**项目管理工具**

**maven**

# 学习目标

第一天：

1、会使用maven构建项目的命令

2、会使用maven构建java项目和java web项目

3、构建maven-web工程整合struts2成功

第二天：

1、依赖管理--传递依赖 版本冲突处理

在web工程中加入struts-spring，通过传递依赖将spring加入进来。

通过锁定版本解决版本冲突。

2.在web的单个工程中实现SSH整合。

3、分模块构建工程

4、会向私服上传jar包。会从私服下载jar包

实现将dao上传私服

从私服下载，统一：在setting.xml中配置仓库

实现：从私服下载dao

1）将dao工程关闭

2）把本地仓库的dao删除

# 教学计划

|  |  |  |  |
| --- | --- | --- | --- |
| 课时 | 章节 | 知识点 | 备注 |
| 一 | Maven介绍  Maven安装  入门程序-入门工程构建 | Maven介绍 | 了解 |
| Maven安装与配置 | 重点 |
| Maven项目工程目录约定 | 重点 |
| 二 | 入门程序--Maven仓库  入门程序--常用的maven命令  Maven概念模型 | Maven仓库 | 重点 |
| 常用的maven命令 | 重点 |
| Maven概念模型 | 了解 |
| 三 | 项目构建 | M2e插件安装配置 | 重点 |
| 坐标定义 | 重点 |
| 构建web工程 | 重点 |
| Eclipse下maven命令的使用 | 重点 |
| 生命周期 | 了解 |
| 四 | 依赖管理  Tomcat插件 | 添加依赖 | 重点 |
| 依赖范围 | 重点 |
| Tomcat插件 | 重点 |
|  |  |
|  |  |
|  |  |

# 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、依赖管理。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编写源代码到编译、测试、运行、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

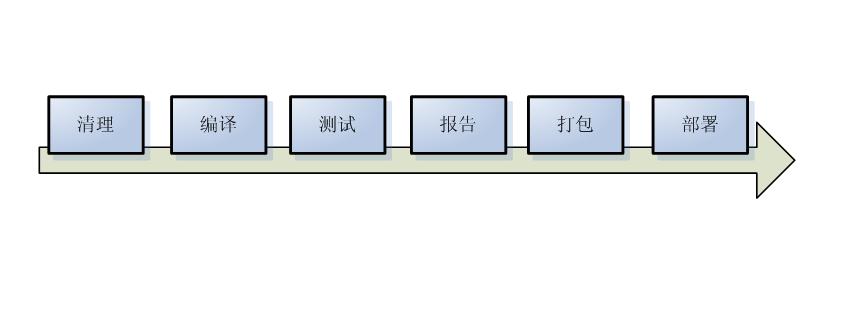
3）对源代码进行编译，java文件编译成class文件

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工程可以打成war包

运行一个maven工程（web工程）需要一个命令：tomat:run

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 ）

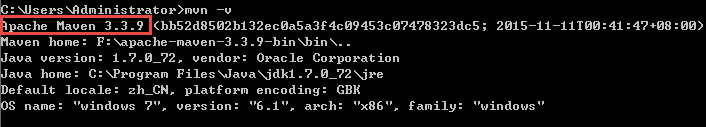
配置 MAVEN\_HOME



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。

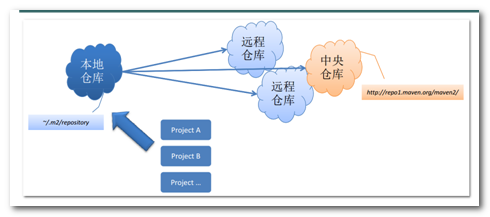


## maven仓库

### maven仓库的作用

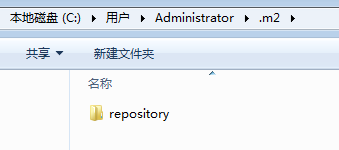
maven的工作需要从仓库下载一些jar包，如下图所示，本地的项目A、项目B等都会通过maven软件从远程仓库（可以理解为互联网上的仓库）下载jar包并存在本地仓库，本地仓库 就是本地文件夹，当第二次需要此jar包时则不再从远程仓库下载，因为本地仓库已经存在了，可以将本地仓库理解为缓存，有了本地仓库就不用每次从远程仓库下载了。

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。



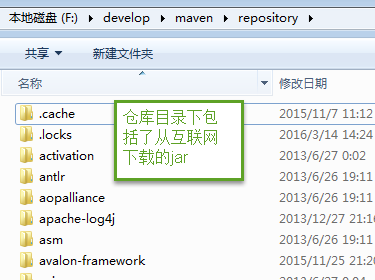
* 远程仓库：如果本地需要插件或者jar包，本地仓库没有，默认去远程仓库下载。

远程仓库可以在互联网内也可以在局域网内。

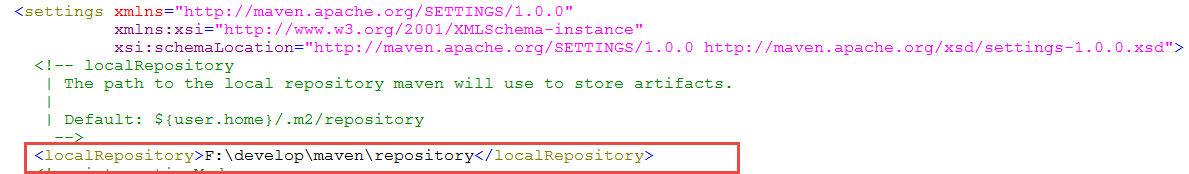
* 中央仓库 ：在maven软件中内置一个远程仓库地址http://repo1.maven.org/maven2 ，它是中央仓库，服务于整个互联网，它是由Maven团队自己维护，里面存储了非常全的jar包，它包含了世界上大部分流行的开源项目构件。

### 配置本地仓库

本课程是在无网的状态下学习，需要配置老师提供的本地仓库，将 “repository.rar”解压至自己的电脑上，本教程解压在F:\develop\maven\repository



