**项目管理工具**

**maven**

# 学习目标

掌握Maven是什么

掌握Maven的安装和配置

使用Maven构建web工程

掌握Maven的常用命令

对Maven工程进行调试

# maven介绍

## 1.1maven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。

## 1.2什么是项目构建

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

### 1.2.1传统项目构建过程

传统的使用eclipse构建项目的过程如下：

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

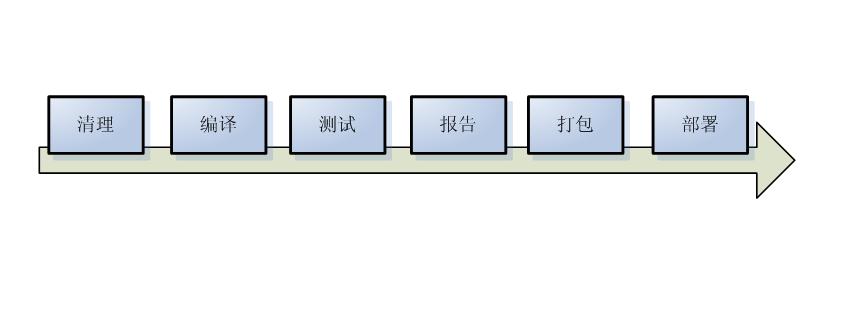
3）对源代码进行编译，java文件编译成class文件

4）执行Junit单元测试

1. 将工程打成war包部署至tomcat运行

### 1.2.2maven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。

**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。

## 1.3什么是依赖管理

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。

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。

### 1.3.1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。

**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。

## 1.4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

## 2.1下载安装

Maven是Apache下的一个顶级项目，可以<http://maven.apache.org/download.cgi> 最新版为maven3.5.2 版本。

将下载好的maven解压到一个不含有中文和空格的目录中，解压后的maven包含以下的文件目录。

|  |
| --- |
| 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 |

## 2.2环境变量配置

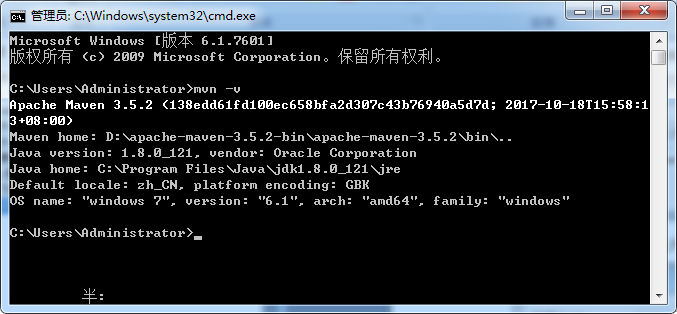
电脑上需安装java环境，安装JDK1.7 + 版本 （将JAVA\_HOME/bin 配置环境变量path ）首先需要配置 MAVEN\_HOME。



接下来将 %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的版本即为安装成功。

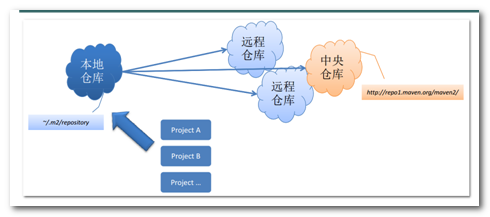


## 2.3 maven仓库

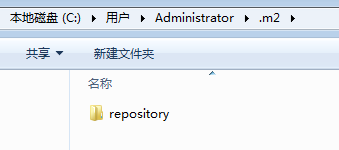
### 2.3.1 maven仓库的作用

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。

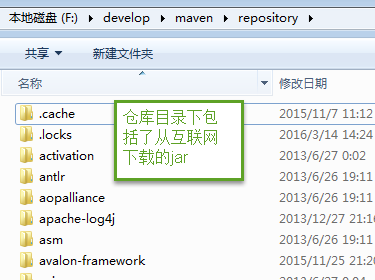


远程仓库：如果本地需要插件或者jar包，本地仓库没有，默认去远程仓库下载。远程仓库可以在互联网内也可以在局域网内。

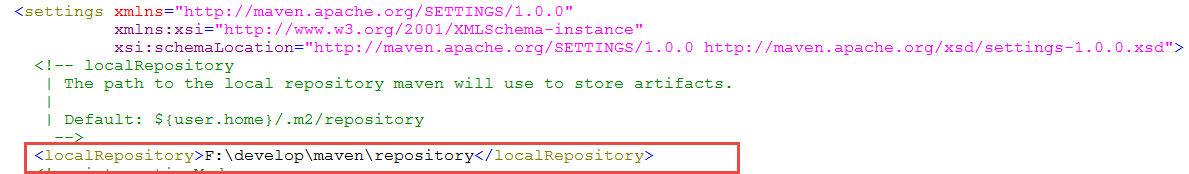
中央仓库：在maven软件中内置一个默认使用的远程仓库，仓库地址的为http://repo1.maven.org/maven2，它是中央仓库，服务于整个互联网，它是由Maven团队自己维护，里面存储了非常全的jar包，它包含了世界上大部分流行的开源项目构件。

