**项目管理工具**

**maven**

# maven介绍

## maven是什么

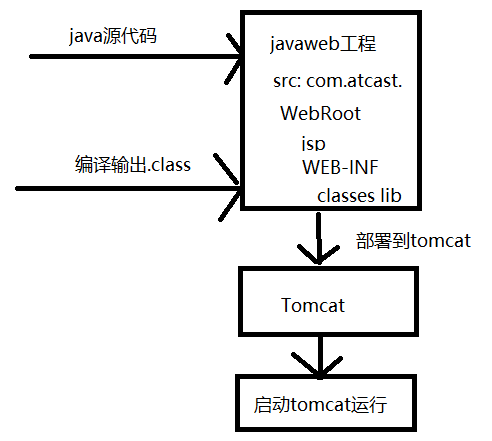
maven翻译为“专家”，“内行”。Maven是Apache下的一个纯java开发的开源项目，它是一个项目管理工具，使用maven对java项目进行构建、赖依管理。当前使用Maven的项目在持续增长。

## 什么是项目构建

项目构建是一个项目从：源代码、编译、测试、打包、部署、运行的过程

### 传统项目构建过程

传统的使用eclipse构建项目的过程如下：



构建过程如下：

1）在eclipse中创建一个java web工程

2）在工程中编写源代码及配置文件等

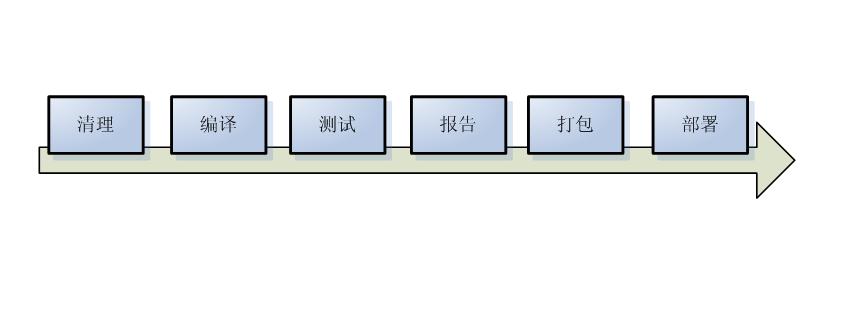
3）对源代码进行编译，java文件编译成class文件

4）执行Junit单元测试

5）将工程打成war包部署至tomcat运行

### maven项目构建过程

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工程可打成war包

maven工程构建的优点：

1、一个命令完成构建、运行，方便快捷。

2、maven对每个构建阶段进行规范，非常有利于大型团队协作开发。

## 什么是依赖管理

什么是依赖？一个java项目可能要使用一些第三方的jar包才可以运行，那么我们说这个java项目依赖了这些第三方的jar包。

举个例子：一个crm系统，它的架构是SSH框架，该crm项目依赖SSH框架，具体它依赖的Hibernate、Spring、Struts2。

什么是依赖管理？就是对项目所有依赖的jar包进行规范化管理。

### 传统项目的依赖管理

传统的项目工程要管理所依赖的jar包完全靠人工进行，程序员从网上下载jar包添加到项目工程中，如下图：程序员手工将Hibernate、struts2、spring的jar添加到工程中的WEB-INF/lib目录下。



手工拷贝jar包添加到工程中的问题是：

1、没有对jar包的版本统一管理，容易导致版本冲突。

2、从网上找jar包非常不方便，有些jar找不到。

3、jar包添加到工程中导致工程过大。

### maven项目的依赖管理

maven项目管理所依赖的jar包不需要手动向工程添加jar包，只需要在pom.xml（maven工程的配置文件）添加jar包的坐标，自动从maven仓库中下载jar包、运行，如下图：