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



### 全局setting与用户setting

maven仓库地址、私服等配置信息需要在setting.xml文件中配置，分为全局配置和用户配置。

在maven安装目录下的有 conf/setting.xml文件，此setting.xml文件用于maven的所有project项目，它作为maven的全局配置。

如需要个性配置则需要在用户配置中设置，用户配置的setting.xml文件默认的位置在：${user.dir} /.m2/settings.xml目录中,${user.dir} 指windows 中的用户目录。

maven会先找用户配置，如果找到则以用户配置文件为准，否则使用全局配置文件。



# 入门程序

## 学习目标

1、说出maven项目工程的目录结构是什么

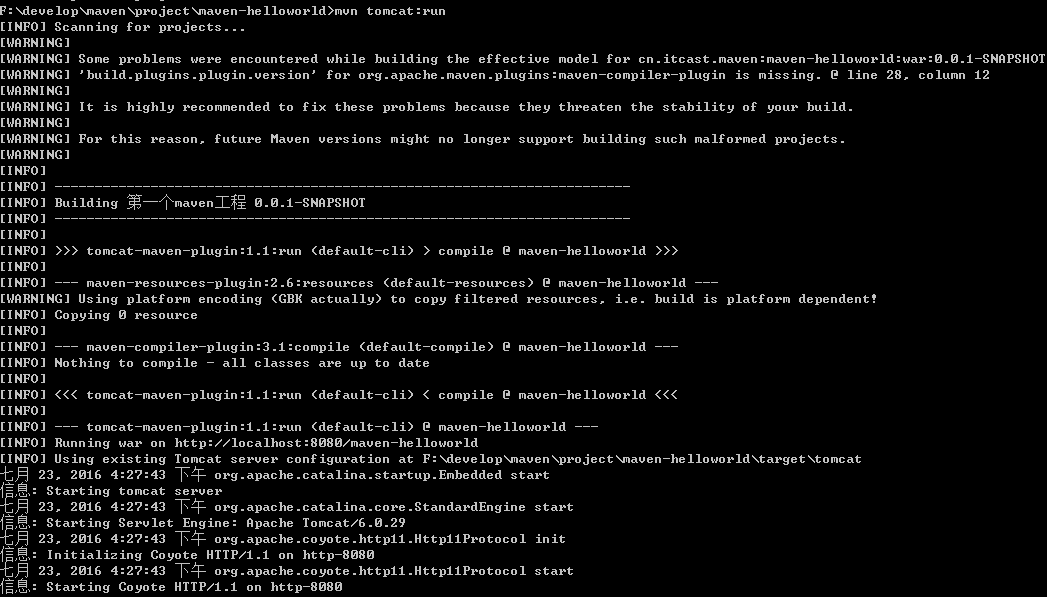
2、在cmd命令行下测试maven的常用命令

## maven工程运行演示

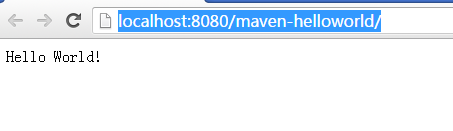
通过使用maven提供的命令来运行maven工程，体会maven构建工程的优点。