### 2.3.2 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：



### 2.3.3 全局setting与用户setting

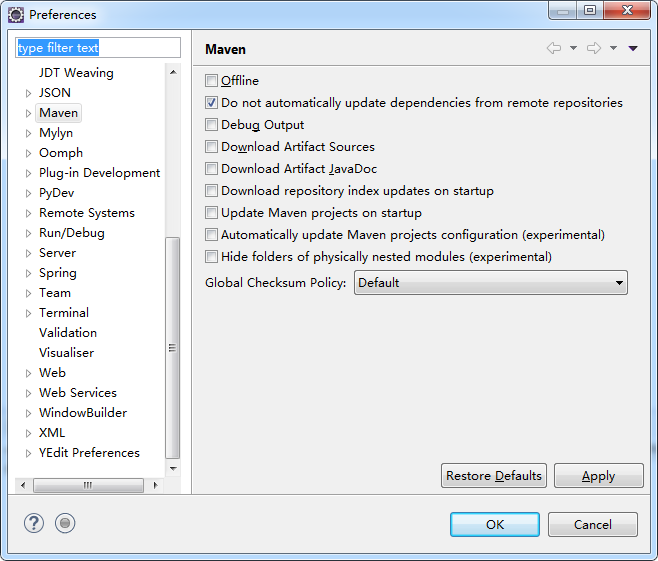
maven仓库地址、私服等配置信息需要在setting.xml文件中配置，分为全局配置和用户配置。在maven安装目录下的有 conf/setting.xml文件，此setting.xml文件用于maven的所有project项目，它作为maven的全局配置。

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。

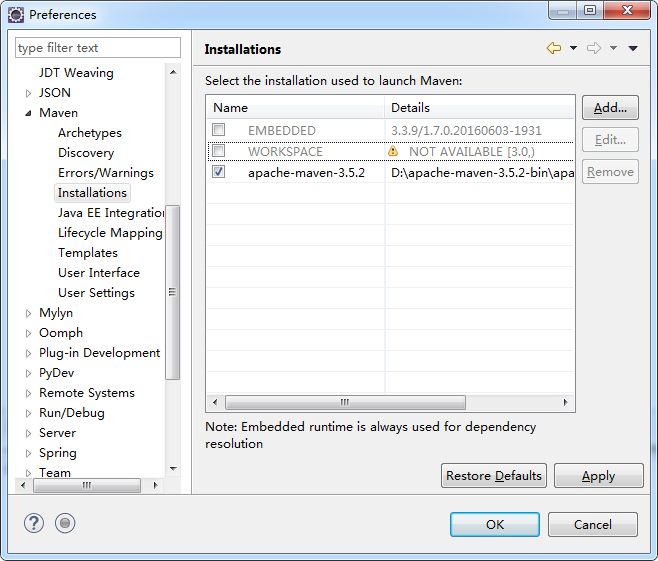
# 项目构建

## 指定maven安装目录

我们使用的Eclipse已经集成了Maven的插件，因此只要在Eclipse中指定本地maven的安装目录即可。

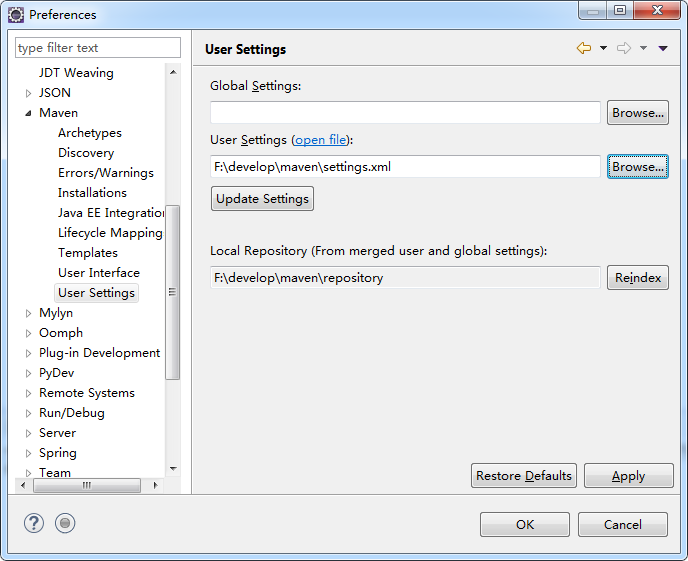


找到maven的配置，添加本地的maven。



## 3.2指定用户setting配置文件

在eclipse中配置使用的maven的setting.xml文件，使用用户自己的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”。

## 3.3 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。

|  |
| --- |
| groupId：项目名称，定义为组织名+项目名，类似包名  artifactId：模块名称  version：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  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 |

