

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

1 maven 介绍

1.1 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。

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

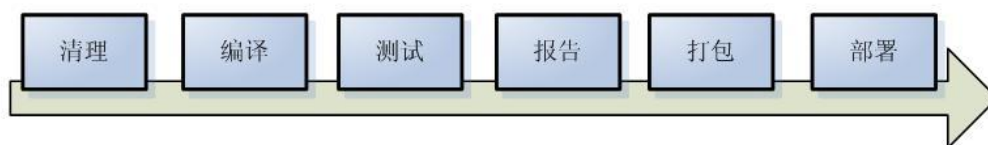
1.2.1 传统项目构建过程

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
- 5) 将工程打成 war 包部署至 tomcat 运行

1.2.2 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。

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。

1.3.2 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 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。

2 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。

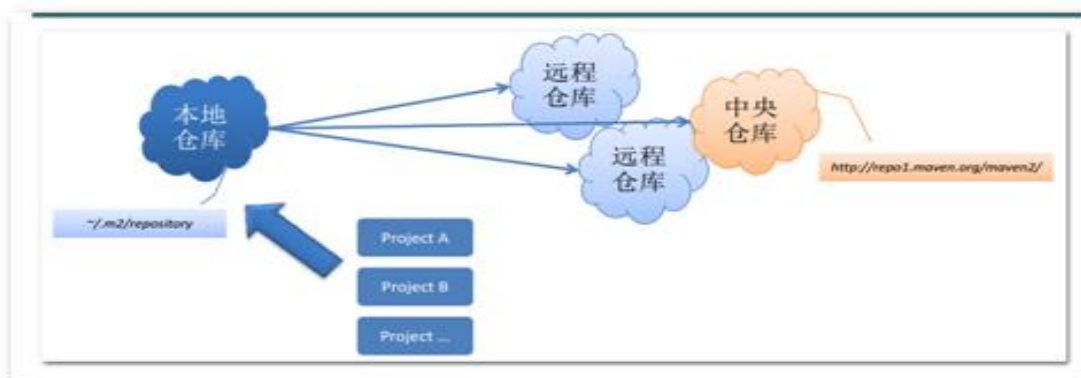


2.3 maven 仓库

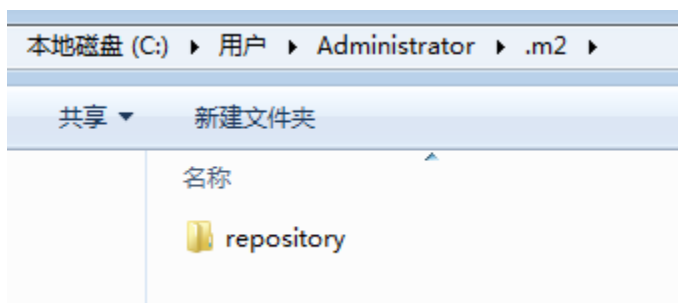
2.3.1 maven 仓库的作用

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。

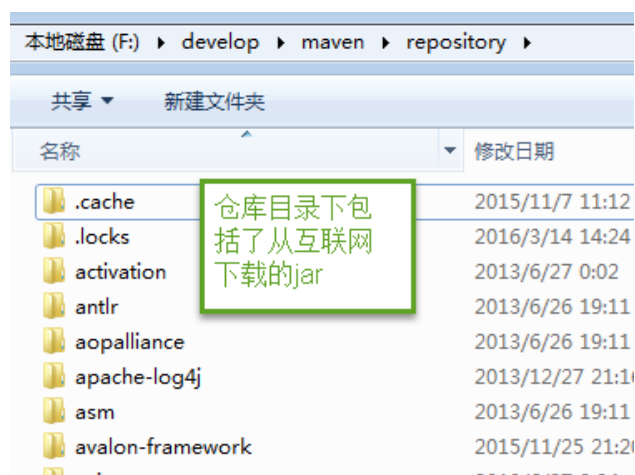


