# maven安装

## 下载安装

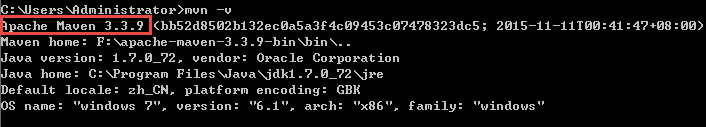
## 环境变量配置

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 ）

将;D:\maven\apache-maven-3.3.9\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。



### 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。



：

## 项目构建

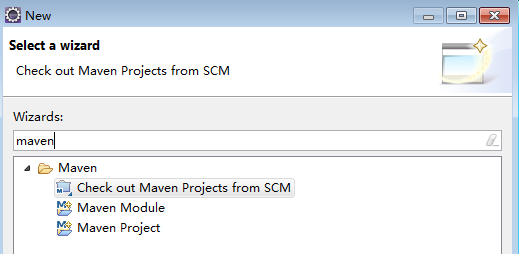
### eclipse与maven

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

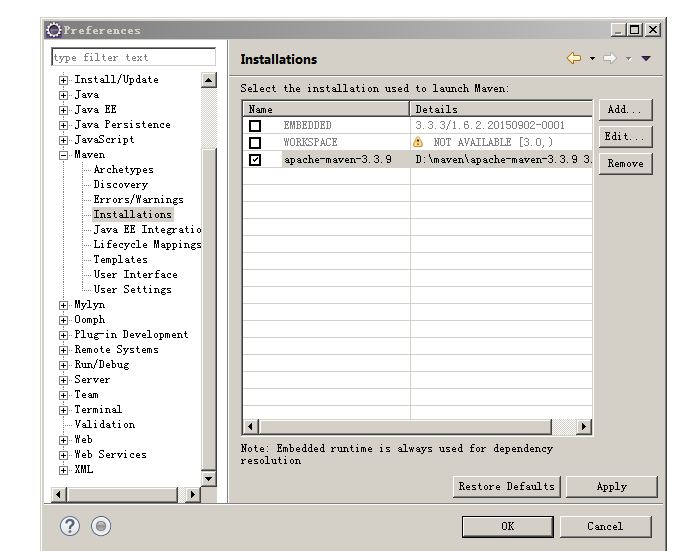
使用eclipse mars 2版本，此版本自带maven插件不用单独安装。

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：



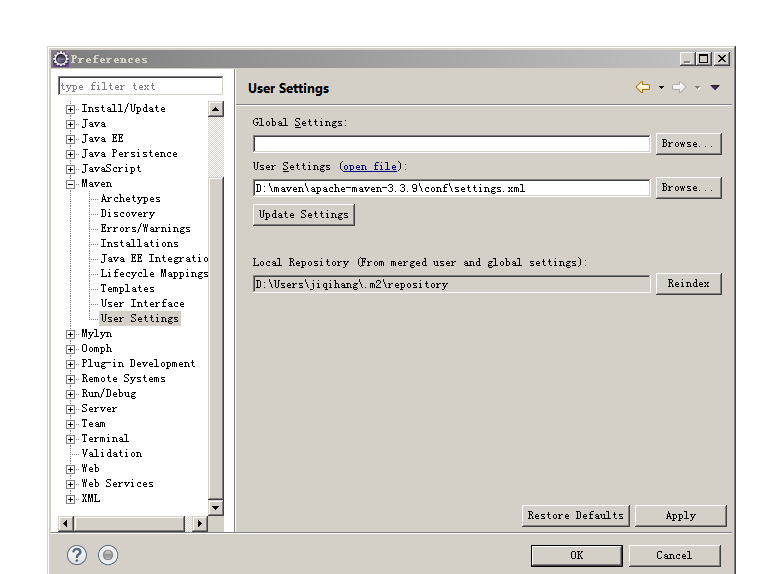
### 指定maven安装目录

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。

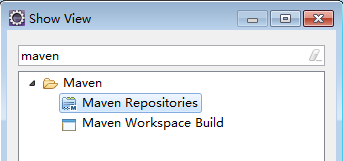


注意：如果修改了 setting.xml文件需要点击上图中的“update settings”按钮对本地仓库重建索引，点击“Reindex”。

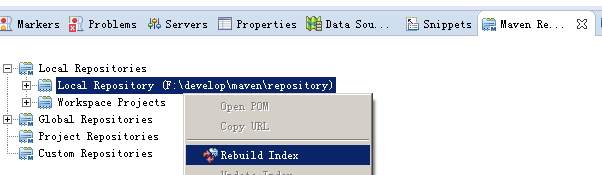
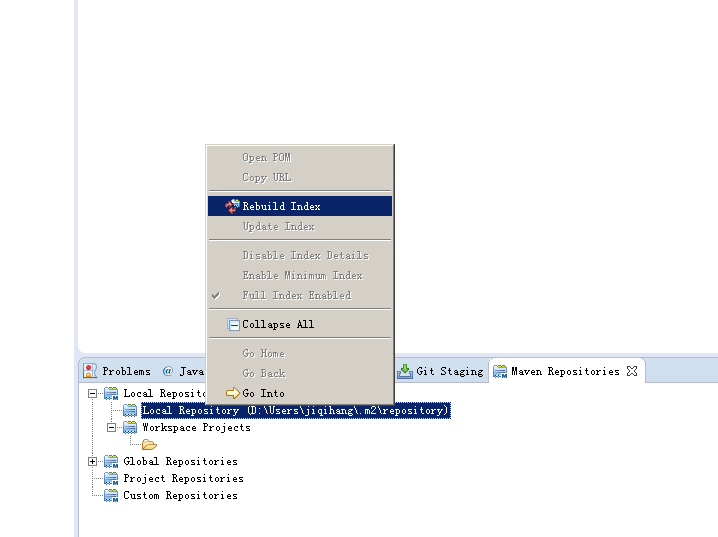
### eclipse浏览仓库