## 3.4构建web工程

### 3.4.1需求

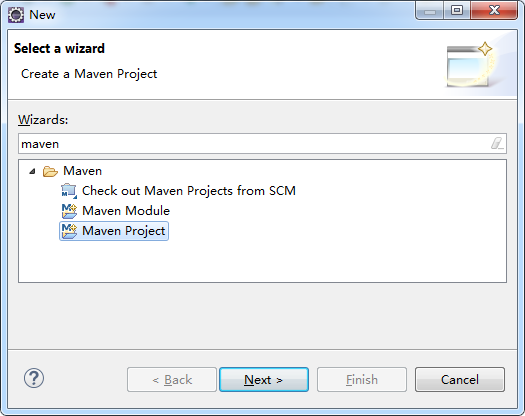
创建一个web工程，实现入门程序的功能。

1）添加index.jsp，输出hello world

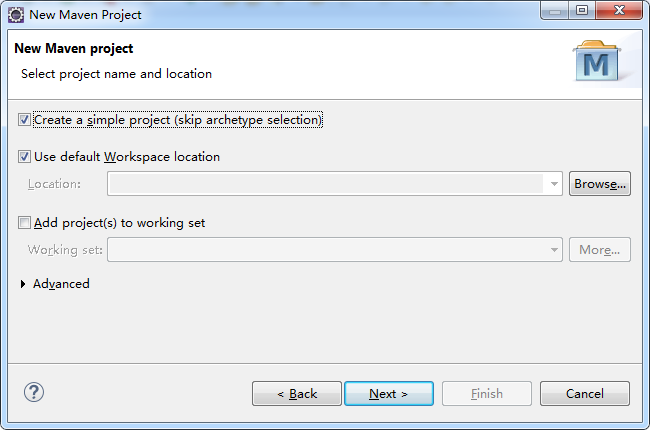
2）添加一个servlet转发到jsp页面。

### 3.4.2第一步创建maven工程

选择创建Maven Project。

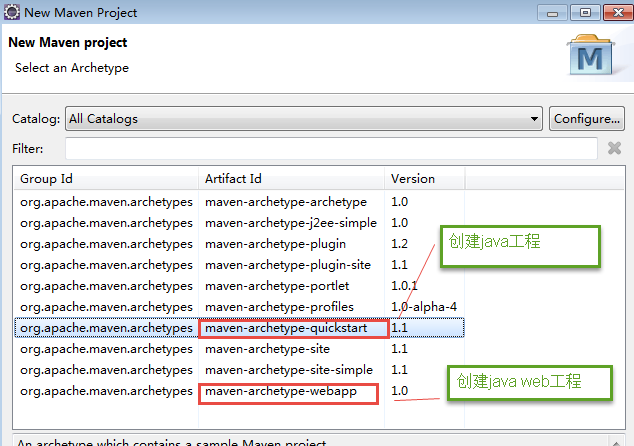


接下来是否使用骨架创建maven工程。这里我们不适用maven的骨架，自己手动创建工程，创建简单工程可以跳过骨架的选择。

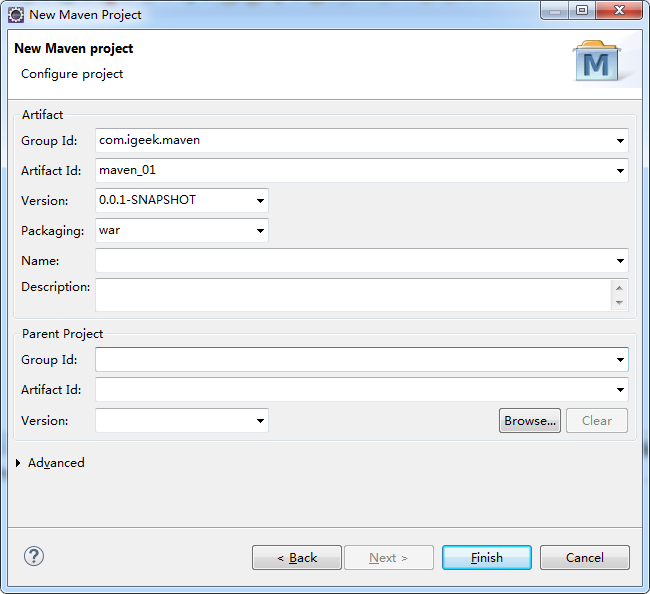


当然我们也可以了解一下maven的骨架:

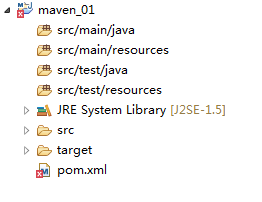
当我们不跳过骨架点击“next”会进入骨架选择页面，如果eclipse中配置本地仓库正确则显示出骨架：



### 3.4.2第二步定义坐标



创建的工程结构如下：

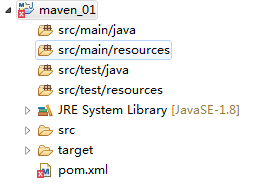


### 3.4.3第三步设置编译版本

查看上边工程的编译版本为1.5，本教程使用jdk1.8，需要设置编译版本为1.8，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>1.8</source>  <target>1.8</target>  <encoding>UTF-8</encoding>  </configuration>  </plugin>  </plugins>  </build> |

执行update project，查看编译版本为1.8：



### 3.4.4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>

### 3.4.5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 {

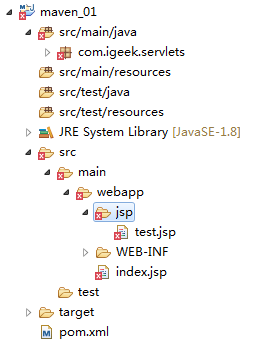
req.getRequestDispatcher("/jsp/test.jsp").forward(req, resp);

}

}

### 3.4.6第六步编写JSP

在webapp下编写index.jsp，同时创建一个jsp目录，并在其中创建test.jsp页面。



test.jsp的内容如下：

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=utf-8"*  pageEncoding=*"utf-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>第一个Maven工程</title>  </head>  <body>  这是一个测试Servlet  </body>  </html> |

index.jsp的内容如下：

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=utf-8"*  pageEncoding=*"utf-8"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=utf-8"*>  <title>第一个Maven工程</title>  </head>  <body>  Hello world!  </body>  </html> |

### 3.4.7第七步添加servlet/jsp的jar包

此时的工程中servlet和jsp报错，是因为缺少servlet和jsp依赖的jar包，我们需要添加这样的依赖。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>