远程仓库：如果本地需要插件或者 jar 包，本地仓库没有，默认去远程仓库下载。远程仓库可以在互联网内也可以在局域网内。

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：

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <!-- localRepository
    | The path to the local repository maven will use to store artifacts.
    | Default: ${user.home}/.m2/repository
    -->
  <localRepository>F:\develop\maven\repository</localRepository>
```

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。

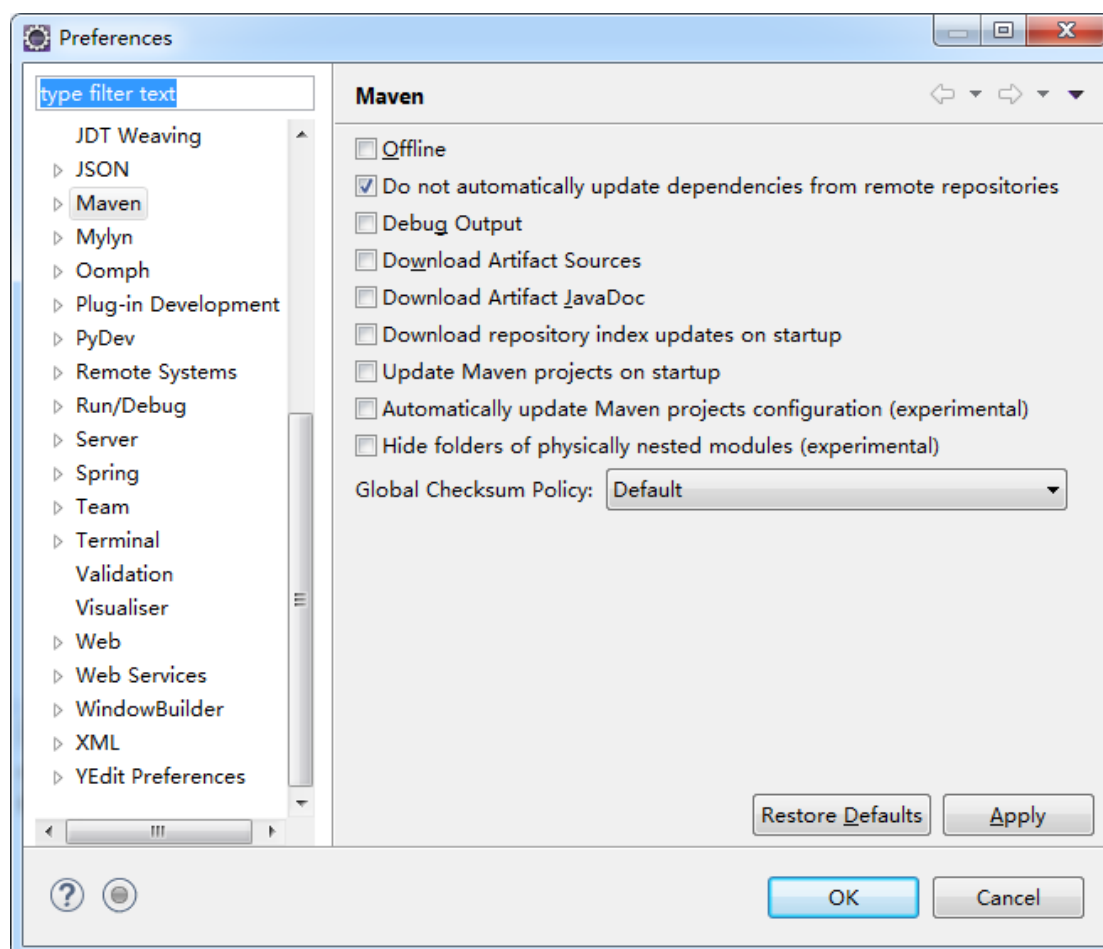
如需要个性配置则需要在用户配置中设置，用户配置的 setting.xml 文件默认的位置在：

`${user.dir} /.m2/settings.xml` 目录中,`${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。

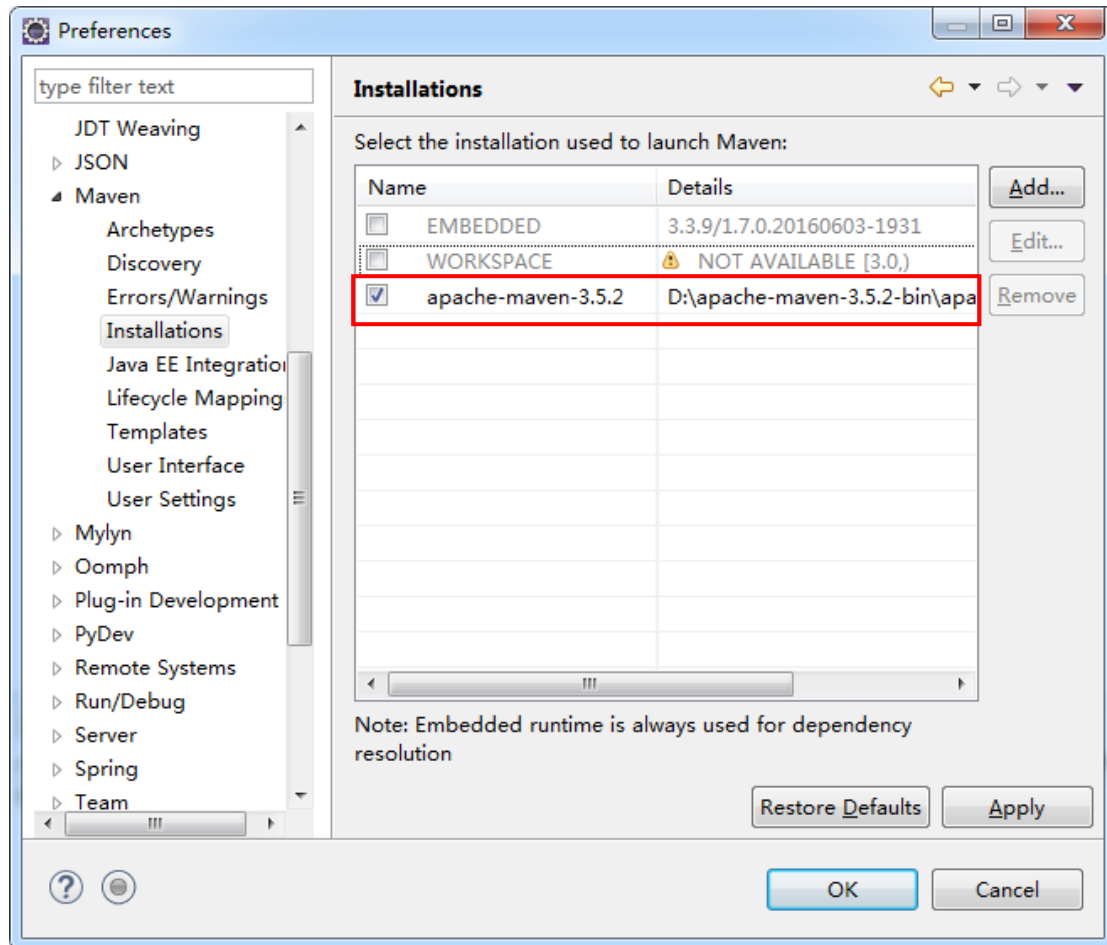
3 项目构建

3.1 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。

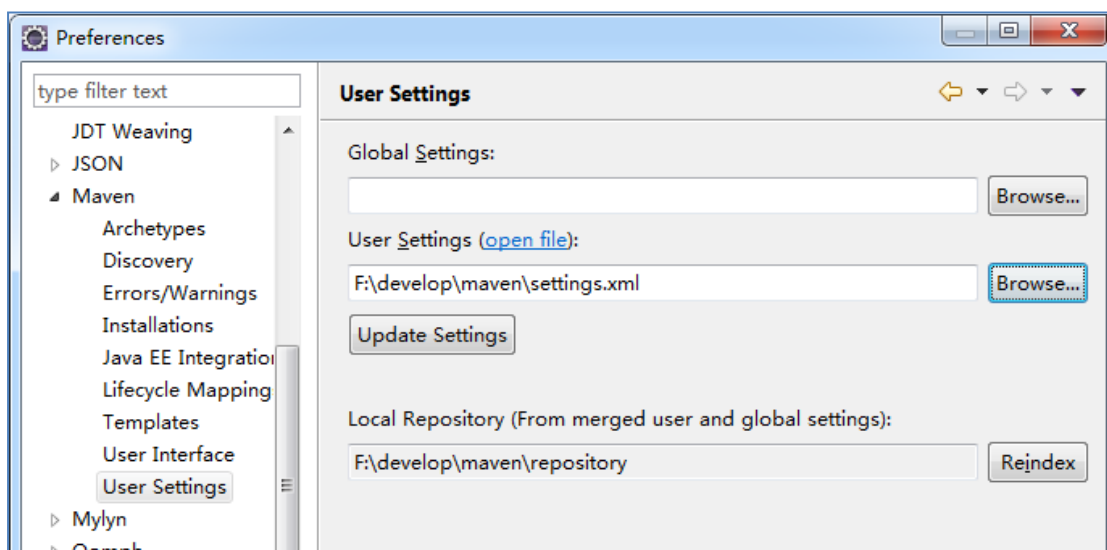


找到 maven 的配置, 添加本地的 maven。



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