使用maven依赖管理添加jar的好处：

1、通过pom.xml文件对jar包的版本进行统一管理，可避免版本冲突。

2、maven团队维护了一个非常全的maven仓库，里边包括了当前使用的jar包，maven工程可以自动从maven仓库下载jar包，非常方便。

## 使用maven的好处

通过上边介绍传统项目和maven项目在项目构建及依赖管理方面的区域，maven有如下的好处：

1、一步构建

maven对项目构建的过程进行标准化，通过一个命令即可完成构建过程。

2、依赖管理

maven工程不用手动导jar包，通过在pom.xml中定义坐标从maven仓库自动下载，方便且不易出错。

3、maven的跨平台，可在window、linux上使用。

4、maven遵循规范开发有利于提高大型团队的开发效率，降低项目的维护成本，大公司都会考虑使用maven来构建项目。

# maven安装

## 下载安装

* 下载

从该网站 <http://maven.apache.org/download.cgi> 下载maven3.3.9 版本

* 解压

将maven解压到一个不含有中文和空格的目录中。

bin目录 mvn.bat （以run方式运行项目）、 mvnDebug.bat（以debug方式运行项目 ）

boot目录 maven运行需要类加载器

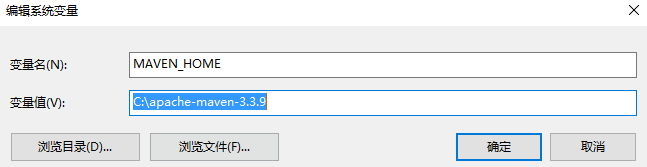
conf目录 settings.xml 整个maven工具核心配置文件

lib目录 maven运行依赖jar包

## 环境变量配置

电脑上需安装java环境，安装JDK1.7 + 版本 （将JAVA\_HOME/bin 配置环境变量path ）

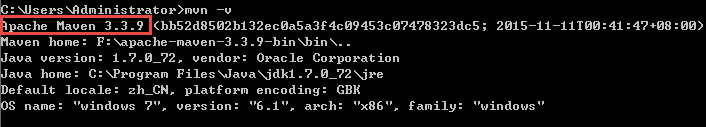
配置 MAVEN\_HOME



将 %MAVEN\_HOME%/bin 加入环境变量 path



通过 mvn -v命令检查 maven是否安装成功，看到maven的版本为3.3.9及java版本为1.7即为安装成功。

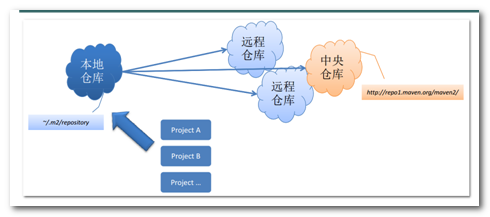


## maven仓库

### maven仓库的作用

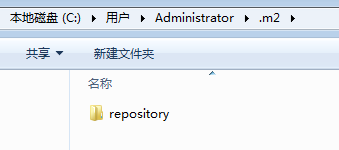
maven的工作需要从仓库下载一些jar包，如下图所示，本地的项目A、项目B等都会通过maven软件从远程仓库（可以理解为互联网上的仓库）下载jar包并存在本地仓库，本地仓库就是本地文件夹，当第二次需要此jar包时则不再从远程仓库下载，因为本地仓库已经存在了，可以将本地仓库理解为缓存，有了本地仓库就不用每次从远程仓库下载了。

下图描述了maven中仓库的类型：



* 本地仓库 ：用来存储从远程仓库或中央仓库下载的插件和jar包，项目使用一些插件或jar包，优先从本地仓库查找

默认本地仓库位置在 ${user.dir}/.m2/repository，${user.dir}表示windows用户目录。



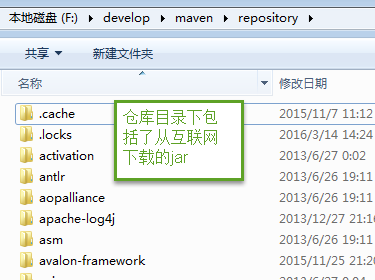
* 远程仓库：如果本地需要插件或者jar包，本地仓库没有，默认去远程仓库下载。

远程仓库可以在互联网内也可以在局域网内。

* 中央仓库 ：在maven软件中内置一个远程仓库地址http://repo1.maven.org/maven2 ，它是中央仓库，服务于整个互联网，它是由Maven团队自己维护，里面存储了非常全的jar包，它包含了世界上大部分流行的开源项目构件。

### 配置本地仓库

需要配置提供的本地仓库，将 “repository.rar”解压至自己的电脑上



在MAVE\_HOME/conf/settings.xml文件中配置本地仓库位置：

C:\apache-maven-3.0.5\conf\settings.xml

|  |
| --- |
| <localRepository>D:\maven\repository</localRepository>  …………  <!—配置从阿里提供的远程仓库-->  <mirrors>  <mirror>  <id>alimaven</id>  <name>aliyun maven</name>  <url>http://maven.aliyun.com/nexus/content/groups/public/</url>  <mirrorOf>central</mirrorOf>  </mirror>  </mirrors> |

# 项目构建

## m2e插件安装配置

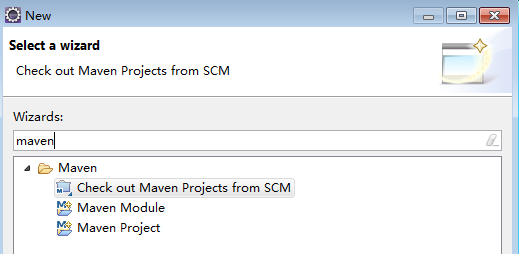
### eclipse与maven

通过入门程序中命令行的方式使用maven工作效率不高，可以在eclipse开发工具中集成maven软件，eclipse是一个开发工具，maven是一个项目管理工具，maven有一套项目构建的规范，在eclipse集成maven软件，最终通过eclipse创建maven工程。