### 3.4.8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>com.igeek.servlets.ServletTest</servlet-class>

</servlet>

<servlet-mapping>

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

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

</servlet-mapping>

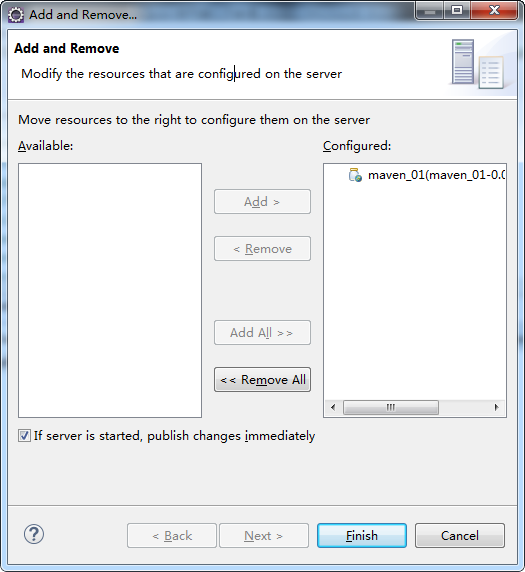
### 3.4.9运行

#### 3.4.9.1使用tomcat服务器来发布

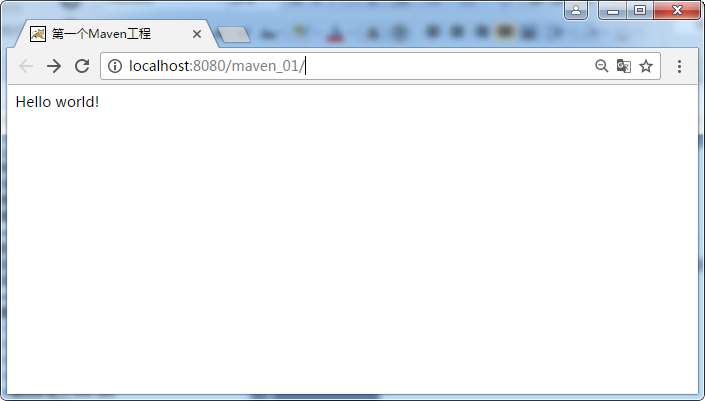
Maven的web工程可以和之前的动态web工程一样，使用Tomcat服务器来发布和运行，这样我们需要首先添加一个Tomcat服务器。