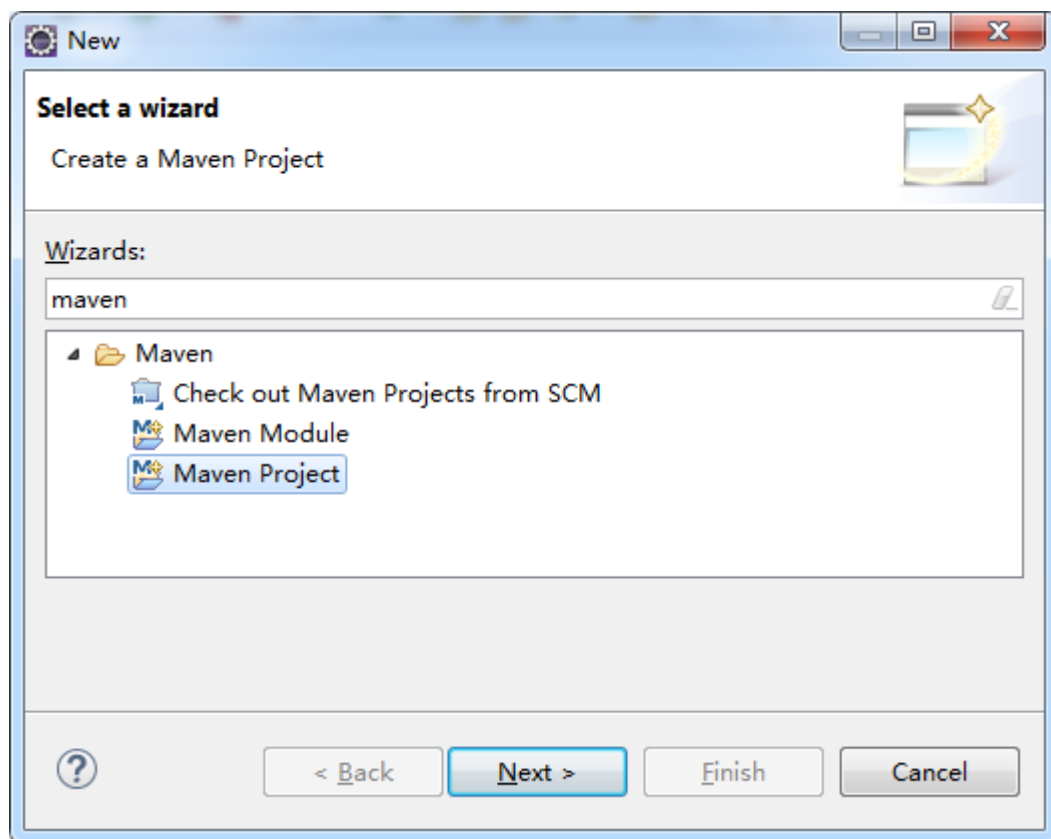
3.4.1 需求

创建一个 web 工程，实现入门程序的功能。

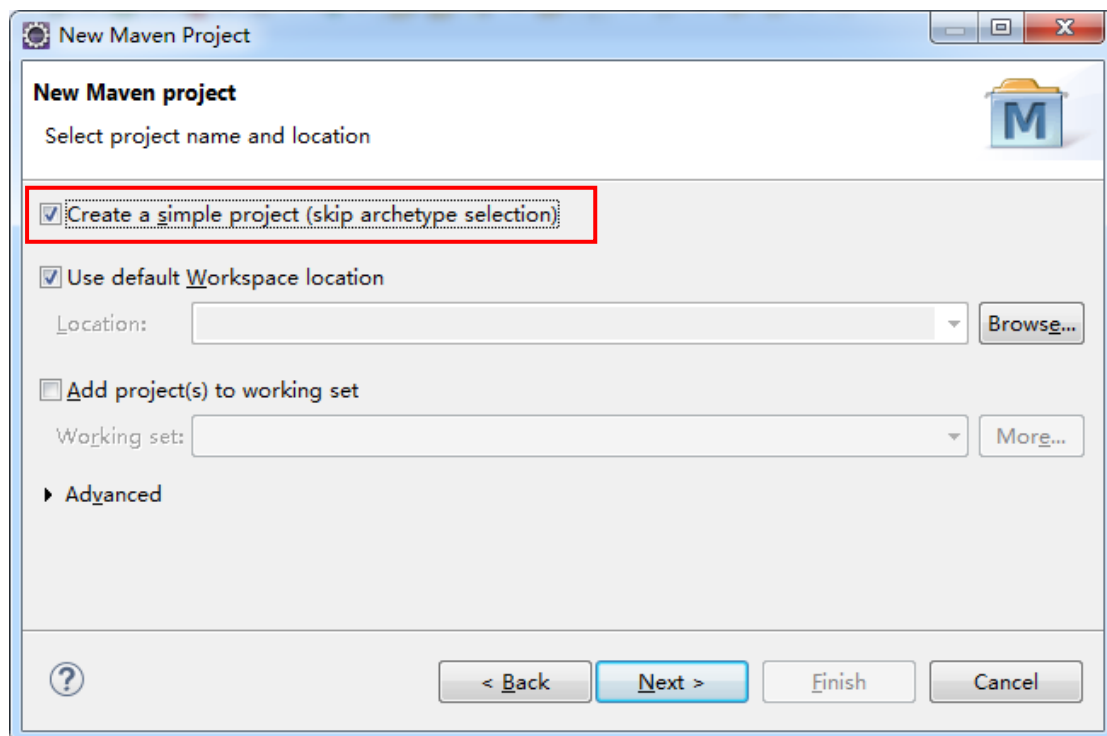
- 1) 添加 index.jsp，输出 hello world
- 2) 添加一个 servlet 转发到 jsp 页面。

3.4.2 第一步创建 maven 工程

选择创建 Maven Project。



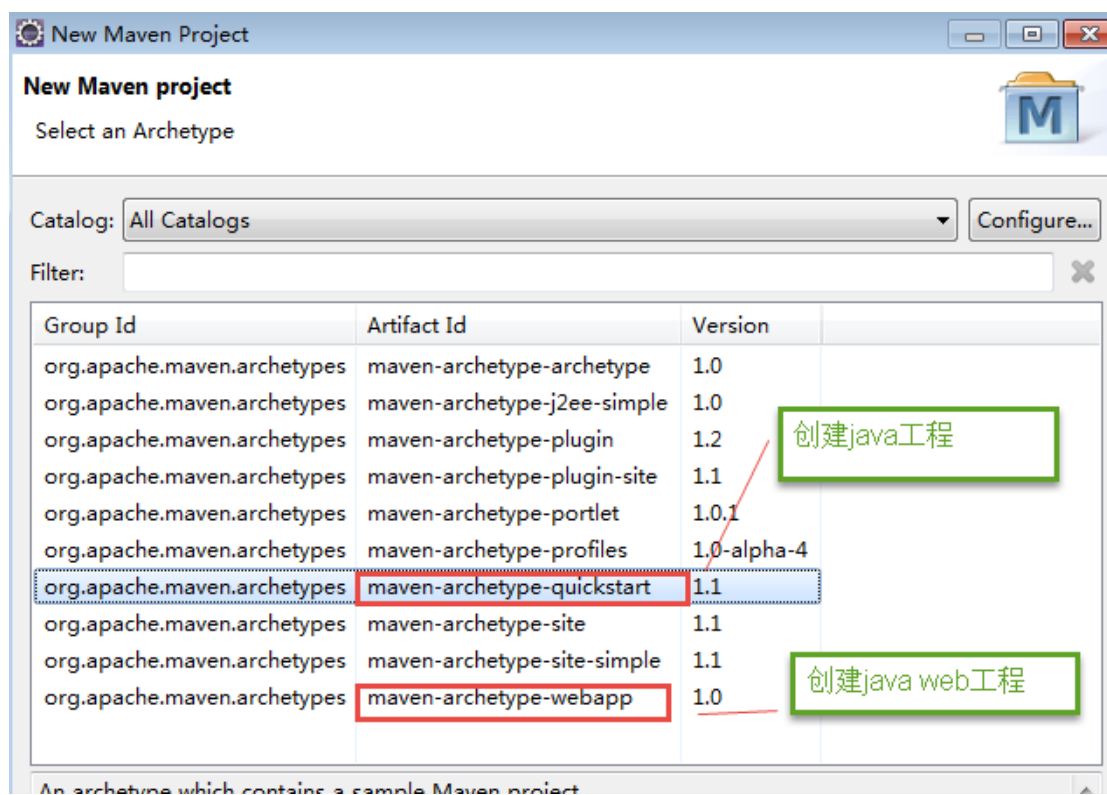
接下来是否使用骨架创建 maven 工程。这里我们不适用 maven 的骨架，自己手动创建工程，创建简单工程可以跳过骨架的选择。



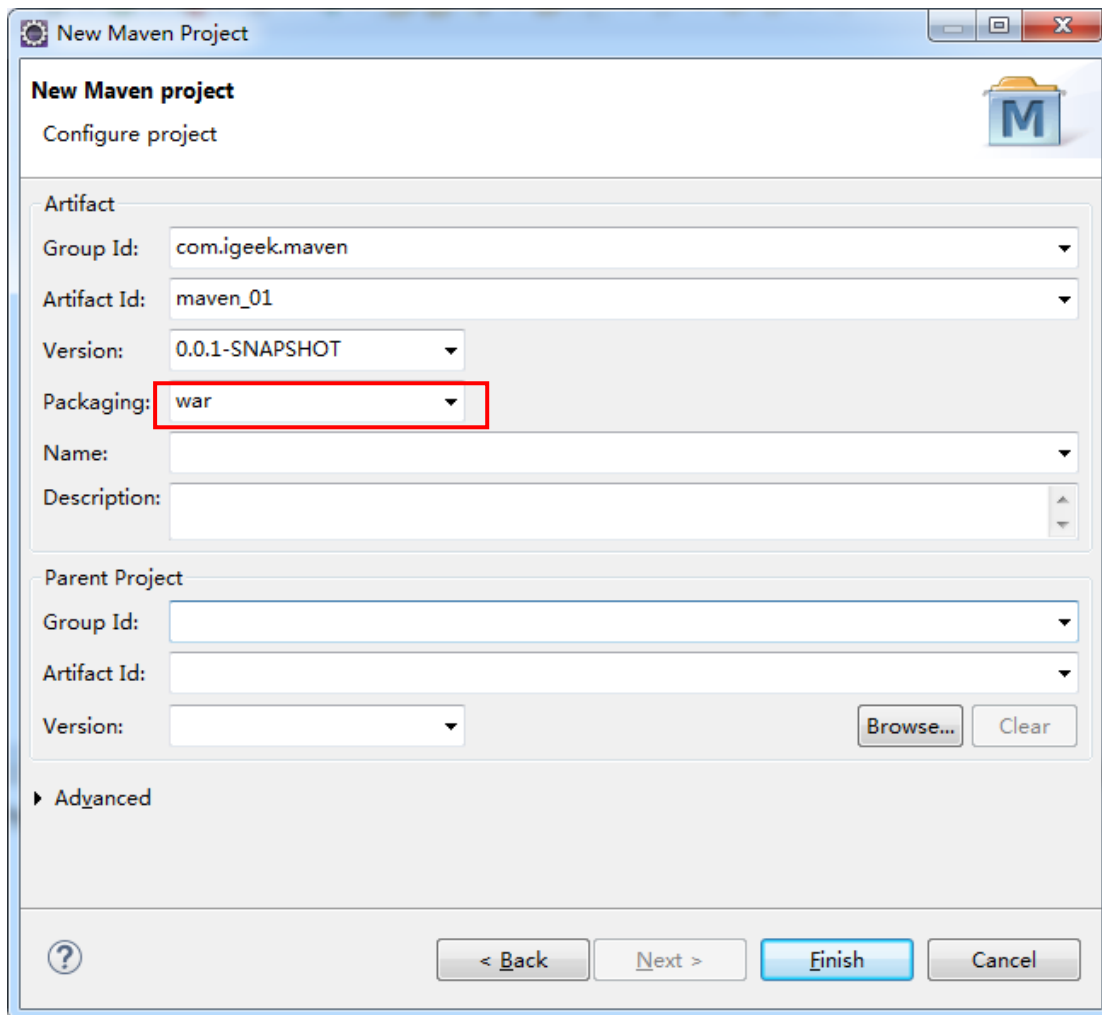
当然我们也可以了解一下 maven 的骨架：

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



The image shows the 'New Maven Project' dialog box in an IDE. The 'Artifact' section is expanded, showing fields for Group Id (com.igeek.maven), Artifact Id (maven_01), Version (0.0.1-SNAPSHOT), and Packaging (war, which is highlighted with a red rectangle). Below this is the 'Parent Project' section with empty fields for Group Id, Artifact Id, and