期

default生命周期

default生命周期定义了真正构建时所需要的执行的所有步骤，它是生命周期中最核心的部分，包含内容很多，　　只讲其中最重要的内容。

process-sources（处理项目主资源文件，将src/main/resources目录的内容经过处理后，复制到项目输出的主classpath目录中）
compile（编译项目主源码，编译src/main/java目录下的java文件至项目输出的主classpath目录中）
process-test-source（处理项目测试资源文件。对src/test/resources目录）
test-compile（编译项目的测试代码）
test（使用单元测试框架运行测试，测试代码不会被打包或部署）
package（接受编译好的代码，打包成可发布格式）
install（发布到本地仓库）
deploy（发布到远程仓库）

6.3 site生命周期

site生命周期

site生命周期建立和发布项目站点，Maven能够基于POM所包含信息，自动生成一个友好的站点，方便团队交流和发布项目信息。

一共包含四阶段　　只讲其中最重要的两阶段

site（生成项目站点文档）
site-deploy（将生成的项目站点发布到服务器上）

6.4 命令行与生命周期

mvn clean: 调用clean生命周期的clean阶段。实际执行为pre-clean和clean阶段

mvn compile : 实际执行default从头到compile阶段

mvn clean install: 调用clean生命周期的clean阶段和default生命周期的install阶段，实际调用了pre-clean ,clean 以及default生命周期的从validate到install所有阶段。其余命令使用诸如此类。