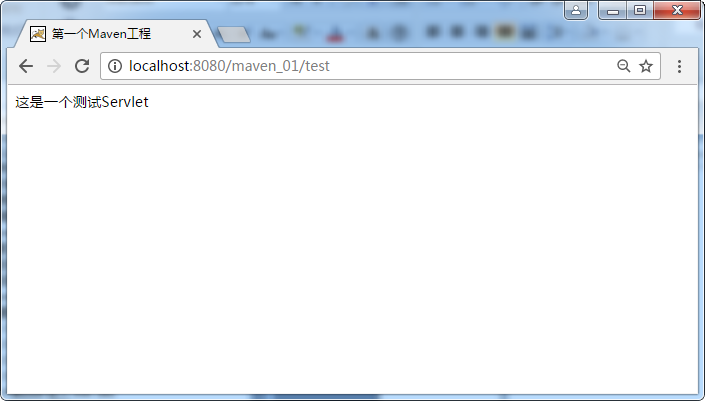
然后将我们的工程添加到Tomcat服务器中。



启动Tomcat服务器之后就可以访问我们的页面了。



访问test页面。



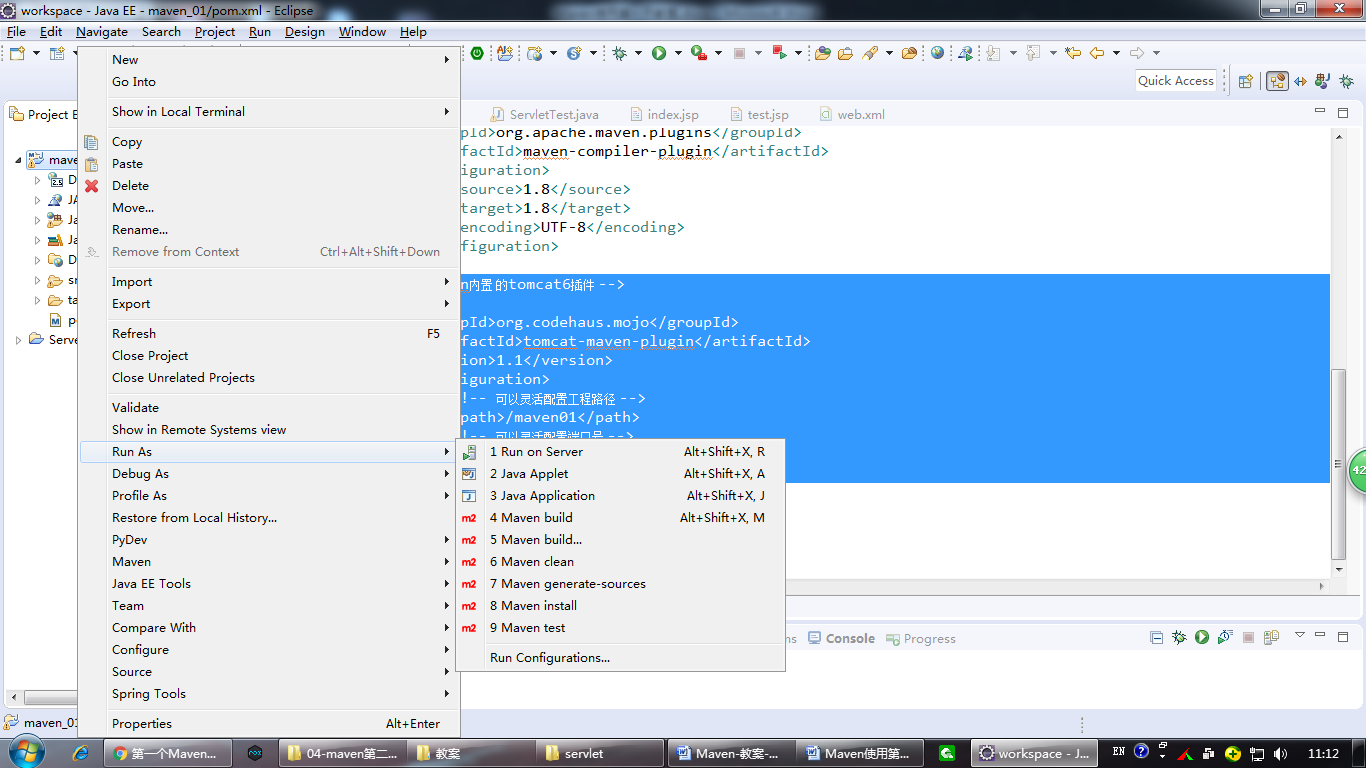
#### 3.4.9.1使用Maven的服务器插件来发布

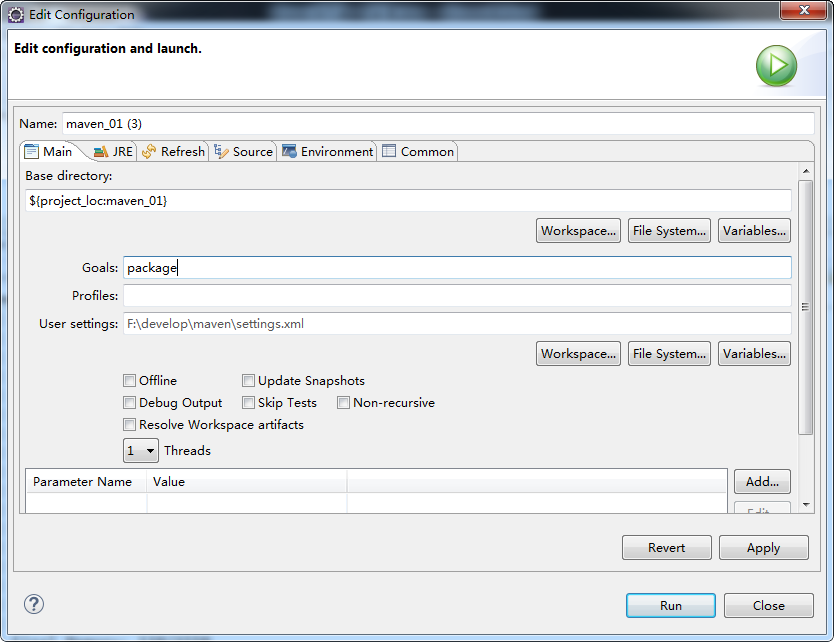
我们也可以使用maven的Tomcat插件来发布工程，我们需要首先添加该插件。可以通过配置plugin修改tomcat的访问路径及端口。

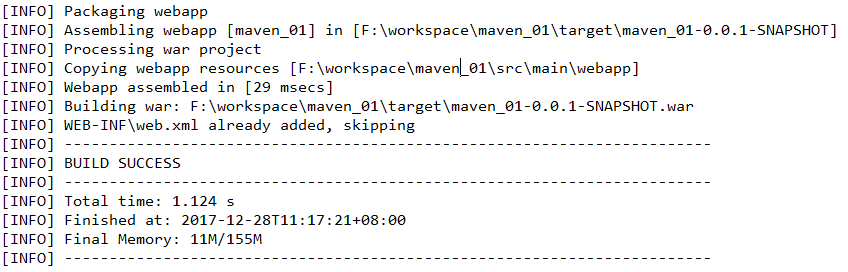
在pom.xml中添加如下配置：

|  |
| --- |
| <!-- maven内置 的tomcat6插件 -->  <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.2</version>  <configuration>  <!-- 可以灵活配置工程路径 -->  <path>/maven01</path>  <!-- 可以灵活配置端口号 -->  <port>8080</port>  </configuration>  </plugin> |