Version, along with 'Browse...' and 'Clear' buttons. At the bottom, there is an 'Advanced' section (collapsed) and navigation buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

创建的工程结构如下：

- └─ maven_01
 - ├─ src/main/java
 - ├─ src/main/resources
 - ├─ src/test/java
 - ├─ src/test/resources
 - ├─ JRE System Library [J2SE-1.5]
 - ├─ src
 - ├─ target
 - └─ pom.xml

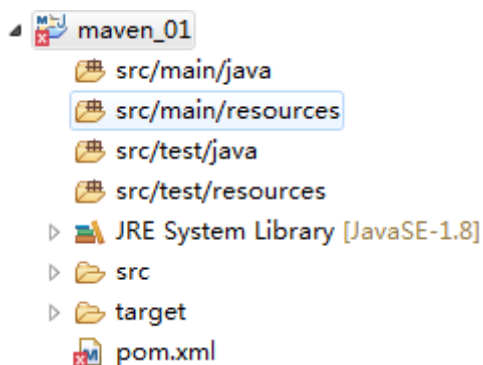
3.4.3 第三步设置编译版本

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 {

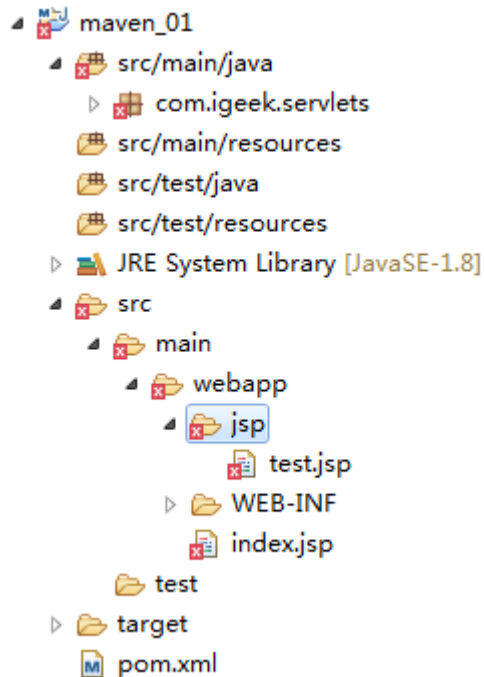
        this.doPost(req, resp);
    }

    @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：

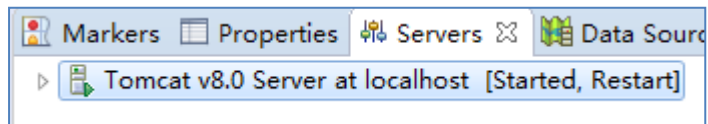
```
<!-- 配置servlet -->
<servlet>
  <servlet-name>servletTest</servlet-name>
  <servlet-class>com.igeek.servlets.ServletTest</servlet-class>
</servlet>
<servlet-mapping>
```