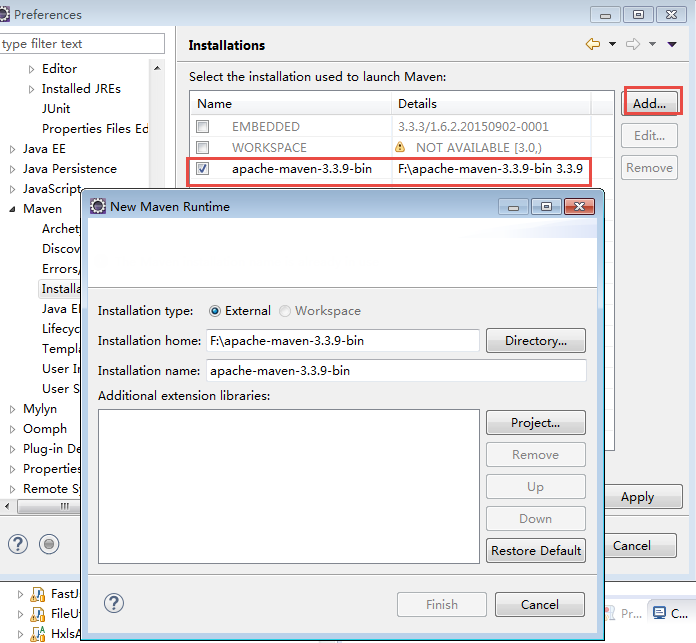
### myeclipse对maven的支持

MyEclipse Professional 2014自带maven插件不用单独安装。

在新建窗口中可看到maven项目的创建项表示maven插件已存在，如下图：

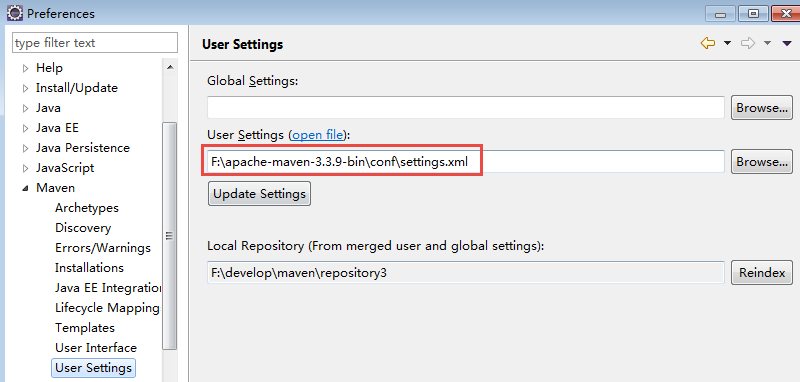


### 指定maven安装目录



### User Setting配置

在eclipse中配置使用的maven的setting.xml文件，使用maven安装目录下的setting.xml文件。

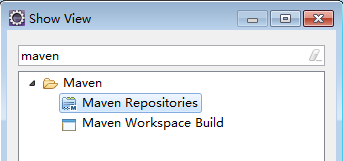


注意：如果修改了 setting.xml文件需要点击上图中的“update settings”按钮对本地仓库重建索引，点击“Reindex”。

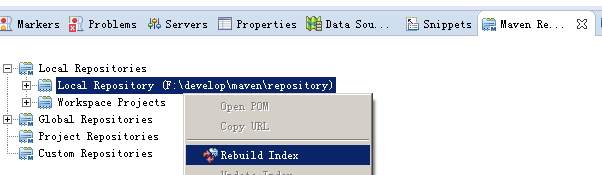
### eclipse浏览仓库

maven配置完成需要测试在eclipse中是否可以浏览maven的本地仓库，如果可以正常浏览maven本地仓库则说明eclipse集成maven已经完成。

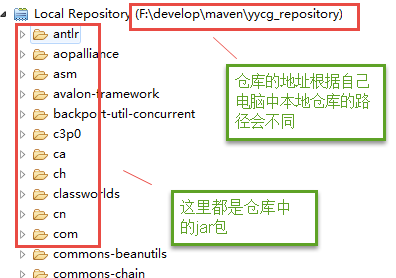
打开eclipse仓库视图，对插件和jar包建立索引



找到Local respository本地仓库项，点击Rebuild index重建索引



重建索引完成点击前边的“+”图标即可查看本地仓库的内容：如下图：



## 定义maven坐标

每个maven工程都需要定义本工程的坐标，坐标是maven对jar包的身份定义，比如：入门程序的坐标定义如下：

<!--项目名称，定义为组织名+项目名，类似包名-->

<groupId>cn.atcast.maven</groupId>

<!-- 模块名称 -->

<artifactId>maven-first</artifactId>

<!-- 当前项目版本号，snapshot为快照版本即非正式版本，release为正式发布版本 -->

<version>0.0.1-SNAPSHOT</version>

<packaging > ：打包类型

jar：执行package会打成jar包

war：执行package会打成war包