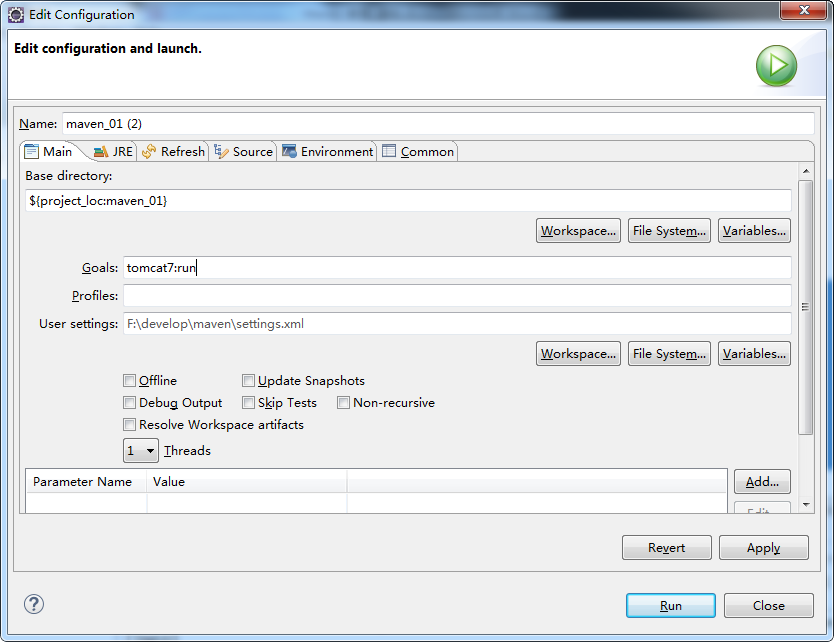
在启动tomcat插件发布工程前，需要首先对工程进行编译和打包，我们使用命令package。

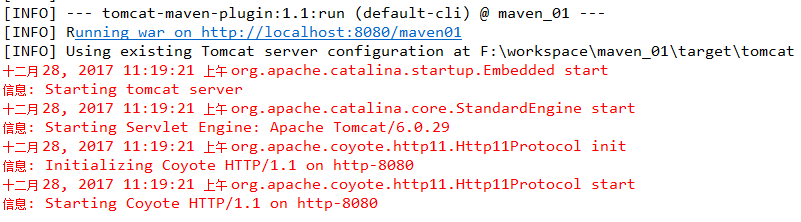




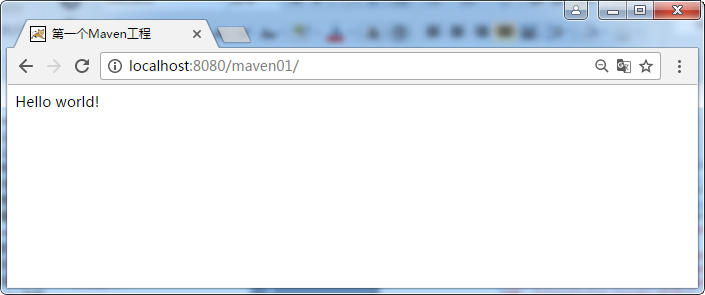


接下来就可以启动服务了，我们使用tomcat:run命令。

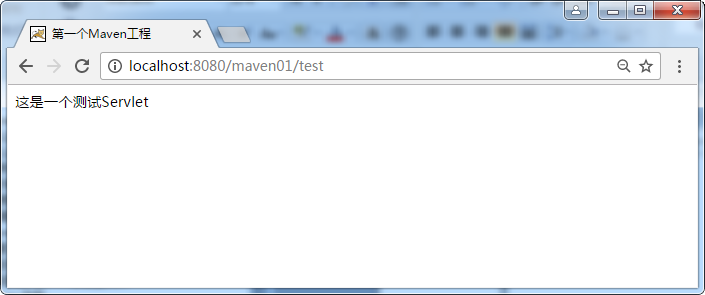




服务器启动完成，接下来就可以访问页面了。



访问test.jsp页面。



# Maven的命令和周期

## 4.1Maven的常用命令

Maven的常用命令包含以下几个，我们需要熟练使用。

### 4.1.1 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。

### 4.1.2 test

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

### 4.1.3 clean

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

### 4.1.4 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。

### 4.1.5 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。

## 4.2Maven的生命周期

### 4.2.1三套生命周期

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：

1. Clean Lifecycle 在进行真正的构建之前进行一些清理工作。
2. Default Lifecycle 构建的核心部分，编译，测试，打包，部署等等。
3. Site Lifecycle 生成项目报告，站点，发布站点。

### 4.2.2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。

**4.2.2.1如下是clean生命周期的阶段**

|  |
| --- |
| 1. pre-clean 执行一些需要在clean之前完成的工作 2. clean 移除所有上一次构建生成的文件 3. post-clean 执行一些需要在clean之后立刻完成的工作 |

**4.2.2.2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。 |

**4.2.2.3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 |

### 4.2.3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。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。

# 依赖管理

## 5.1添加依赖

### dependency

在pom.xml中添加dependency标签，如下：

|  |
| --- |
| <dependency>  <groupId><groupId>  <artifactId></artifactId>  <version></version>  </dependency> |

如果要添加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>

### 5.1.2查找坐标

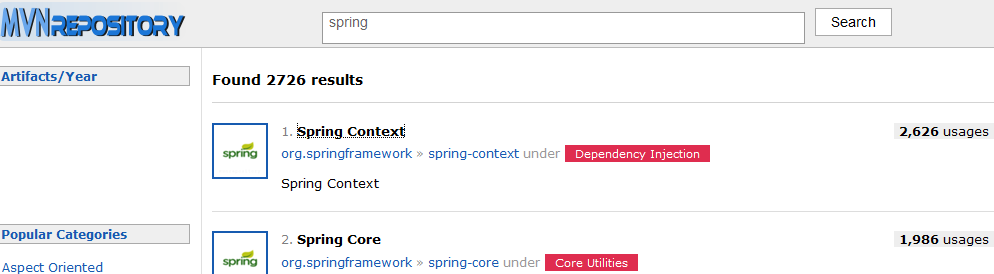
添加依赖需要指定依赖jar包的坐标，但是很多情况我们是不知道jar包的的坐标，可以通过如下方式查询：