```
<servlet-name>servletTest</servlet-name>
<url-pattern>/test</url-pattern>
</servlet-mapping>
```

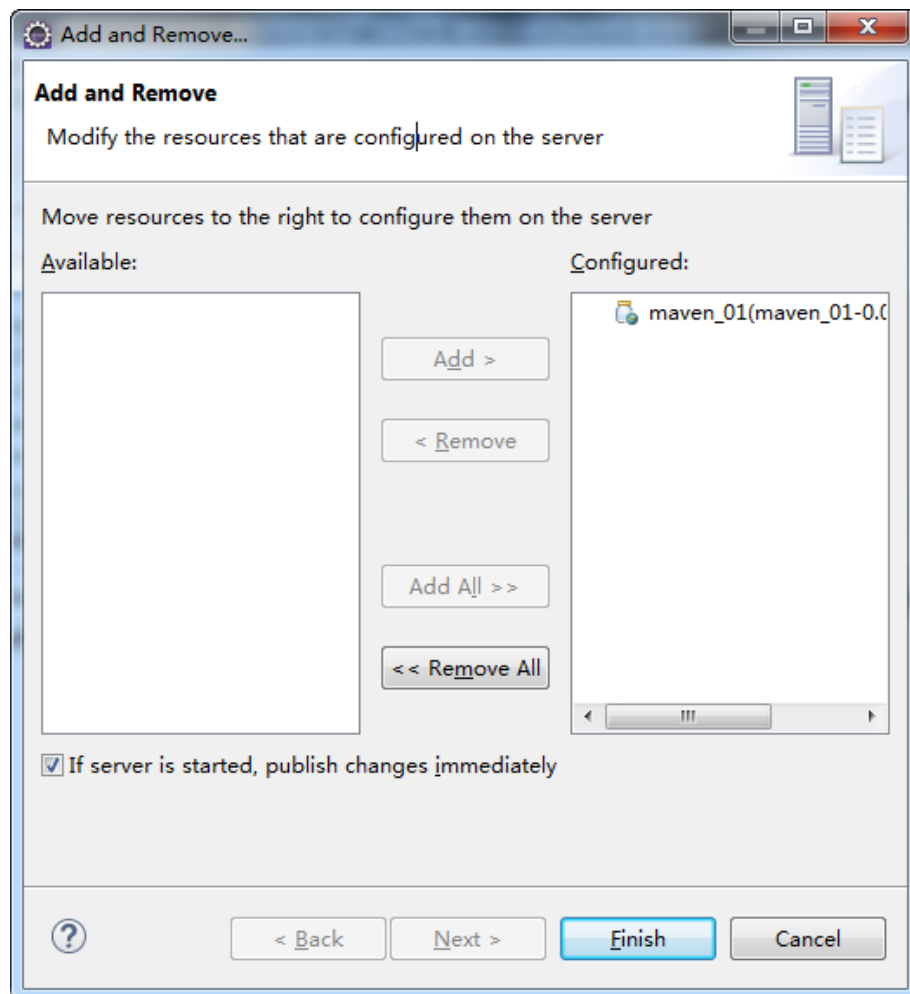
3.4.9 运行

3.4.9.1 使用 tomcat 服务器来发布

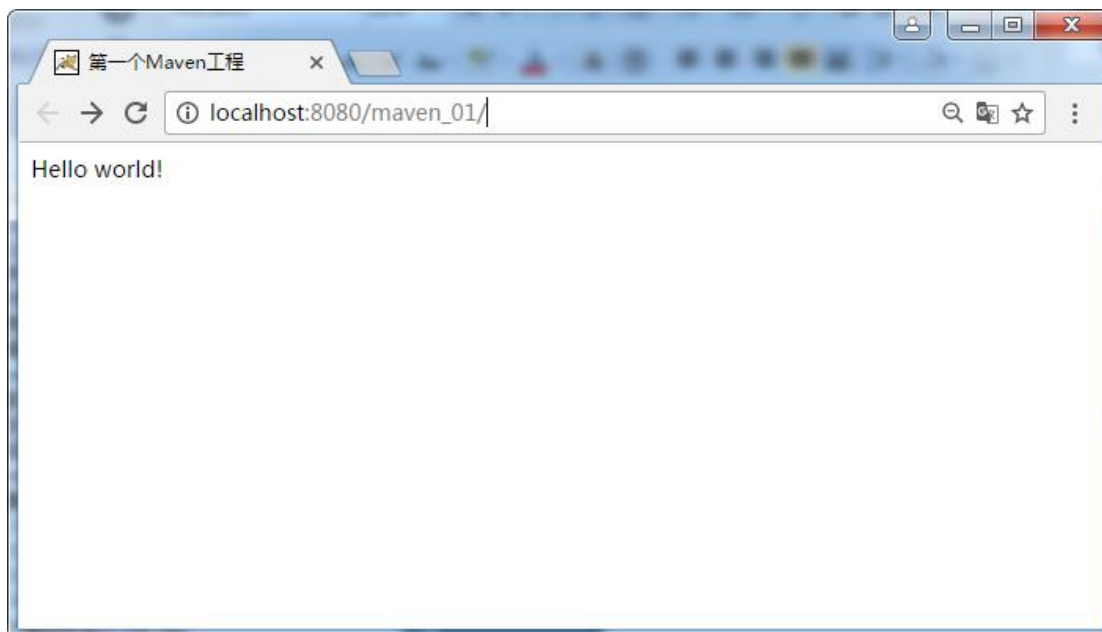
Maven 的 web 工程可以和之前的动态 web 工程一样，使用 Tomcat 服务器来发布和运行，这样我们需要首先添加一个 Tomcat 服务器。



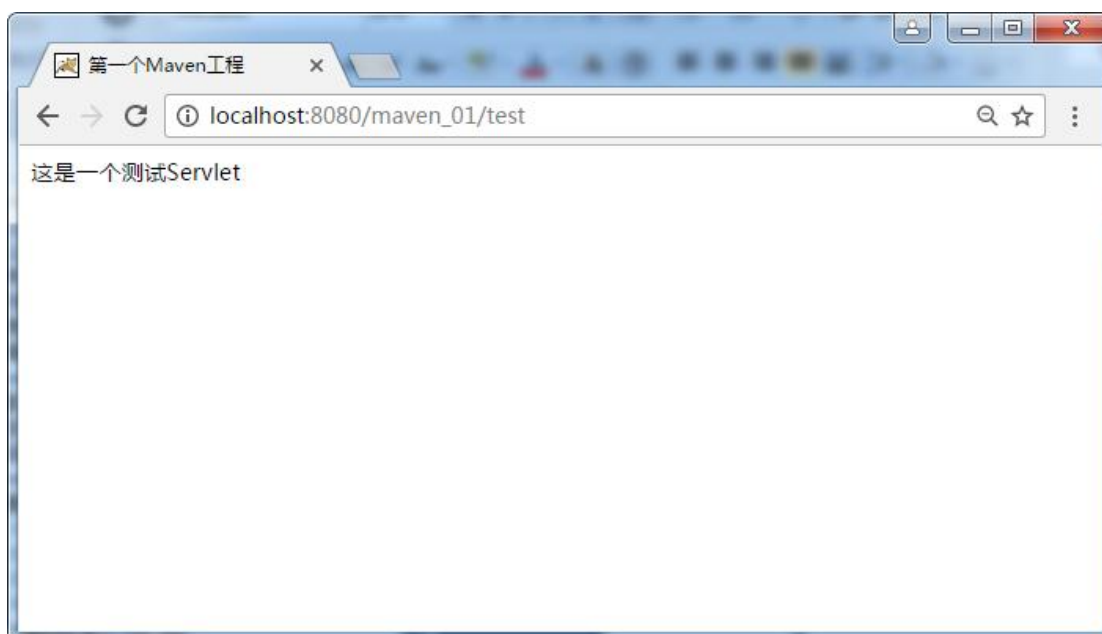
然后将我们的工程添加到 Tomcat 服务器中。



启动 Tomcat 服务器之后就可以访问我们的页面了。



访问 test 页面。



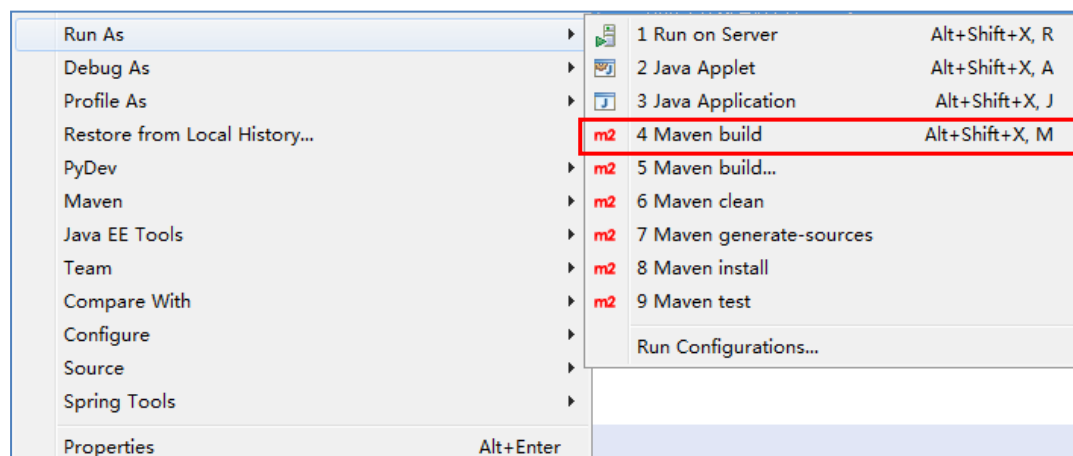
3.4.9.1 使用 Maven 的服务器插件来发布

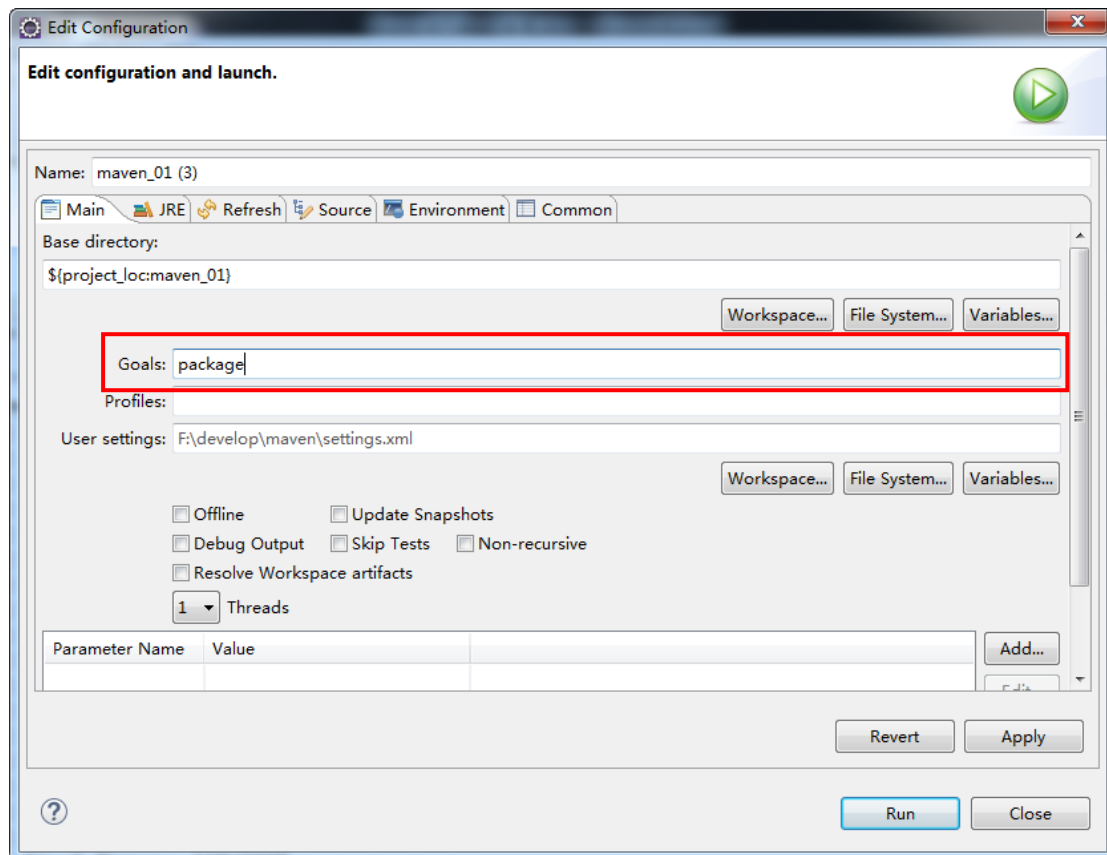
我们也可以使用 maven 的 Tomcat 插件来发布工程，我们需要首先添加该插件。可以通过配置 plugin 修改 tomcat 的访问路径及端口。

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>
```