mvn tomcat:run （一键启动）

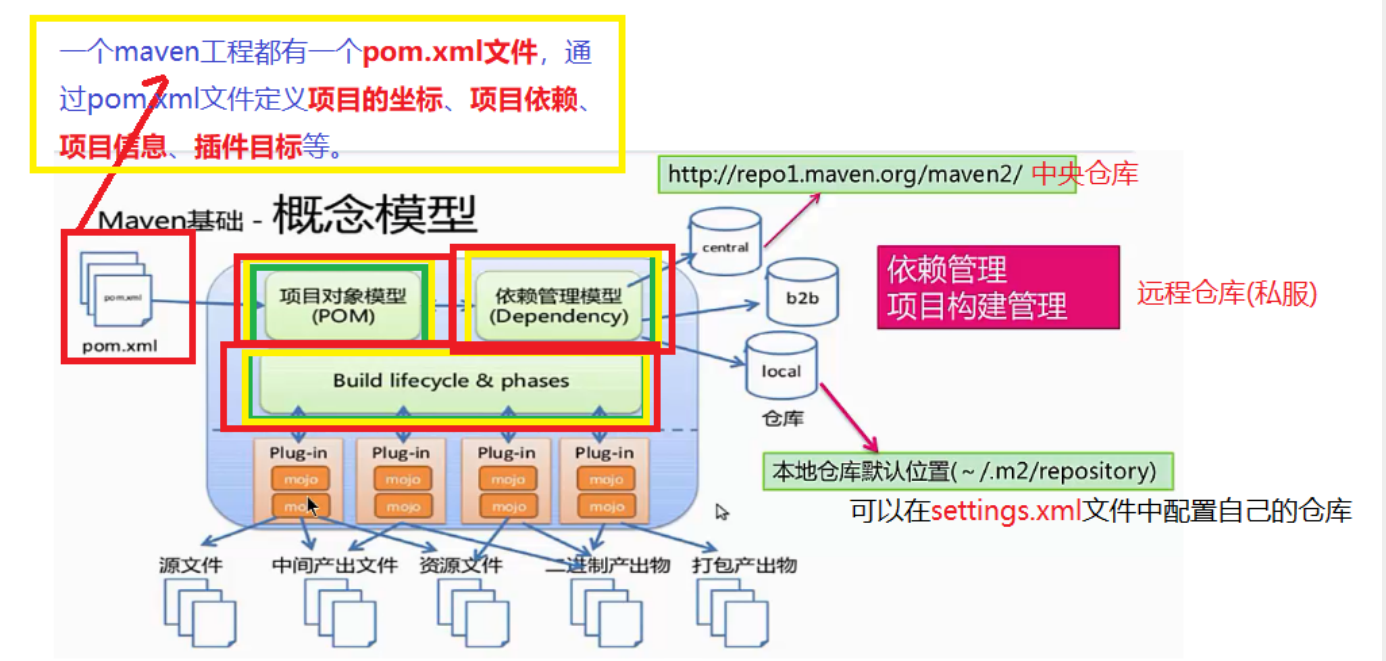
mvn test 编译并运行了test中内容

mvn package: 打包

mvn install : 发布项目到本地仓库

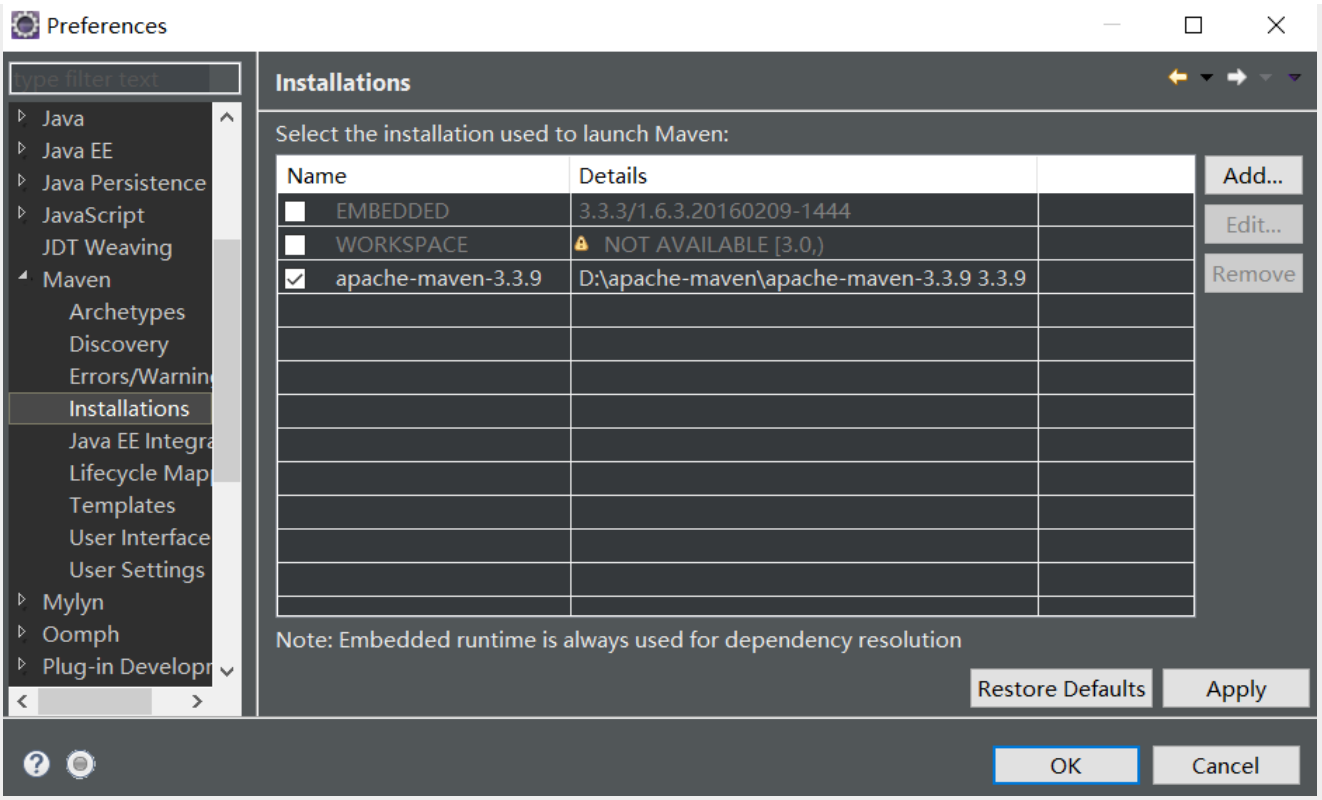
mvn deploy: 发布项目到远程仓库

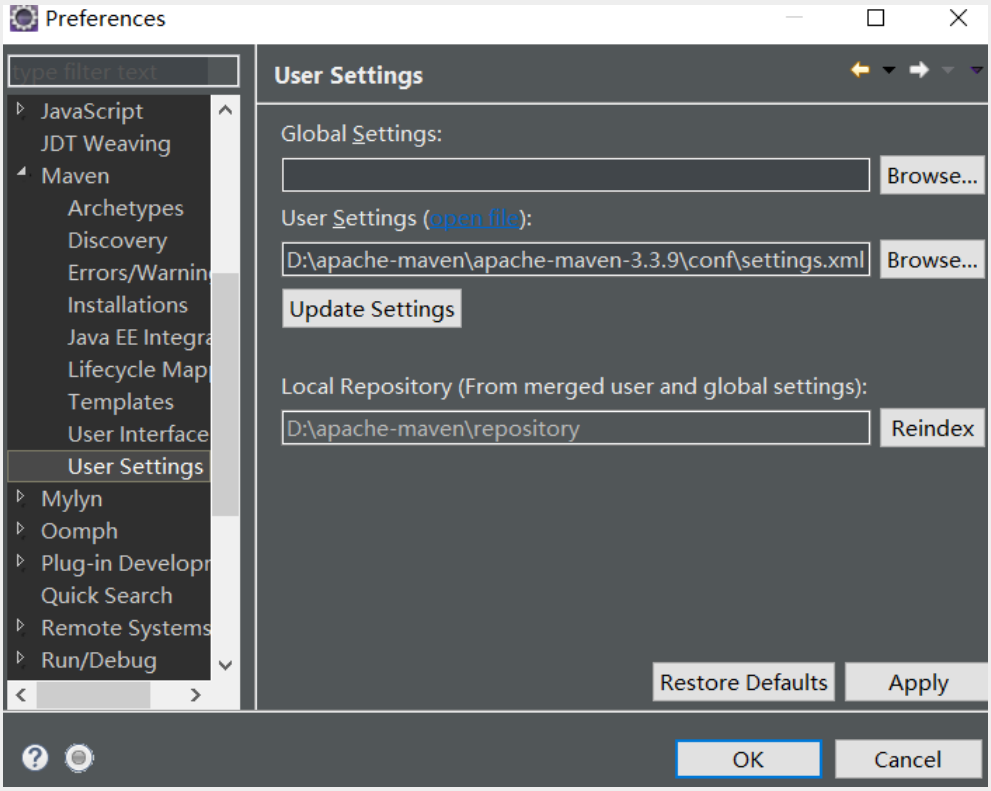