### 运行web工程

进入maven工程目录（当前目录有pom.xml文件），运行tomcat:run命令。

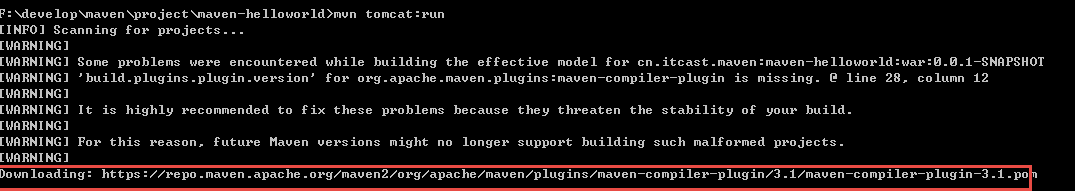


根据上边的提示信息，通过浏览器访问：http://localhost:8080/maven-helloworld/



### 问题处理

如果本地仓库配置错误会报下边的错误



分析：

maven工程运行先从本地仓库找jar包，本地仓库没有再从中央仓库找，上边提示downloading。。。表示 从中央仓库下载jar，由于本地没有联网，报错。

解决：

在maven安装目录的conf/setting.xml文件中配置本地仓库，参考：“maven安装/配置本地仓库章节”。

## Maven项目工程目录约定

使用maven创建的工程我们称它为maven工程，maven工程具有一定的目录规范，如下：

src/main/java —— 存放项目的.java文件

src/main/resources —— 存放项目资源文件，如spring, hibernate配置文件

src/test/java —— 存放所有单元测试.java文件，如JUnit测试类

src/test/resources —— 测试资源文件

target —— 项目输出位置，编译后的class文件会输出到此目录

pom.xml——maven项目核心配置文件

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

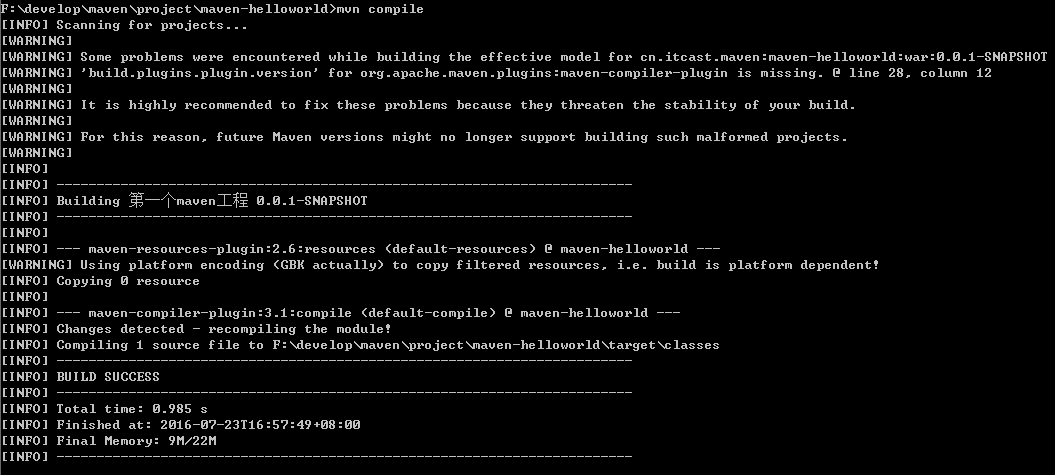
## 常用的maven命令

在cmd的状态下测试

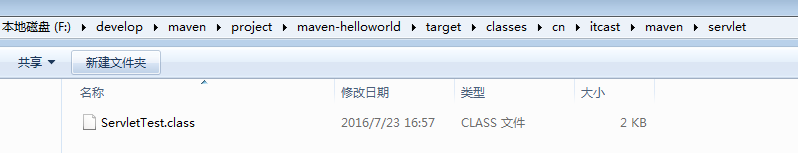
### compile

compile是maven工程的编译命令，作用是将src/main/java下的文件编译为class文件输出到target目录下。

cmd进入命令状态，执行mvn compile，如下图提示成功：



查看 target目录，class文件已生成，编译完成。



### test

test是maven工程的测试命令，会执行src/test/java下的单元测试类。

cmd执行mvn test执行src/test/java下单元测试类，下图为测试结果，运行1个测试用例，全部成功。



### clean

clean是maven工程的清理命令，执行 clean会删除target目录的内容。

### package

package是maven工程的打包命令，对于java工程执行package打成jar包，对于web工程打成war包。

### install

install是maven工程的安装命令，执行install将maven打成jar包或war包发布到本地仓库。

## 生命周期(了解)

### 三套生命周期

maven对项目构建过程分为三套相互独立的生命周期，请注意这里说的是“三套”，而且“相互独立”，这三套生命周期分别是：

Clean Lifecycle 在进行真正的构建之前进行一些清理工作。

Default Lifecycle 构建的核心部分，编译，测试，打包，部署等等。

Site Lifecycle 生成项目报告，站点，发布站点。

### 生命周期的阶段

每个生命周期都有很多阶段，每个阶段对应一个执行命令。

1. **如下是clean生命周期的阶段**

pre-clean 执行一些需要在clean之前完成的工作

clean 移除所有上一次构建生成的文件

post-clean 执行一些需要在clean之后立刻完成的工作

1. **如下是default周期的内容：**

validate

generate-sources

process-sources

generate-resources

process-resources 复制并处理资源文件，至目标目录，准备打包。

compile 编译项目的源代码。

process-classes

generate-test-sources

process-test-sources

generate-test-resources

process-test-resources 复制并处理资源文件，至目标测试目录。

test-compile 编译测试源代码。

process-test-classes

test 使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。

prepare-package

package 接受编译好的代码，打包成可发布的格式，如 JAR 。

pre-integration-test

integration-test

post-integration-test

verify

install 将包安装至本地仓库，以让其它项目依赖。

deploy 将最终的包复制到远程的仓库，以让其它开发人员与项目共享。

1. **如下是site生命周期的阶段**

pre-site 执行一些需要在生成站点文档之前完成的工作

site 生成项目的站点文档

post-site 执行一些需要在生成站点文档之后完成的工作，并且为部署做准备

site-deploy 将生成的站点文档部署到特定的服务器上

### 命令与生命周期的阶段

每个maven命令对应生命周期的某个阶段，例如：mvn clean 命令对应clean生命周期的clean阶段， mvn test 命令对应default生命周期的test阶段。

执行命令会将该命令在的在生命周期当中之前的阶段自动执行，比如：执行mvn clean 命令会自动执行pre-clean和clean两个阶段，mvn test命令会自动执行validate、compile、test等阶段。

注意：执行某个生命周期的某个阶段不会影响其它的生命周期！

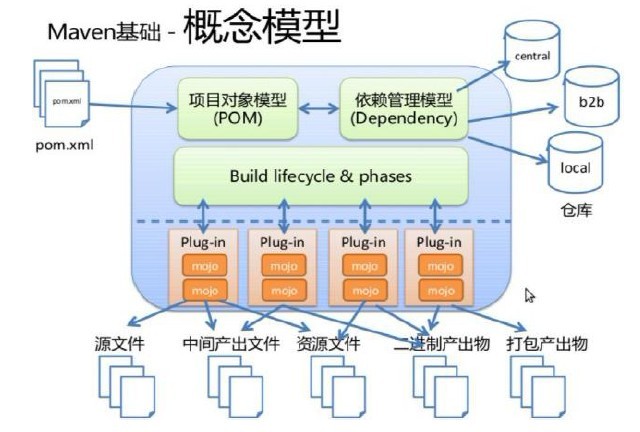
如果要同时执行多个生命周期的阶段可在命令行输入多个命令，中间以空格隔开，例如：

clean package 该命令执行clean生命周期的clean阶段和default生命周期的package阶段。

## maven的概念模型

Maven包含了一个项目对象模型 (Project Object Model)，一组标准集合，一个项目生命周期(Project Lifecycle)，一个依赖管理系统(Dependency Management System)，和用来运行定义在生命周期阶段(phase)中插件(plugin)目标(goal)的逻辑。

下图是maven的概念模型图：



* 项目对象模型 (Project Object Model)

一个maven工程都有一个pom.xml文件，通过pom.xml文件定义项目的坐标、项目依赖、项目信息、插件目标等。

* 依赖管理系统(Dependency Management System)