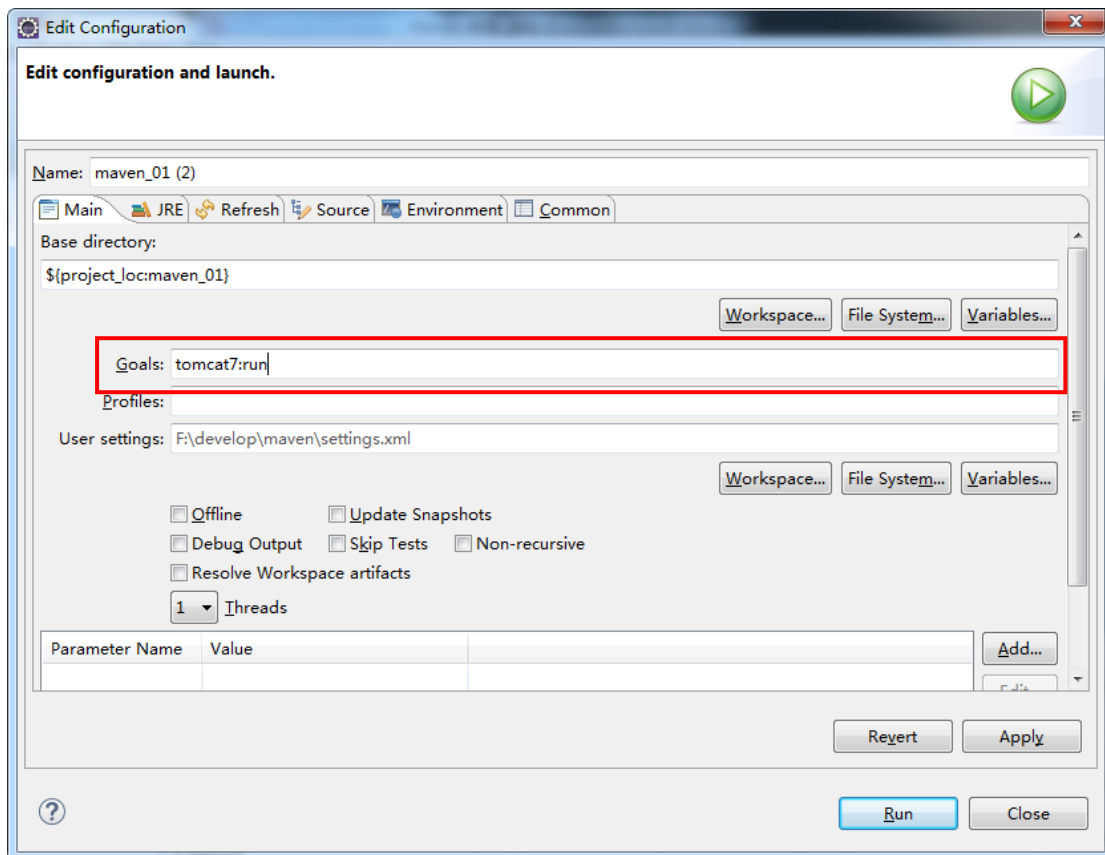
在启动 tomcat 插件发布工程前，需要首先对工程进行编译和打包，我们使用命令 package。