7. Maven的概念模型



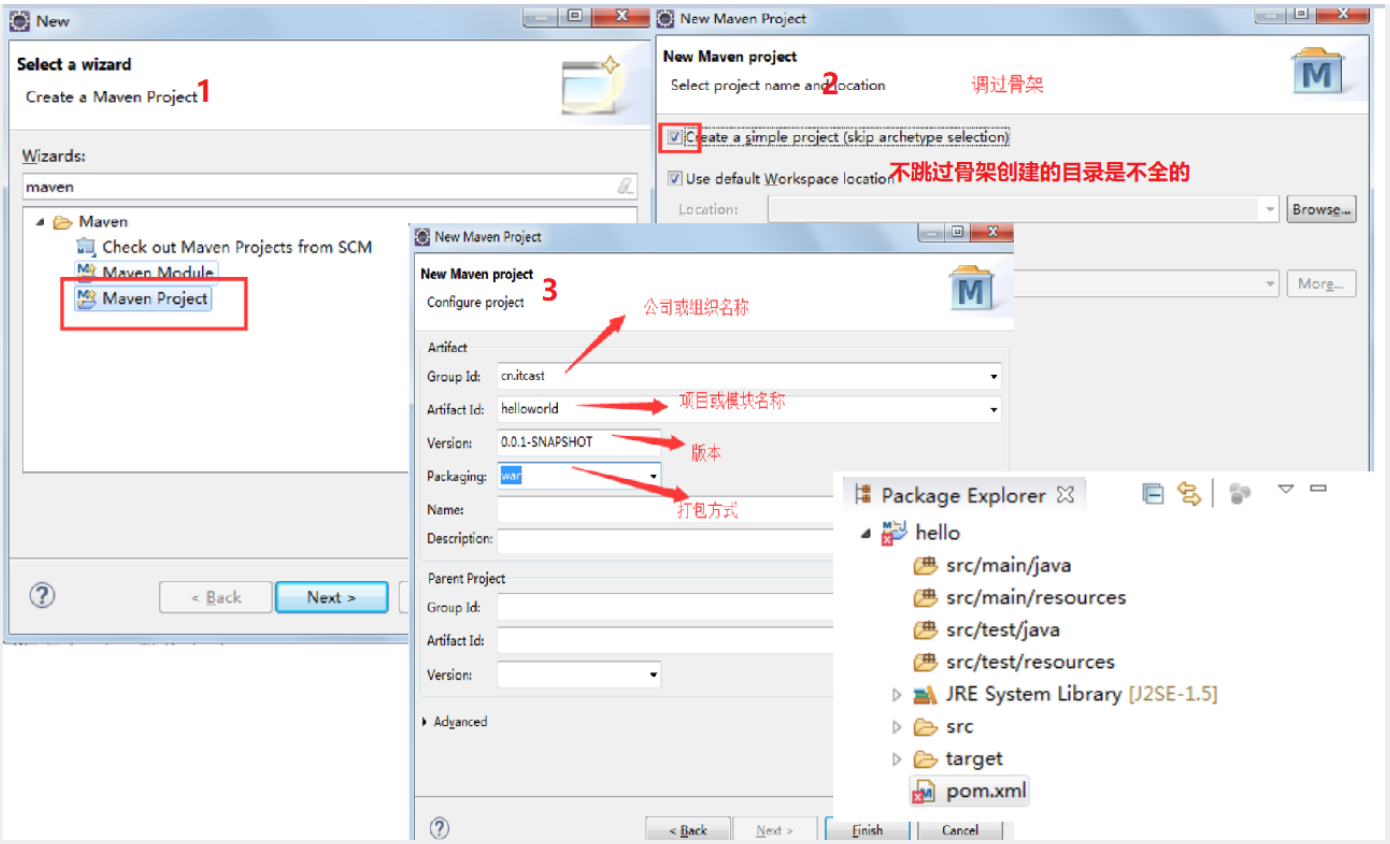
8. 在Eclipse中创建Maven项目

8.1 配置本地maven和设置仓库地址

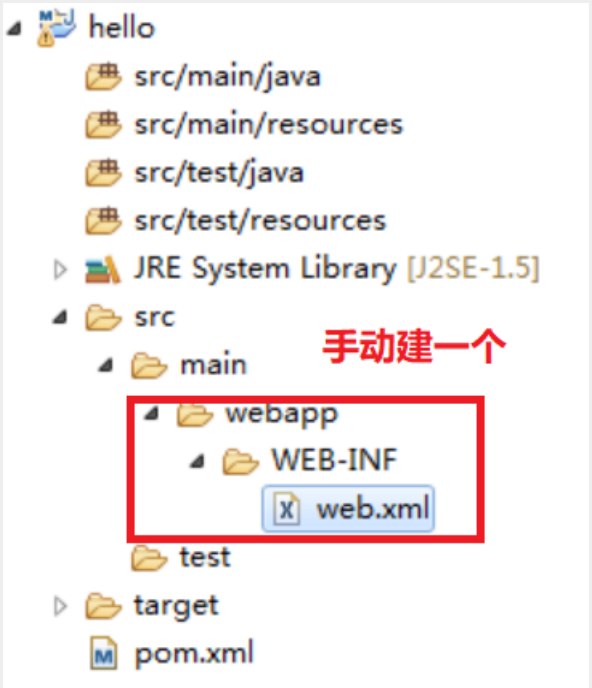




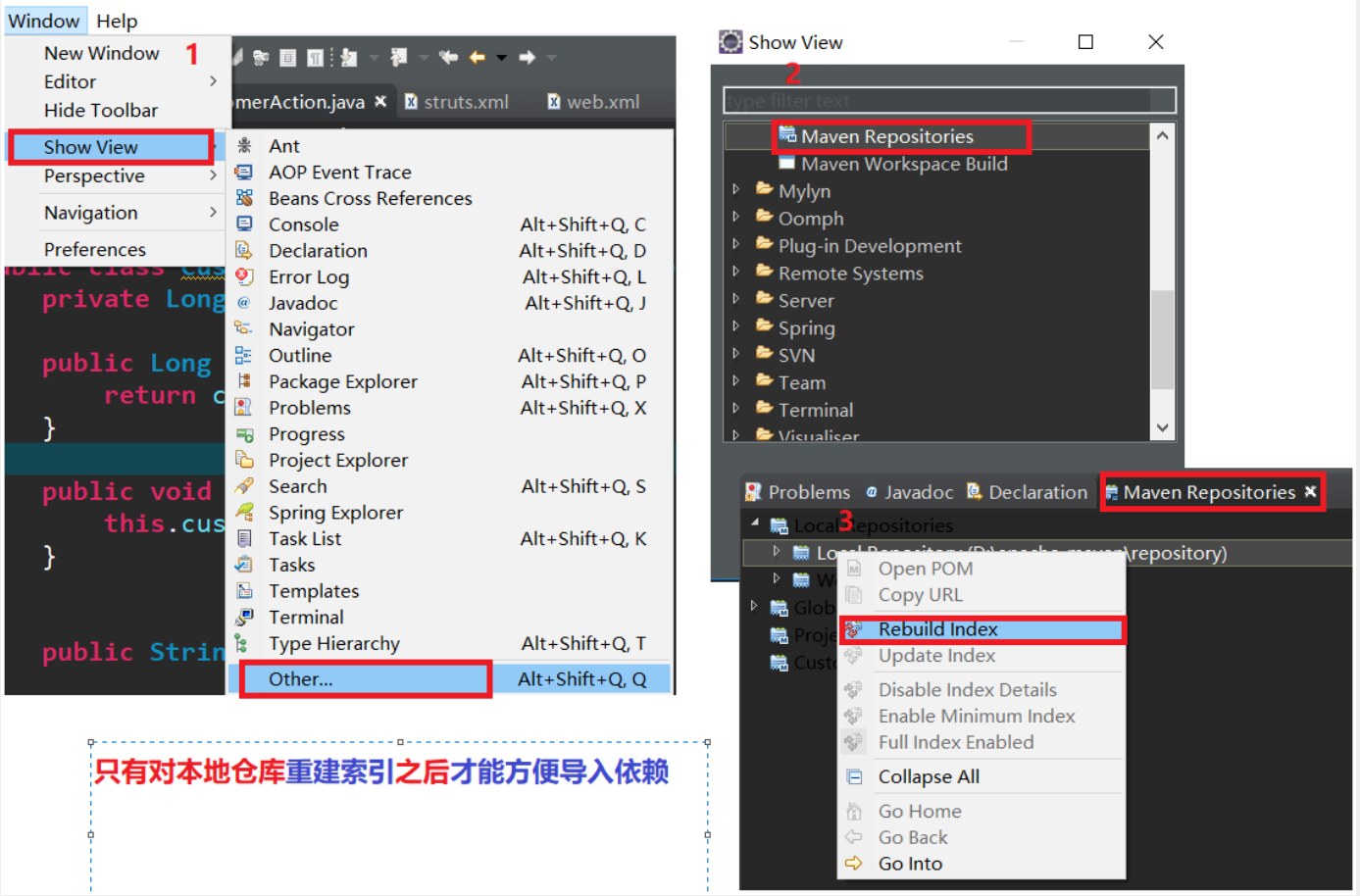
8.2 根据向导创建maven项目