通过maven的依赖管理对项目所依赖的jar 包进行统一管理。

比如：项目依赖junit4.9，通过在pom.xml中定义junit4.9的依赖即使用junit4.9，如下所示是junit4.9的依赖定义：

<!-- 依赖关系 -->

<dependencies>

<!-- 此项目运行使用junit，所以此项目依赖junit -->

<dependency>

<!-- junit的项目名称 -->

<groupId>junit</groupId>

<!-- junit的模块名称 -->

<artifactId>junit</artifactId>

<!-- junit版本 -->

<version>4.9</version>

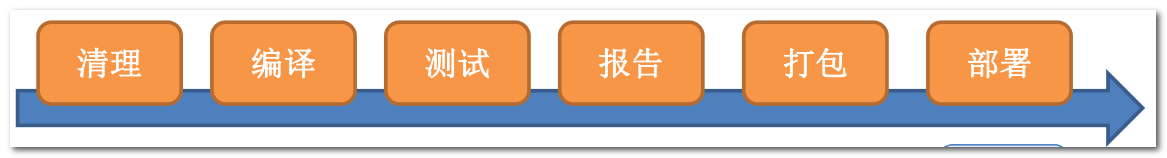
<!-- 依赖范围：单元测试时使用junit -->

<scope>test</scope>

</dependency>

* 一个项目生命周期(Project Lifecycle)

使用maven完成项目的构建，项目构建包括：清理、编译、测试、部署等过程，maven将这些过程规范为一个生命周期，如下所示是生命周期的各各阶段：



maven通过执行一些简单命令即可实现上边生命周期的各各过程，比如执行mvn compile执行编译、执行mvn clean执行清理。

* 一组标准集合

maven将整个项目管理过程定义一组标准，比如：通过maven构建工程有标准的目录结构，有标准的生命周期阶段、依赖管理有标准的坐标定义等。

* 插件(plugin)目标(goal)

maven 管理项目生命周期过程都是基于插件完成的。

# 项目构建

## 学习目标

1、独立在eclipse中安装配置maven插件

2、说出maven坐标的含义

3、使用eclipse创建mavan工程（java工程及java web工程）

4、使用eclipse运行maven的命令

## m2e插件安装配置

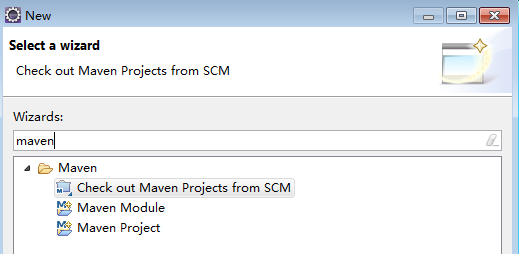
### eclipse与maven

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。

### 插件安装

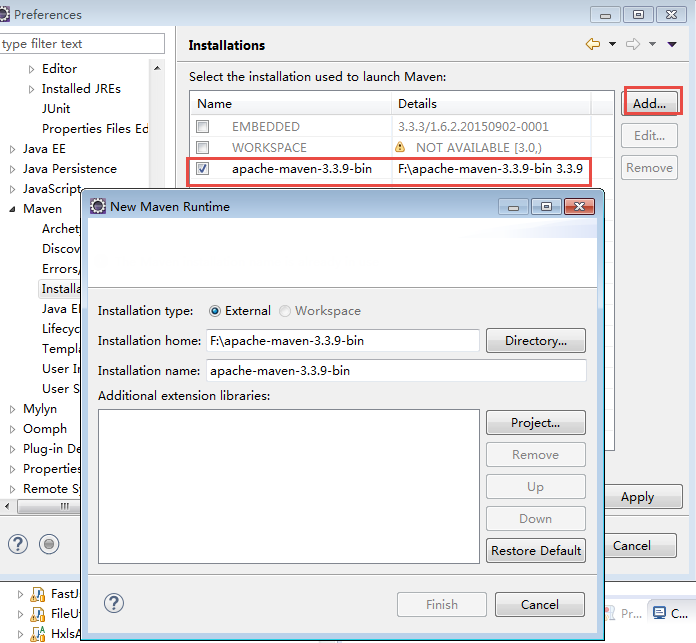
本教程 使用eclipse mars 2版本，此版本自带maven插件不用单独安装。

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：



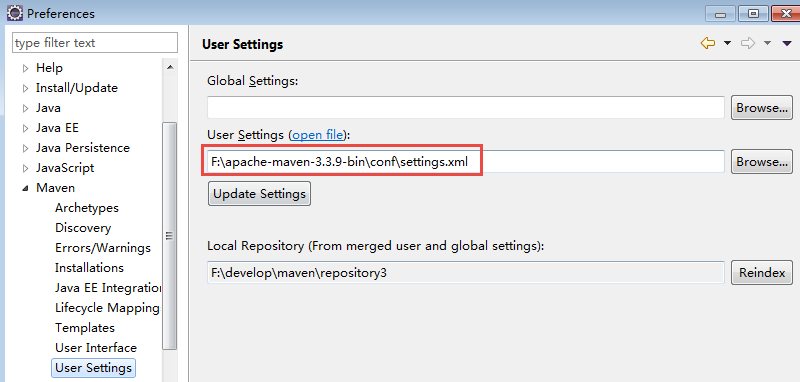
### 指定maven安装目录

一些高版本的eclipse已经内置了maven的安装，下图是eclipse mars2版本中已经内置了maven3.3.3版本，项目为了统一maven的版本不会使用eclipse内置的maven版本，这里我们maven3.3.9。



### 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。

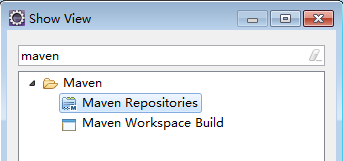


注意：如果修改了 setting.xml文件需要点击上图中的“update settings”按钮对本地仓库重建索引，点击“Reindex”。

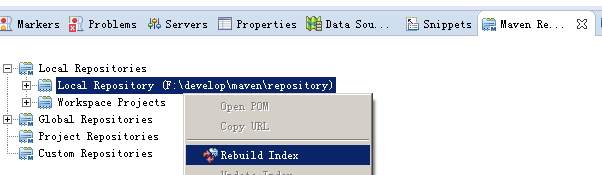
### eclipse浏览仓库

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。

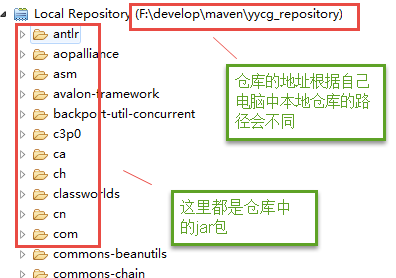
打开eclipse仓库视图，对插件和jar包建立索引



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.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