pom ：用于maven工程的继承，通常父工程设置为pom

## 构建web工程

### 需求

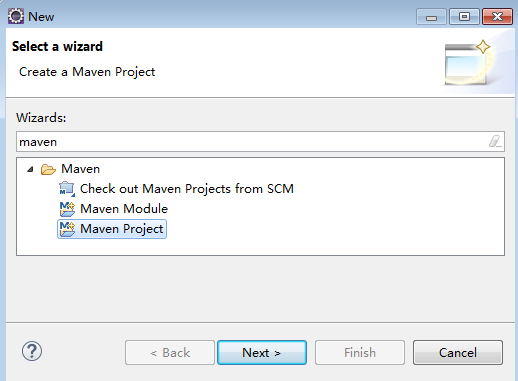
创建一个web工程，实现入门程序的功能。

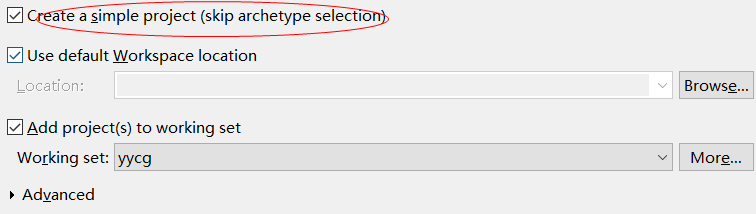
1）添加index.jsp，输出hello world

2）添加一个servlet转发到jsp页面。

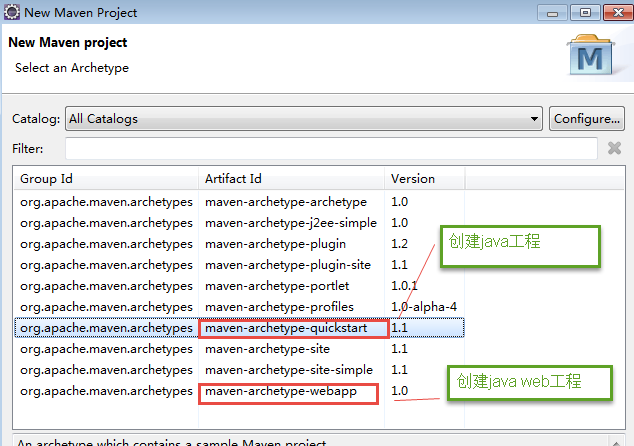
### 第一步创建maven工程

选择 Maven Project





点击“next”会进入骨架选择页面，如果eclipse中配置本地仓库正确则显示出骨架：

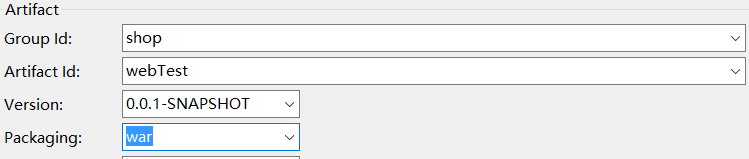


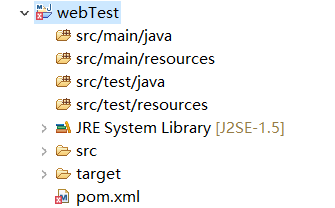
选择webapp会创建一个maven工程（java web工程）

选择quickstart会创建一个maven工程（java工程）

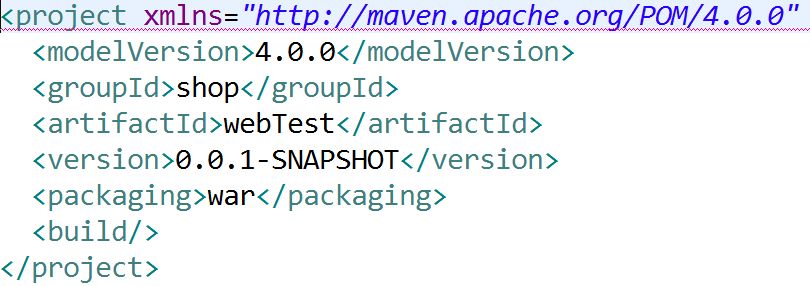
如果使用骨架创建会导致项目的包不完整，不建议。

### 第二步定义坐标

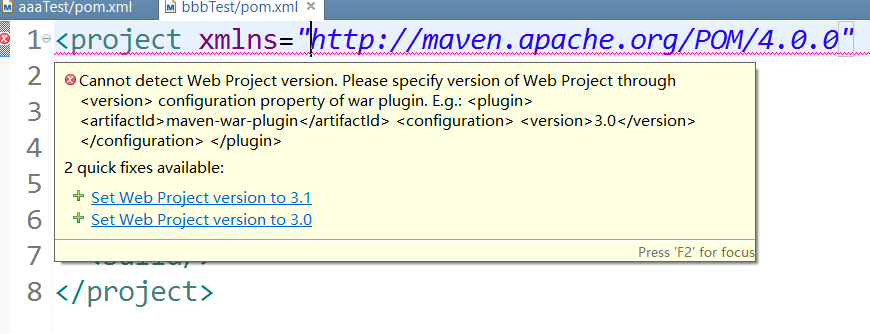




pom.xml报错



解决



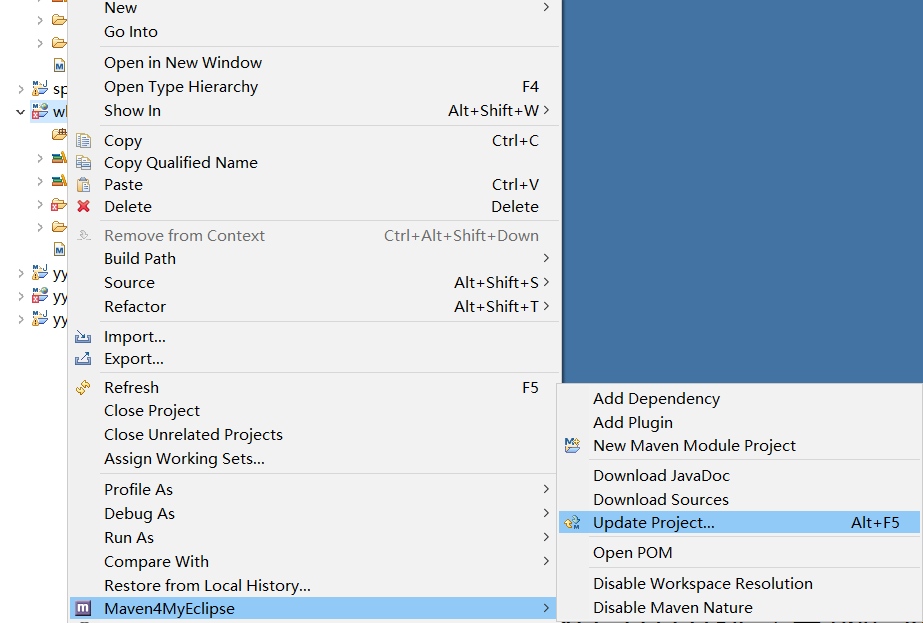
### 第三步设置编译版本

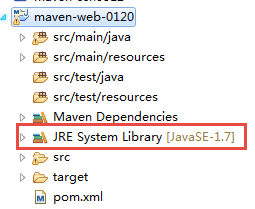
查看上边工程的编译版本为1.5，本教程 使用jdk1.7，需要设置编译版本为1.7，这里需要使用maven的插件来设置：

在pom.xml中加入：

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>shop</groupId>  <artifactId>webTest</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <build>  <plugins>  <plugin>  <artifactId>maven-war-plugin</artifactId>  <configuration>  <version>3.1</version>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <configuration>  <path>/</path>  <port>8081</port>  <server>tomcat7</server>  </configuration>  </plugin>  </plugins>  </build>  </project> |

执行update project，查看编译版本为1.7：



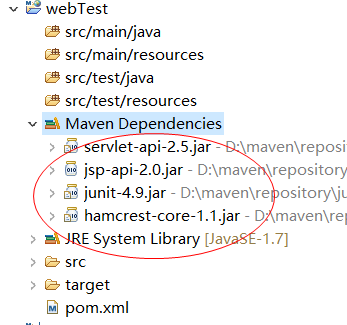


### 添加servelt/jsp的包

在maven工程中添加jar的方式是需要在pom.xml中添加servlet/jsp的坐标，maven自动从创建下载servlet/jsp的jar包

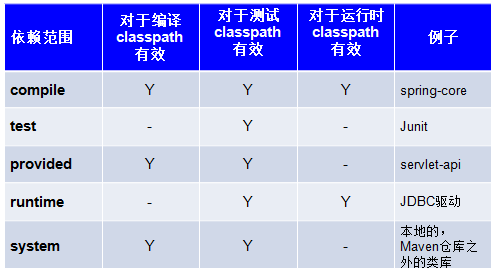
在pom.xml中加入：

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>shop</groupId>  <artifactId>webTest</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <build>  <plugins>  <plugin>  <artifactId>maven-war-plugin</artifactId>  <configuration>  <version>3.1</version>  </configuration>  </plugin>  <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>  <dependencies>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>javax.servlet.jsp</groupId>  <artifactId>jsp-api</artifactId>  <version>2.0</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.9</version>  <scope>test</scope>  </dependency>  </dependencies>  </project> |