8.3 处理红色叉号



8.4 重建仓库索引



8.5 修改JRE运行环境

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.5.1</version>
    <configuration>
        <source>1.7</source>
        <target>1.7</target>
        <encoding>UTF-8</encoding>
    </configuration>
</plugin>
```

JRE环境

8.6 修改tomcat配置

```
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <port>8888</port>
        <path>/first</path>
    </configuration>
</plugin>
```

本地仓库的tomcat

这里可以配置端口和访问路径

8.7 依赖范围：scope属性

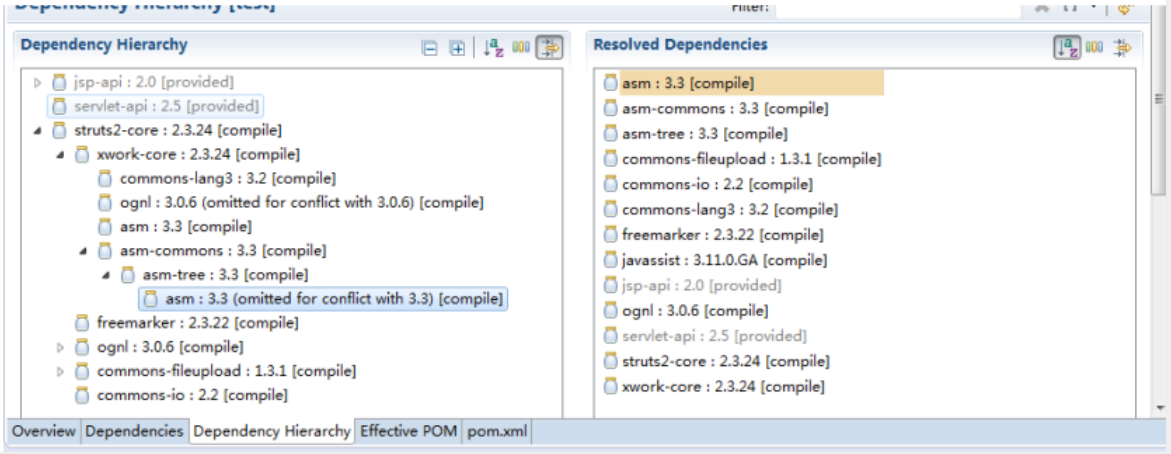
	编译	测试	运行	打包	
compile	✓	✓	✓	✓	默认依赖配置
provided	✓	✓			jsp-api.jar servlet-api.jar 与tomcat冲突，**编译和测试**时需要而已
runtime		✓	✓	✓	数据库驱动包
test		✓			junit包