## 构建web工程

### 需求

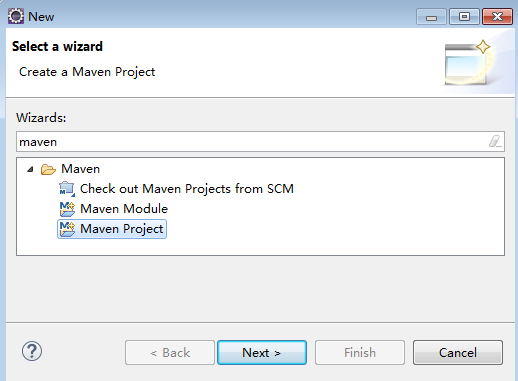
创建一个web工程，实现入门程序的功能。

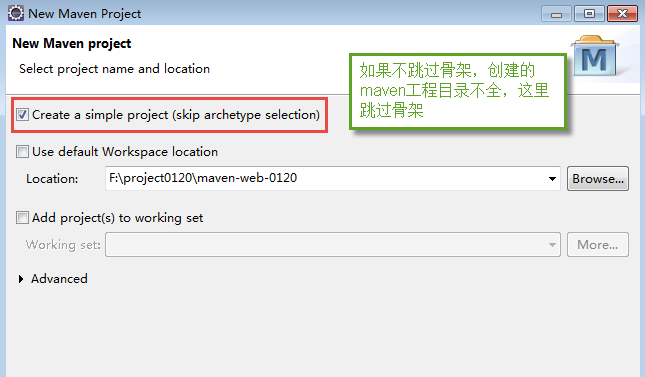
1）添加index.jsp，输出hello world

2）添加一个servlet转发到jsp页面。

### 第一步创建maven工程

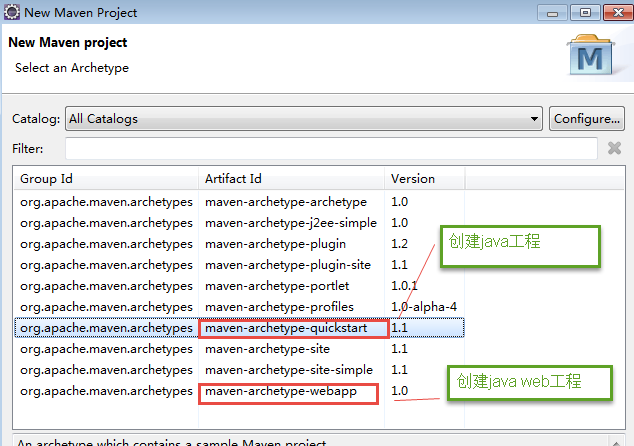
选择 Maven Project





了解一下骨架界面:

当我们不跳过骨架点击“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：

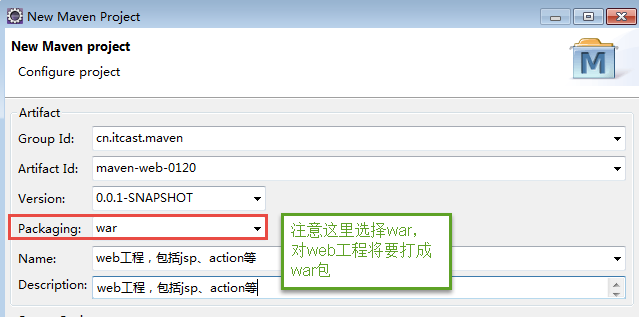


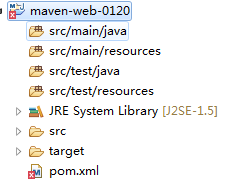
选择webapp会创建一个maven工程（java web工程）

选择quickstart会创建一个maven工程（java工程）

注意：使用骨架的问题是创建的maven工程目录不完整，所以不推荐使用，本教程创建maven工程统一跳过骨架。

### 第二步定义坐标





### 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：

<build>

<plugins>

<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>

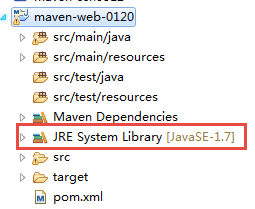
</configuration>

</plugin>

</plugins>

</build>

执行update project，查看编译版本为1.7：



### 第四步定义web.xml

在src/webapp中添加WEB-INF/web.xml文件，内容为：

<?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/web-app\_2\_5.xsd"*

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"*>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

</web-app>

### 第五步编写servlet

在src/main/java中创建ServletTest



**public** **class** ServletTest **extends** HttpServlet {

@Override

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

**this**.doPost(req, resp);

}

@Override

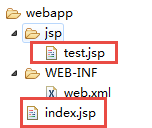
**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

req.getRequestDispatcher("/jsp/test.jsp").forward(req, resp);

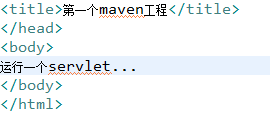
}

}

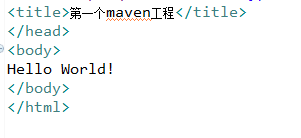
### 第六步编写jsp



test.jsp的内容如下：

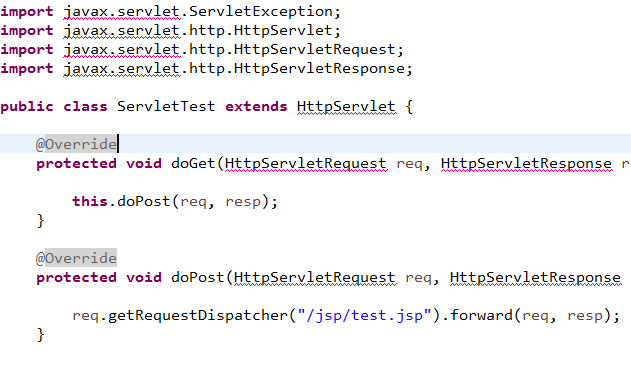


index.jsp的内容如下：



### 第七步添加servlet/jsp的jar包

servlet类中报错，无法解析javax.sevlet....