依赖范围包括：

* compile：编译范围，指A在编译时依赖B，此范围为默认依赖范围。编译范围的依赖会用在编译、测试、运行，由于运行时需要所以编译范围的依赖会被打包。
* provided：provided依赖只有在当JDK或者一个容器已提供该依赖之后才使用， provided依赖在编译和测试时需要，在运行时不需要，比如：servlet api被tomcat容器提供。
* runtime：runtime依赖在运行和测试系统的时候需要，但在编译的时候不需要。比如：jdbc的驱动包。由于运行时需要所以runtime范围的依赖会被打包。
* test：test范围依赖 在编译和运行时都不需要，它们只有在测试编译和测试运行阶段可用，比如：junit。由于运行时不需要所以test范围依赖不会被打包。
* system：system范围依赖与provided类似，但是你必须显式的提供一个对于本地系统中JAR文件的路径，需要指定systemPath磁盘路径，system依赖不推荐使用。



### 第四步定义web.xml

在src/webapp中添加WEB-INF目录，并在其下加入web.xml文件，内容为：

web.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"*  xmlns:web=*"http://java.sun.com/xml/ns/javaee"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>  <servlet>  <description></description>  <display-name>HelloServlet</display-name>  <servlet-name>HelloServlet</servlet-name>  <servlet-class>cn.atcast.web.HelloServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>HelloServlet</servlet-name>  <url-pattern>/HelloServlet</url-pattern>  </servlet-mapping>  </web-app> |

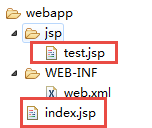
### 第五步编写servlet

在src/main/java中创建ServletTest

cn.atcast.web包下创建HelloServlet.java

|  |
| --- |
| **package** cn.atcast.web;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** HelloServlet **extends** HttpServlet {    **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  response.setContentType("text/html;charset=UTF-8");  response.getWriter().write("hello world");  }  **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

### 第六步编写jsp



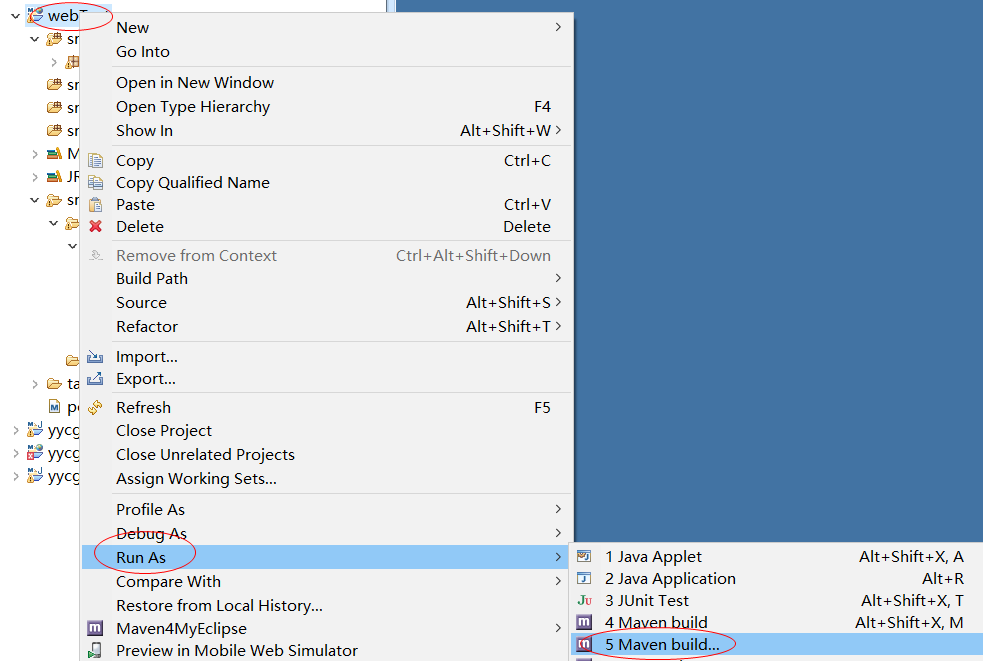
test.jsp的内容如下：

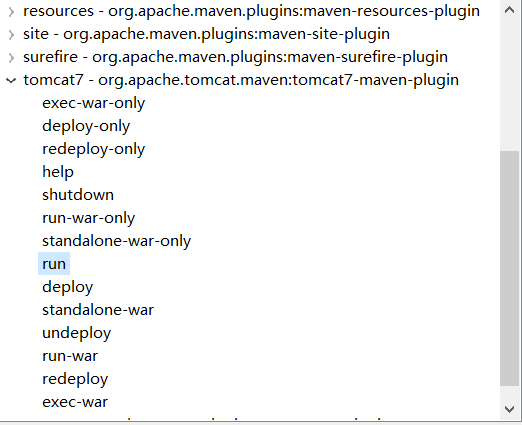
|  |
| --- |
| <body>  运行servlet <br>  </body> |

index.jsp的内容如下：

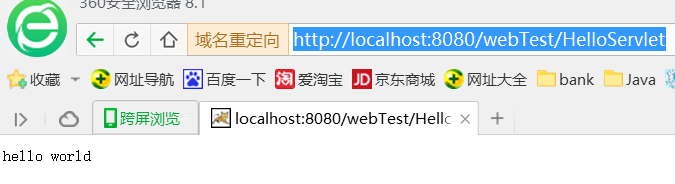
|  |
| --- |
| <body>  <h1>maven版本管理 </h1>  </body> |