```xml
<dependencies>
  <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jsp-api</artifactId>
      <version>2.0</version>
      <scope>provided</scope>
  </dependency>
  <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.5</version>
      <scope>provided</scope>
  </dependency>
  <!-- junit 是个人开发的 没有域名 -->
  <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.9</version>
      <scope>test</scope>
  </dependency>
  <dependency>
      <groupId>org.apache.struts</groupId>
      <artifactId>struts2-core</artifactId>
      <version>2.3.24</version>
  </dependency>
</dependencies>
```

8.8 依赖管理：处理版本冲突

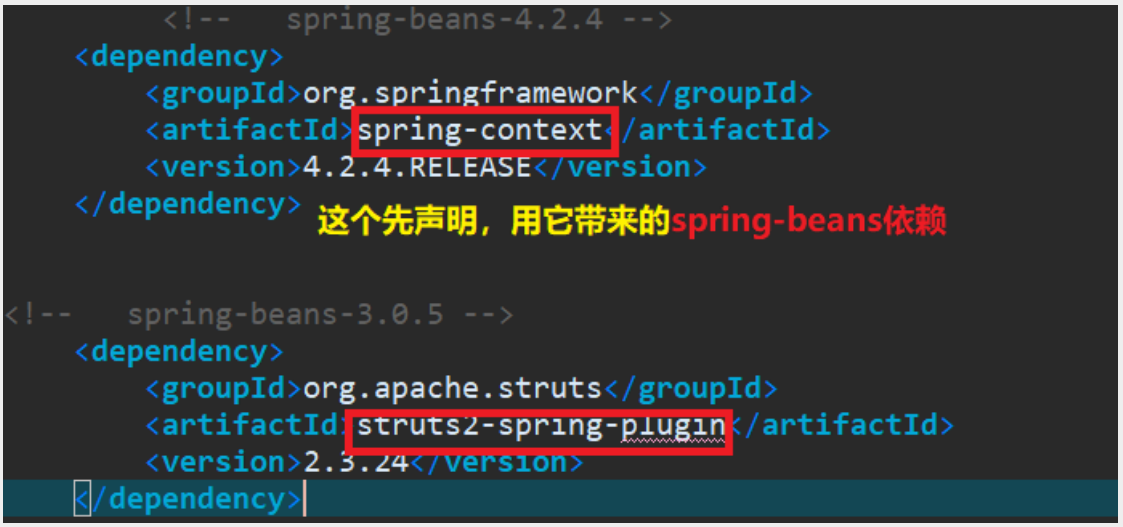
8.8.1 依赖传递

只添加了一个 struts2-core 依赖，发现项目中出现了很多 jar，
这种情况 叫 依赖传递

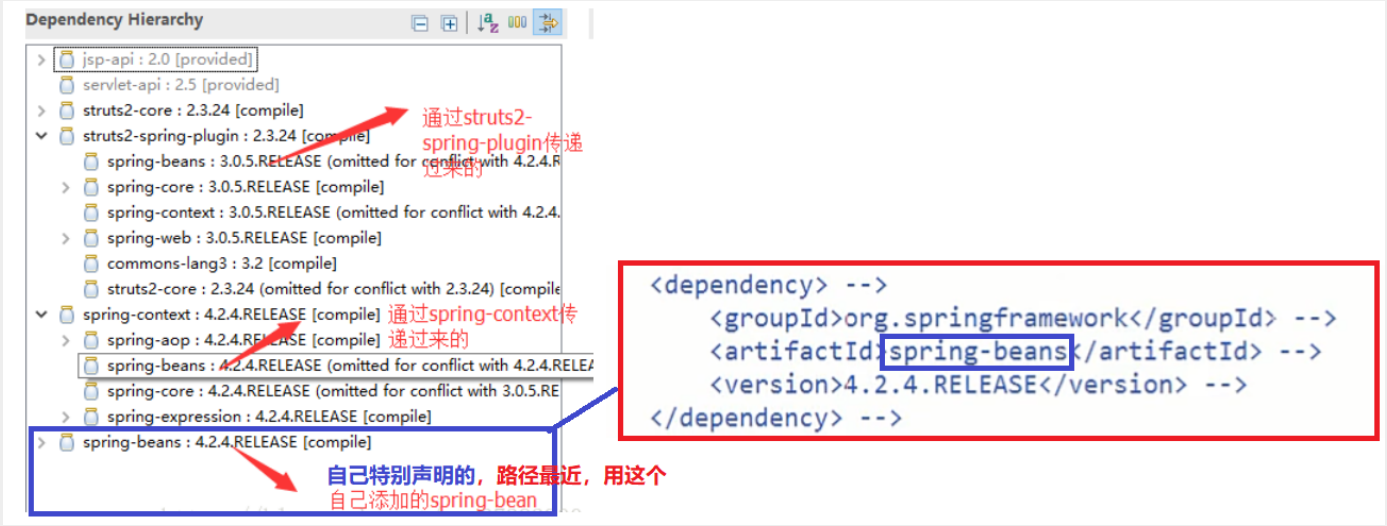


8.8.2 版本冲突