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

编辑pom.xml，如下：

<!-- 添加servlet-api，jsp-api -->

<dependencies>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

</dependency>

</dependencies>

### 第八步配置servlet

在web.xml中配置servlet，如下所示：

<!-- 配置servlet -->

<servlet>

<servlet-name>servletTest</servlet-name>

<servlet-class>cn.itcast.maven.servlet.ServletTest</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>servletTest</servlet-name>

<url-pattern>/test</url-pattern>

</servlet-mapping>

### 运行

cmd命令行进入工程目录，运行tomcat:run命令

## eclipse下使用maven命令

在eclipse下测试命令

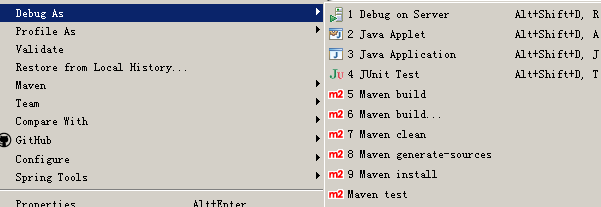
clean

compile

test

package

install



Run as 采用 mvn 命令运行 ，Debug as 采用 mvnDebug 命令调试方式运行（可打断点）

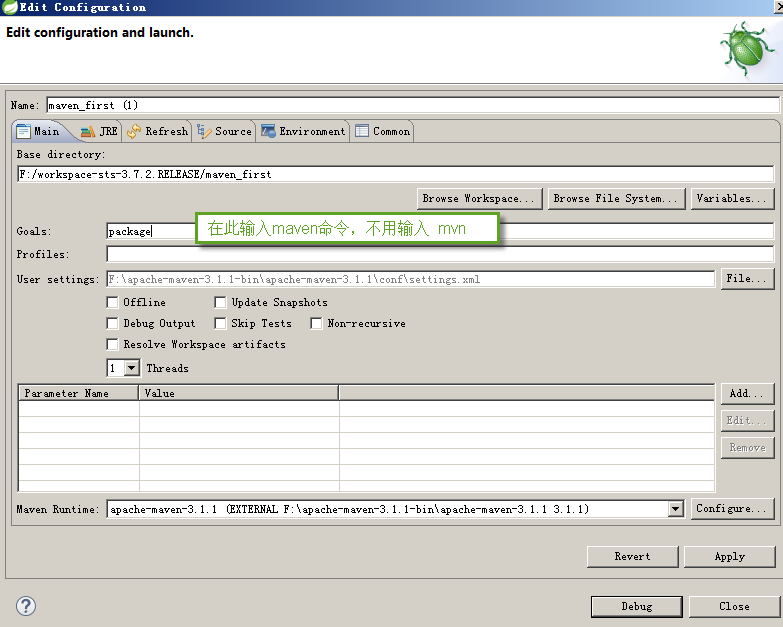
Maven clean 清理target目录

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

* 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

* 安装命令

安装命令 mvn install 将工程打包后发布到本地仓库

---- 安装到仓库/groupId/artifactId/version 目录

# 依赖管理-添加依赖

## 需求

实现web工程整合struts2框架。

## 添加依赖

### dependency

在pom.xml中添加dependency标签，如下：

<dependency>

<groupId><groupId>

<artifactId></artifactId>

<version></version>

</dependency>

参考入门工程的Junit4.9的依赖在web工程的pom.xml中添加dependency

<dependencies>

<!-- 添加junit4.9依赖 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

</dependency>

</dependencies>

### 查找坐标

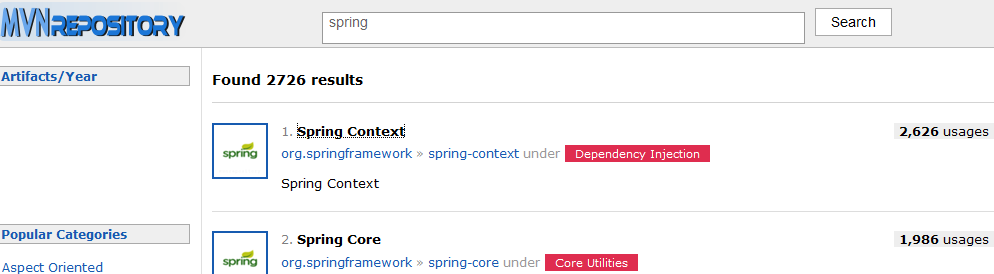
添加依赖需要指定依赖jar包的坐标，但是很多情况我们是不知道jar包的的坐标，可以通过如下方式查询：