### 运行





测试HelloServlet



## eclipse下使用maven命令

在eclipse下测试命令

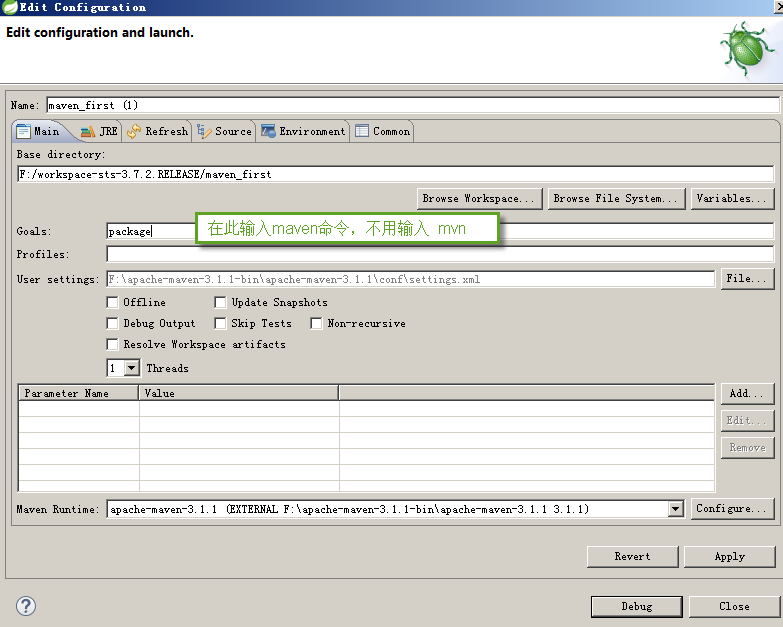
Maven clean 清理target目录

Maven test 执行单元测试

Maven install将工程打包后发布到本地仓库

Maven build 使用之前操作过的命令

Maven build … 手动输入命令内容，如下图：



* 编译命令 mvn compile

编译后 .class文件在 target/classes 下 （这个命令只会对java源程序编译， 不会编译测试代码 ， 编译测试类 mvn test-compile , 编译后.class 文件在 target\test-classes ）

* 测试命令 mvn test

执行所有测试用例方法， 重新编译

在src/test/java下建立包mail,其下建立测试文件mailTest.java

|  |
| --- |
| **package** mail;  **import** java.util.Properties;  **import** org.junit.Test;  **public** **class** mailTest {  @Test  **public** **void** testJavaMail() **throws** Exception{  System.*out*.println(4/0);  }  } |

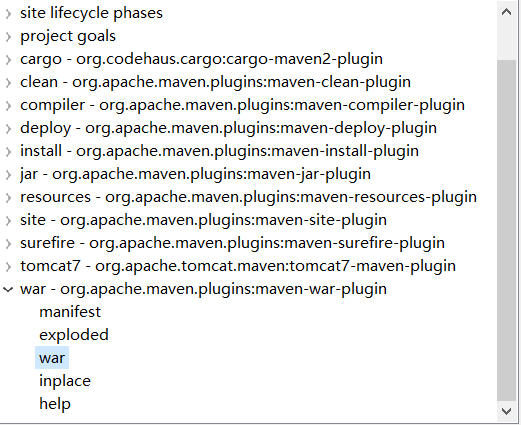
* 清除命令 mvn clean

清除target目录 （清除所有编译结果或者打包结果 ），清理后编译。

* 打包命名 mvn package

java项目生成 jar包， web项目生成war包

默认生成jar包名称 ： artifactId-version.jar



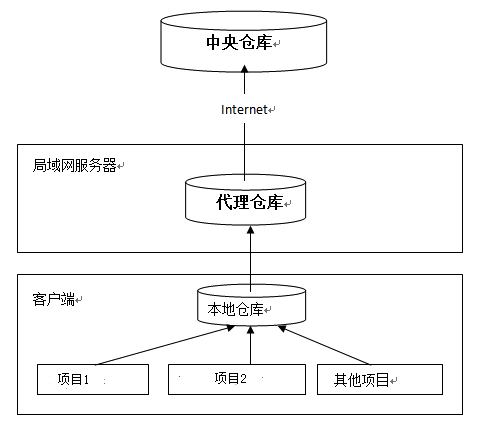
# Maven私服

问题：项目组编写了一个通用的工具类，其它项目组将类拷贝过去使用，当工具类修改bug后通过邮件发送给各各项目组，这种分发机制不规范可能导致工具类版本不统一。

解决方案：项目组将写的工具类通过maven构建，打成jar，将jar包发布到公司的maven仓库中（私服），公司其它项目通过maven依赖管理从仓库自动下载jar包。