8.8.2.1 第一声明优先原则



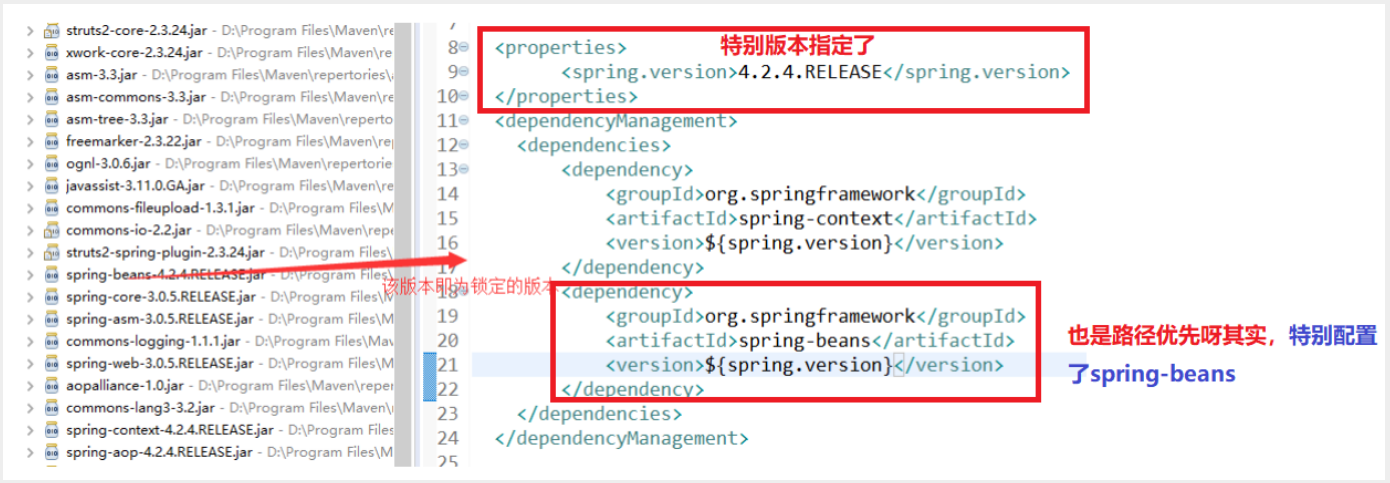
8.8.2.2 路径优先原则



8.8.2.3 排除原则


```
<!--  spring-beans.3.0.5 -->
    <dependency>
        <groupId>org.apache.struts</groupId>
        <artifactId>struts2-spring-plugin</artifactId>
        <version>2.3.24</version>
        <exclusions>
          <exclusion>
                <groupId>org.springframework</groupId>
                <artifactId>spring-beans</artifactId>
          </exclusion>
        </exclusions>
    </dependency>
```

8.8.2.4 版本锁定原则




```
<properties>
      <spring.version>4.2.4.RELEASE</spring.version>
</properties>
<dependencyManagement>
  <dependencies>
      <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>
  </dependencies>
</dependencyManagement>
```

9. Maven分模块开发

10. Maven私服