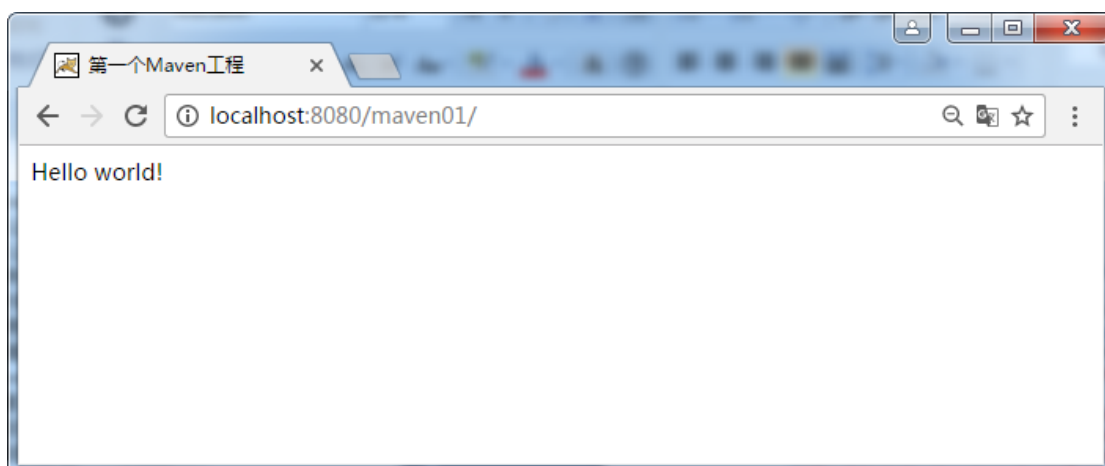
```
[INFO] Packaging webapp
[INFO] Assembling webapp [maven_01] in [F:\workspace\maven_01\target\maven_01-0.0.1-SNAPSHOT]
[INFO] Processing war project
[INFO] Copying webapp resources [F:\workspace\maven_01\src\main\webapp]
[INFO] Webapp assembled in [29 msecs]
[INFO] Building war: F:\workspace\maven_01\target\maven_01-0.0.1-SNAPSHOT.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.124 s
[INFO] Finished at: 2017-12-28T11:17:21+08:00
[INFO] Final Memory: 11M/155M
[INFO] -----
```