公司在自己的局域网内搭建自己的远程仓库服务器，称为私服，私服服务器即是公司内部的maven远程仓库，每个员工的电脑上安装maven软件并且连接私服服务器，员工将自己开发的项目打成jar并发布到私服服务器，其它项目组从私服服务器下载所依赖的构件（jar）。

私服还充当一个代理服务器，当私服上没有jar包会从互联网中央仓库自动下载，如下图：



私服作用：

1、将自己的工程打包jar发布到私服

2、从私服下载jar包到自己的工程中

# 依赖版本冲突

依赖版本冲突：当一个项目依赖的构件比较多时，它们相互之前存在依赖，当你需要对依赖版本统一管理时如果让maven自动来处理可能并不能如你所愿，如下例子：

（传递依赖：当A 依赖B、B依赖C，在A中导入B后会自动导入C，C是A的传递依赖，如果C依赖D则D也可能是A的传递依赖。）

同时加入以下依赖，观察依赖：

<!-- struts2-spring-plugin依赖spirng-beans-3.0.5 -->

<dependency>

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

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

<version>2.3.24</version>

</dependency>

<!-- spring-context依赖spring-beans-4.2.4 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.2.4.RELEASE</version>

</dependency>

org.apache.struts依赖spirng-beans-3.0.5，spring-context依赖spring-beans-4.2.4，但是发现spirng-beans-3.0.5加入到工程中，而我们希望spring-beans-4.2.4加入工程。

### 依赖调解原则

maven自动按照下边的原则调解：

* **声明者优先原则**

在pom文件定义依赖，先声明的依赖为准。

测试：

如果将上边struts-spring-plugins和spring-context顺序颠倒，系统将导入spring-beans-4.2.4。

分析：

由于spring-context在前边，以spring-context依赖的spring-beans-4.2.4为准，所以最终spring-beans-4.2.4添加到了工程中。

### 排除依赖

上边的问题也可以通过排除依赖方法辅助依赖调解，如下：

比如在依赖struts2-spring-plugin的设置中添加排除依赖，排除spring-beans，

下边的配置表示：依赖struts2-spring-plugin，但排除struts2-spring-plugin所依赖的spring-beans。

<!-- struts2-spring-plugin依赖spirng-beans-3.0.5 -->

<dependency>

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

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

<version>2.3.24</version>

<!-- 排除 spring-beans-->

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

</exclusion>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

</exclusion>

</exclusions>

</dependency>

### 锁定版本

面对众多的依赖，有一种方法不用考虑依赖路径、声明优化等因素可以采用直接锁定版本的方法确定依赖构件的版本，版本锁定后则不考虑依赖的声明顺序或依赖的路径，以锁定的版本的为准添加到工程中，此方法在企业开发中常用。

首先父工程中pom.xml文件添加

如下的配置是锁定了spring-beans和spring-context的版本：



注意：在工程中锁定依赖的版本并不代表在工程中添加了依赖，如果工程需要添加锁定版本的依赖则需要单独添加<dependencies></dependencies>标签，如下：

上边添加的依赖并没有指定版本，原因是已在<dependencyManagement>中锁定了版本，所以在<dependency>下不需要再指定版本。

# 总结

## maven仓库

1、maven仓库的类型有哪些？

2、maven工程查找仓库的流程是什么？

3、本地仓库如何配置？

## 常用的maven命令

常用 的maven命令包括：

compile：编译

clean：清理

test：测试

package：打包

install：安装

## 坐标定义

在pom.xml中定义坐标，内容包括：groupId、artifactId、version，详细内容如下：

<!--项目名称，定义为组织名+项目名，类似包名-->

<groupId>cn.itcast.maven</groupId>

<!-- 模块名称 -->

<artifactId>maven-first</artifactId>

<!-- 当前项目版本号，snapshot为快照版本即非正式版本，release为正式发布版本 -->

<version>0.0.1-SNAPSHOT</version>

<packaging > ：打包类型

jar：执行package会打成jar包

war：执行package会打成war包

pom ：用于maven工程的继承，通常父工程设置为pom

## pom基本配置

pom.xml是Maven项目的核心配置文件，位于每个工程的根目录，基本配置如下：

<project > ：文件的根节点 .  
<modelversion > ： pom.xml使用的对象模型版本  
<groupId > ：项目名称，一般写项目的域名  
<artifactId > ：模块名称，子项目名或模块名称  
<version > ：产品的版本号 .

<packaging > ：打包类型，一般有jar、war、pom 等   
<name > ：项目的显示名，常用于 Maven 生成的文档。   
<description > ：项目描述，常用于 Maven 生成的文档

<dependencies> ：项目依赖构件配置，配置项目依赖构件的坐标

<build> ：项目构建配置，配置编译、运行插件等。