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。

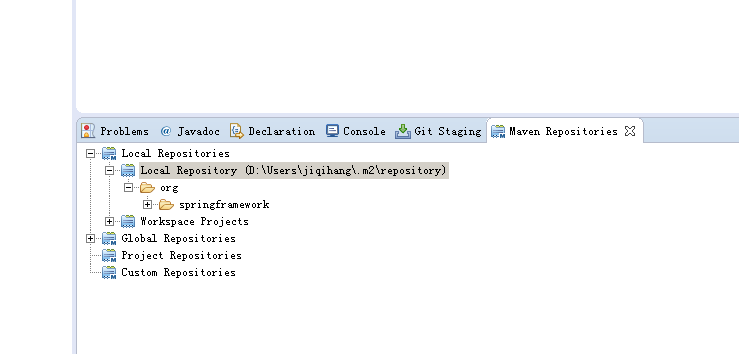
打开eclipse仓库视图，对插件和jar包建立索引



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

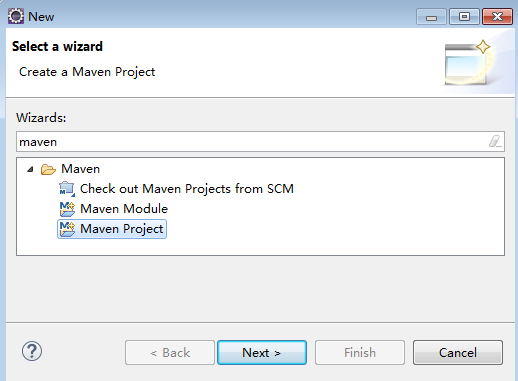
war：执行package会打成war包

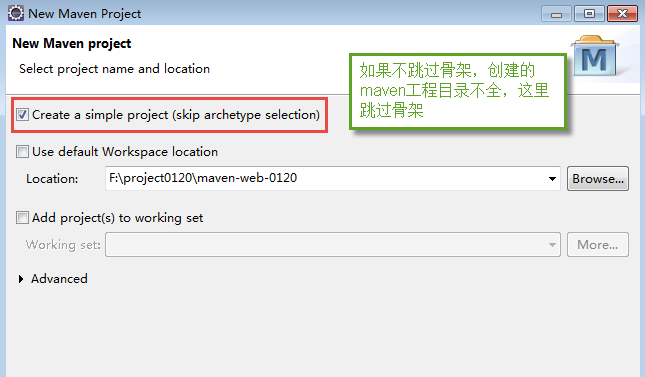
pom ：用于maven工程的继承，通常父工程设置为pom

## 构建web工程

### 第一步创建maven工程

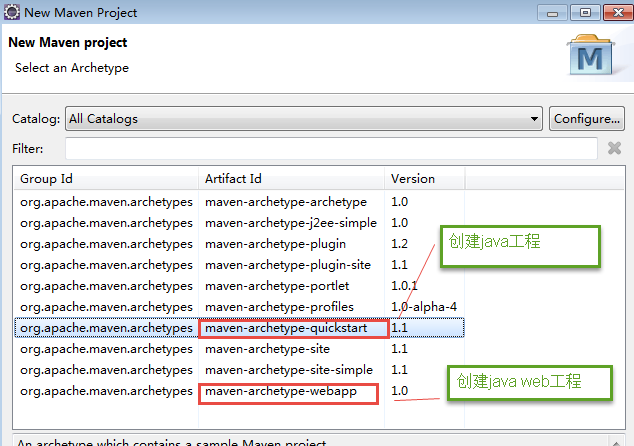
选择 Maven Project





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：

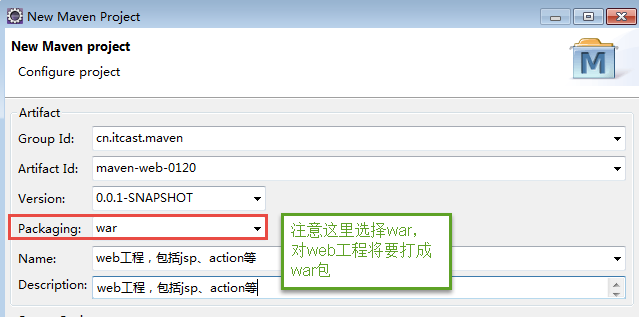


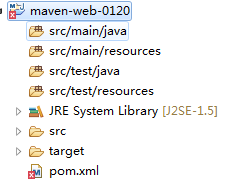
选择webapp会创建一个maven工程（java web工程）

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>

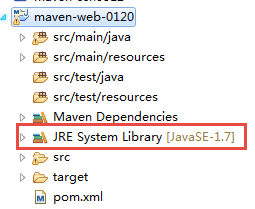
</configuration>

</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