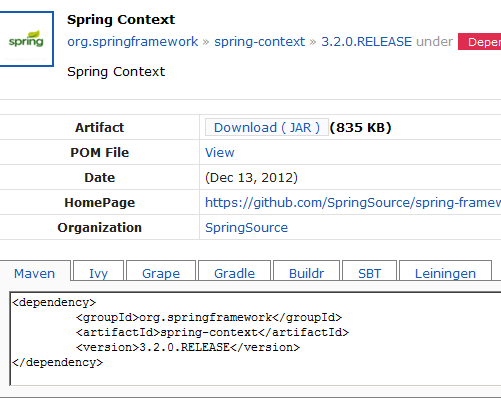
* 方法一：从互联网搜索

http://search.maven.org/

http://mvnrepository.com/

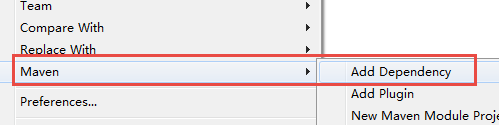
网站搜索示例：

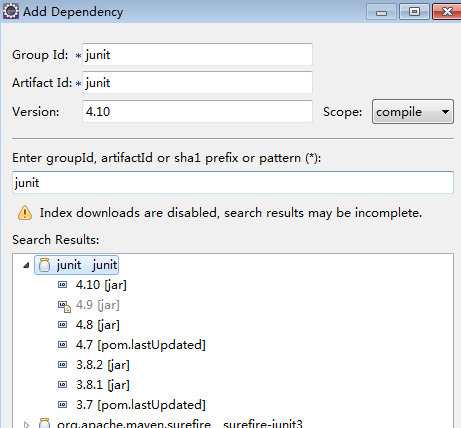




* 方法二：使用maven插件的索引功能

如果在本地仓库有我们要的jar包，可以在pom.xml中邮件添加依赖





## 构建web工程

### 添加struts2依赖

编写web工程的pom.xml文件，添加依赖，如下：

<!-- 依赖struts2 -->

<dependency>

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

<artifactId>struts2-core</artifactId>

<version>2.3.24</version>

</dependency>

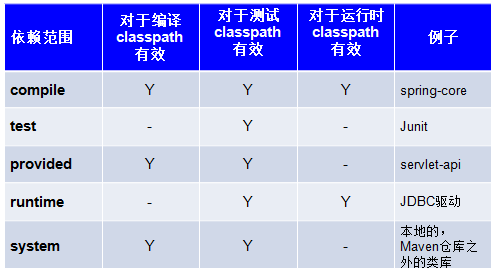
## 依赖范围

### 基本概念

A依赖B，需要在A的pom.xml文件中添加B的坐标，添加坐标时需要指定依赖范围，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。



在maven-web工程中测试各各scop。

测试总结：

* 默认引入 的jar包 ------- compile 【默认范围 可以不写】（编译、测试、运行 都有效 ）
* servlet-api 、jsp-api ------- provided （编译、测试 有效， 运行时无效 防止和tomcat下jar冲突）
* jdbc驱动jar包 ---- runtime （测试、运行 有效 ）
* junit ----- test （测试有效）

依赖范围由强到弱的顺序是：compile>provided>runtime>test

### 测试

1、向web工程 添加jsp/servlet依赖

<!-- servlet jsp -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

</dependency>

package打war观察jsp-api和servlet-api是否在war中存在？

2、向dao工程 添加jdbc依赖

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

<scope>runtime</scope>

</dependency>

package打war观察mysql-connctor-java是否在war中存在？

## 完整的pom.xml

根据需求web工程要实现整合struts2，完整的pom.xml如下：

<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>cn.itcast.maven</groupId>

<artifactId>maven-web-0120</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<name>web工程，包括jsp、action等</name>

<description>web工程，包括jsp、action等</description>

<dependencies>

<!-- 添加junit4.9依赖 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.9</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.6</version>

<scope>runtime</scope>

</dependency>

<!-- servlet jsp -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

</dependency>

<!-- 依赖struts2 -->

<dependency>

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

<artifactId>struts2-core</artifactId>

<version>2.3.24</version>

</dependency>

</dependencies>

<build>

<plugins>

<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>

</project>

## action类

编写action，实现查询客户信息：

**public** **class** CustomerAction **extends** ActionSupport {

**private** Long custId;

/\*\*

\* **@return** the custId

\*/

**public** Long getCustId() {

**return** custId;

}

/\*\*

\* **@param** custId

\* the custId to set

\*/

**public** **void** setCustId(Long custId) {

**this**.custId = custId;

}

// 查询客户信息

**public** String querycustomer() {

System.***out***.println("客户请求客户Id："+custId);

**return** "success";

}

}

## struts.xml

在src/main/resources创建struts.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE struts PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 2.3//EN"

"http://struts.apache.org/dtds/struts-2.3.dtd">

<struts>

<!-- 配置常量 -->

<!-- 字符集 -->

<constant name=*"struts.i18n.encoding"* value=*"UTF-8"*></constant>

<!-- 开发模式 -->

<constant name=*"struts.devMode"* value=*"true"*></constant>

<!-- 主题 -->

<constant name=*"struts.ui.theme"* value=*"simple"*></constant>

<!-- 扩展名 -->

<constant name=*"struts.action.extension"* value=*"action"*></constant>

<!-- 通用package -->

<package name=*"customer"* namespace=*"/"* extends=*"struts-default"*>

<action name=*"querycustomer"* class=*"cn.itcast.crm.action.CustomerAction"*

method=*"querycustomer"*>

<result name=*"success"*>/jsp/querycustomer.jsp</result>

</action>

</package>

</struts>

## web.xml

在web.xml中配置struts2的前端控制器

<?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/web-app\_2\_5.xsd"*

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"*>

<filter>

<filter-name>struts2</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

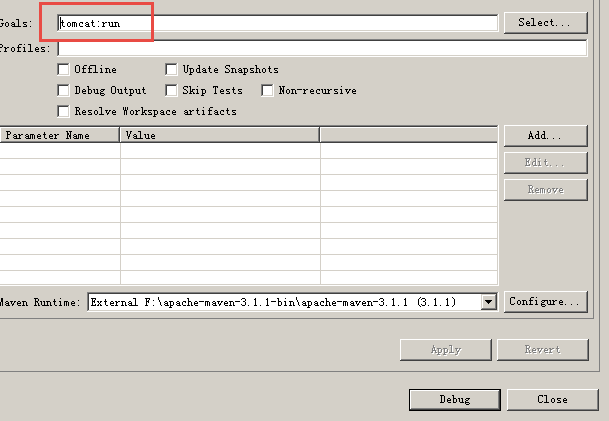
</welcome-file-list>

</web-app>

# maven工程运行调试

## tomcat插件

maven内置tomcat的插件（org.codehaus.mojo. tomcat-maven-plugin），执行tomcat:run命令即可启动tomcat



访问：http://localhost:8080/工程名/...