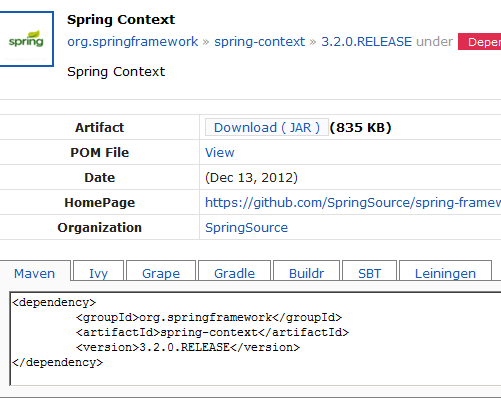
* 方法一：从互联网搜索

http://search.maven.org/

http://mvnrepository.com/

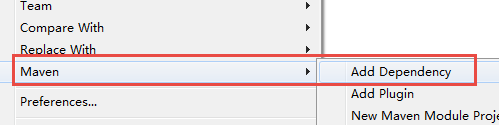
网站搜索示例：

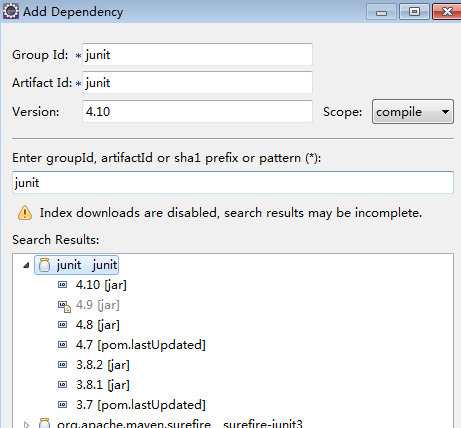




* 方法二：使用maven插件的索引功能

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

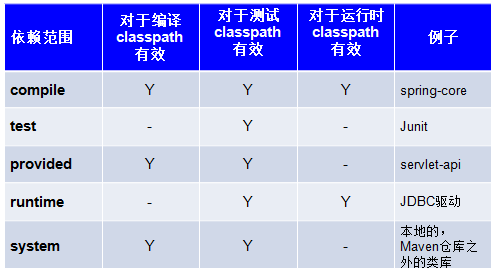




## 5.2依赖范围

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。



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