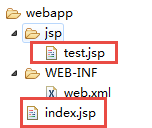
**protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {

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

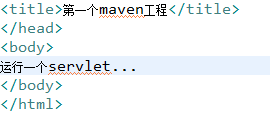
}

}

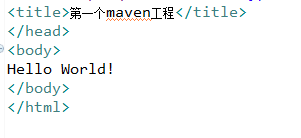
### 第六步编写jsp



test.jsp的内容如下：

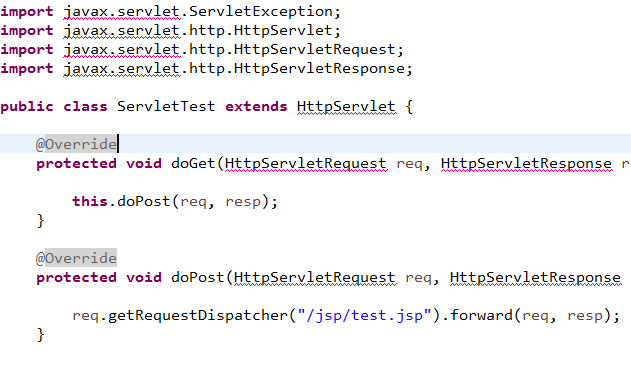


index.jsp的内容如下：



### 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>

# 依赖管理-添加依赖

## 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>

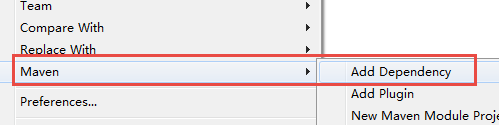
</dependency>

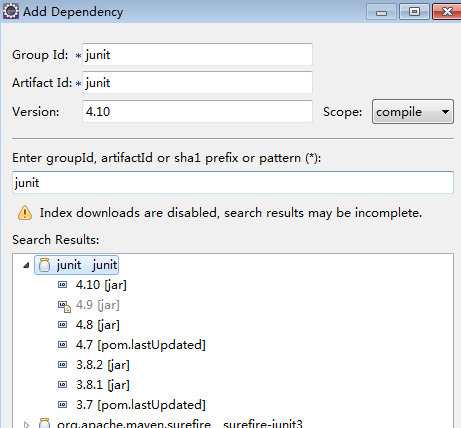
</dependencies>

### 查找坐标

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>

## 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>