可以通过配置plugin修改tomcat的访问路径及端口：

<build>

<plugins>

<!-- maven内置 的tomcat6插件 -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>tomcat-maven-plugin</artifactId>

<version>1.1</version>

<configuration>

<!-- 可以灵活配置工程路径 -->

<path>/ssh</path>

<!-- 可以灵活配置端口号 -->

<port>8080</port>

</configuration>

</plugin>

</plugins>

</build>

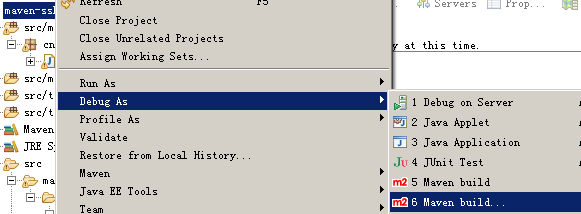
## 端口占用处理

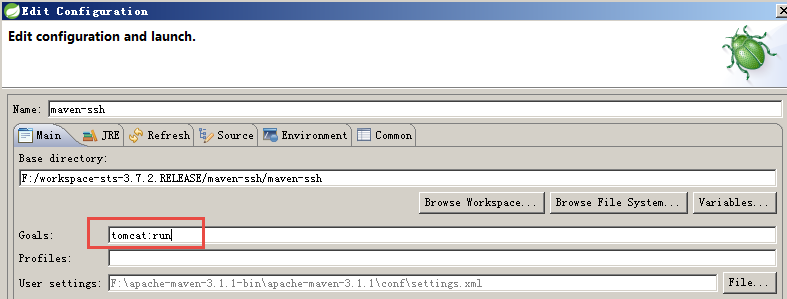
重新执行tomcat:run命令重启工程，重启之前需手动停止 tomcat，否则报下边的错误：

Caused by: java.net.BindException: Address already in use: JVM\_Bind

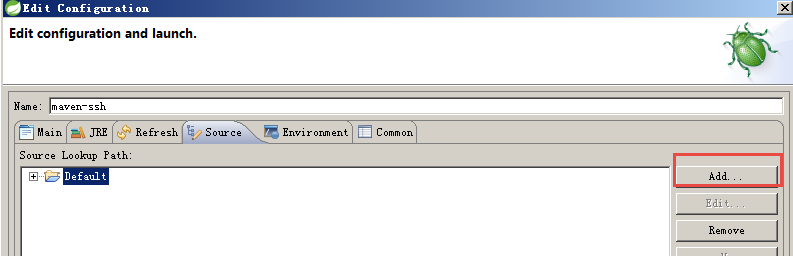
## 断点调试

maven工程断点调试必须采用“Debug As”方式启动，并且需要引入源码才可源码跟踪：

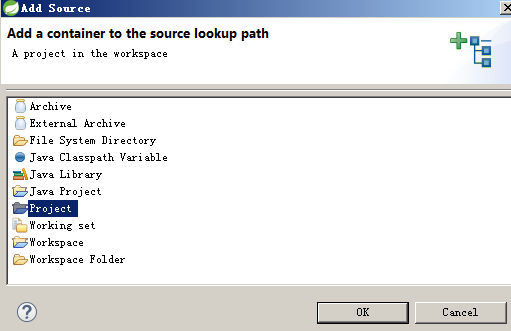


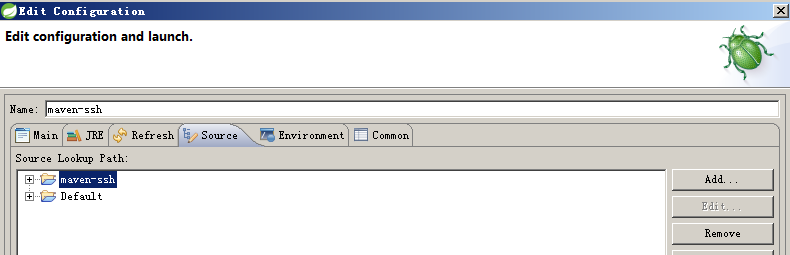


引入源码：

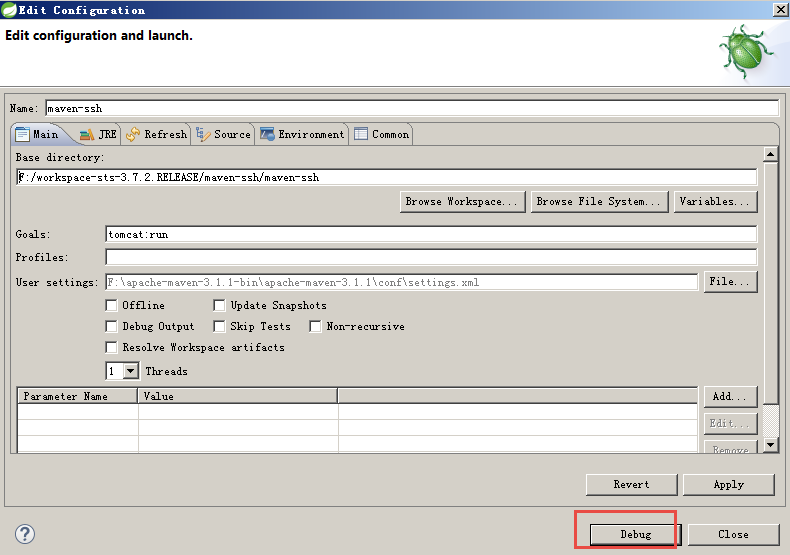


添加，选择本工程：





以debug方式运行：



# 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。