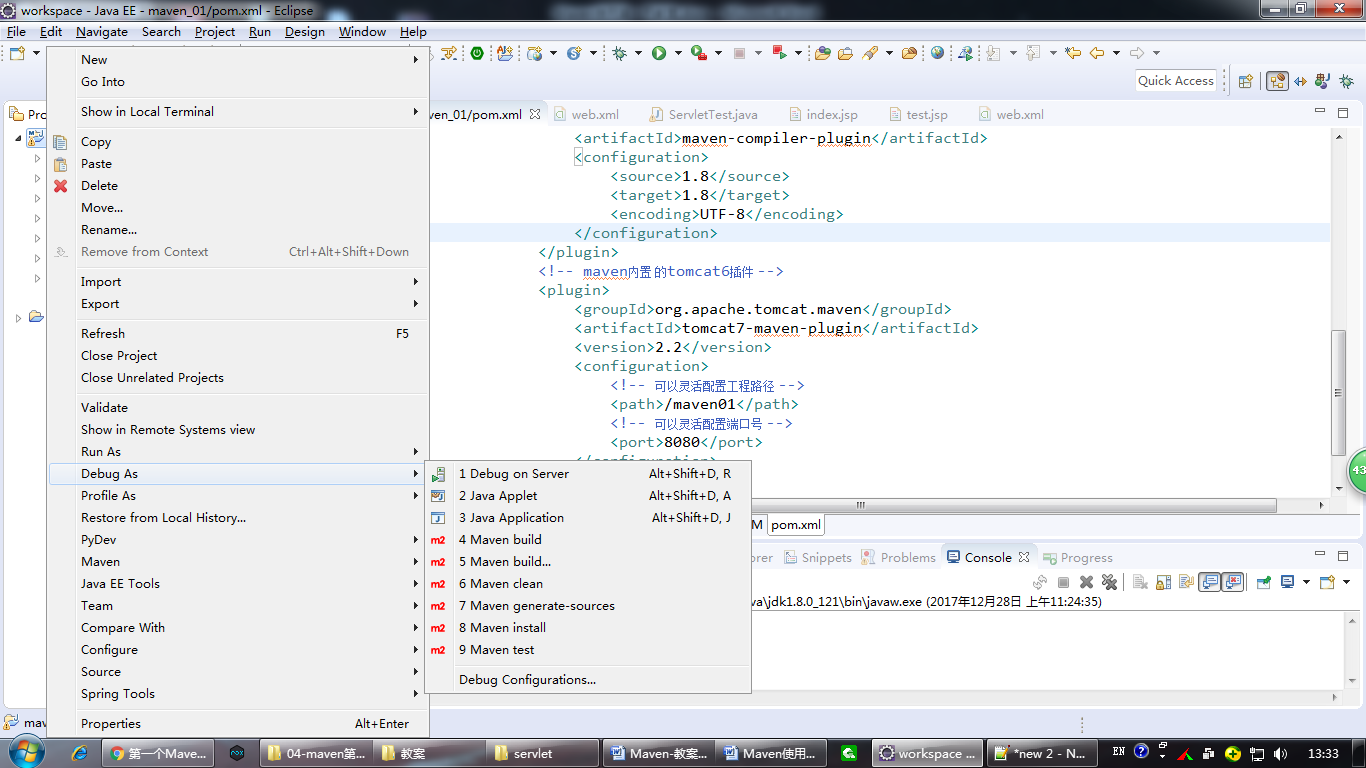
# maven工程运行调试

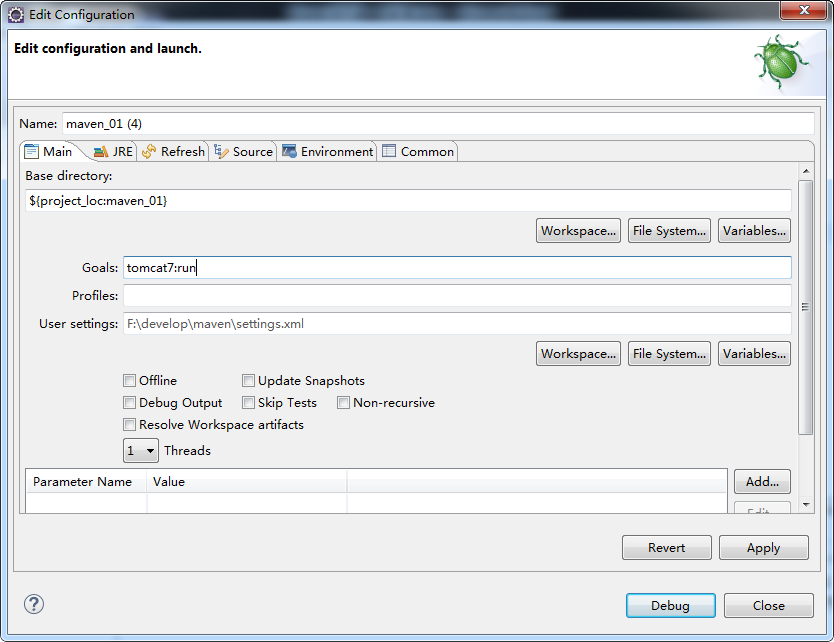
## 6.1端口占用处理

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

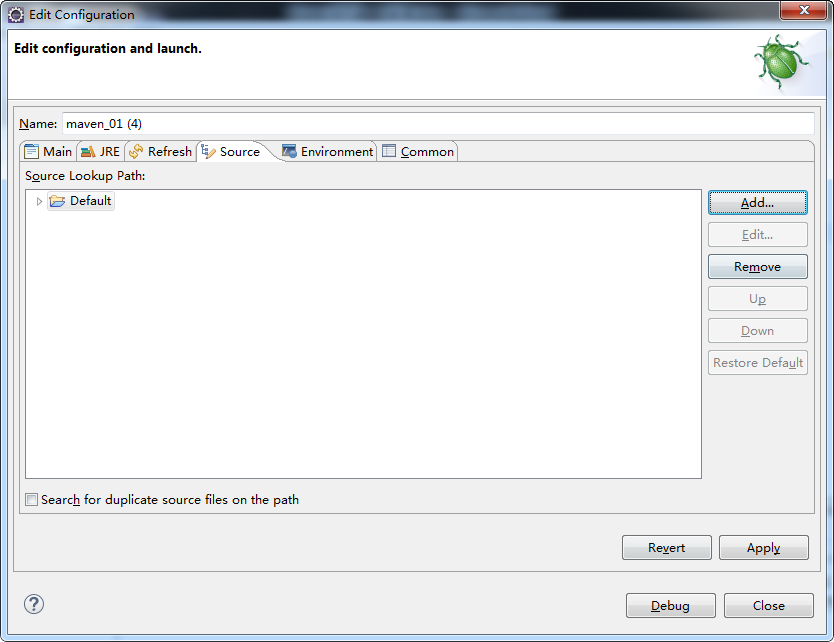
## 6.2断点调试

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：

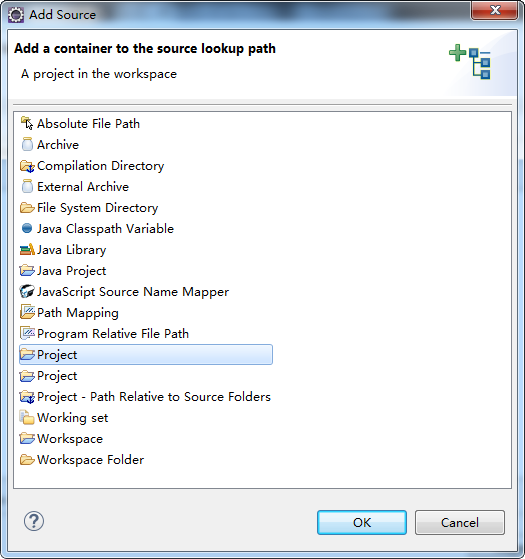


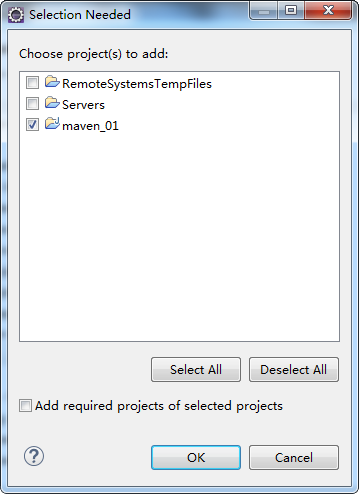


引入源码：



添加，选择本工程：





以debug方式运行：