接下来就可以启动服务了，我们使用 tomcat:run 命令。

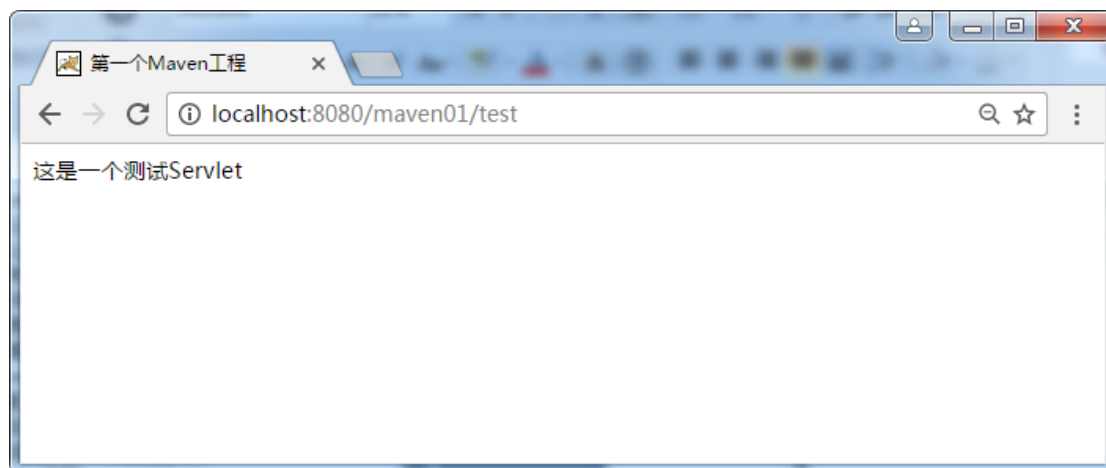


```
[INFO] --- tomcat-maven-plugin:1.1:run (default-cli) @ maven_01 ---
[INFO] Running war on http://localhost:8080/maven01
[INFO] Using existing Tomcat server configuration at F:\workspace\maven_01\target\tomcat
十二月 28, 2017 11:19:21 上午 org.apache.catalina.startup.Embedded start
信息: Starting tomcat server
十二月 28, 2017 11:19:21 上午 org.apache.catalina.core.StandardEngine start
信息: Starting Servlet Engine: Apache Tomcat/6.0.29
十二月 28, 2017 11:19:21 上午 org.apache.coyote.http11.Http11Protocol init
信息: Initializing Coyote HTTP/1.1 on http-8080
十二月 28, 2017 11:19:21 上午 org.apache.coyote.http11.Http11Protocol start
信息: Starting Coyote HTTP/1.1 on http-8080
```

服务器启动完成，接下来就可以访问页面了。



访问 test.jsp 页面。



4 Maven 的命令和周期

4.1 Maven 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

- | |
|--|
| <ol style="list-style-type: none">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生命周期当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。

5 依赖管理

5.1 添加依赖

5.1.1 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

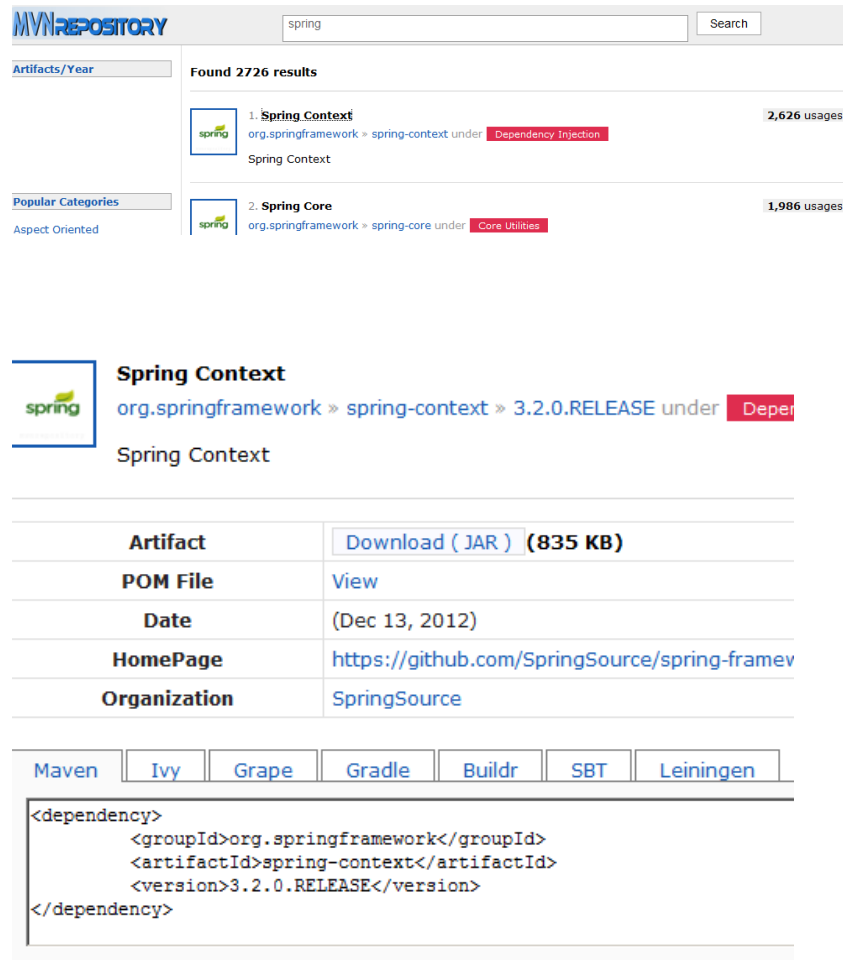
添加依赖需要指定依赖 jar 包的坐标，但是很多情况我们是不知道 jar 包的坐标，可以通过如下方式查询：

■ 方法一：从互联网搜索

<http://search.maven.org/>

<http://mvnrepository.com/>

网站搜索示例：

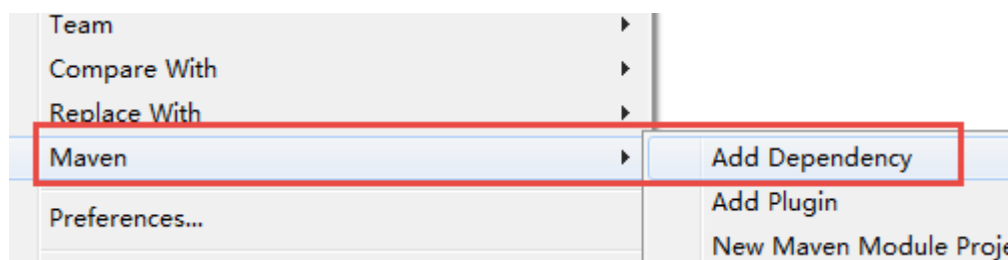


The screenshot shows the Maven Repository search results for the keyword 'spring'. The search bar at the top contains 'spring' and a 'Search' button. Below the search bar, it says 'Found 2726 results'. The results are listed in a table with columns for 'Artifact', 'Download (JAR)', and 'POM File'. The first result is 'Spring Context' (org.springframework » spring-context) with 2,626 usages. The second result is 'Spring Core' (org.springframework » spring-core) with 1,986 usages. Below the search results, there is a section for 'Spring Context' with a 'Download (JAR)' button (835 KB) and a 'POM File' link. Below this, there is a table with columns for 'Artifact', 'Download (JAR)', 'POM File', 'Date', 'HomePage', and 'Organization'. The table contains one row for 'Spring Context' with the following values: 'Download (JAR)' (835 KB), 'POM File' (View), 'Date' (Dec 13, 2012), 'HomePage' (https://github.com/SpringSource/spring-framev), and 'Organization' (SpringSource). Below the table, there are tabs for 'Maven', 'Ivy', 'Grape', 'Gradle', 'Buildr', 'SBT', and 'Leiningen'. The 'Maven' tab is selected, and it shows the following XML snippet:

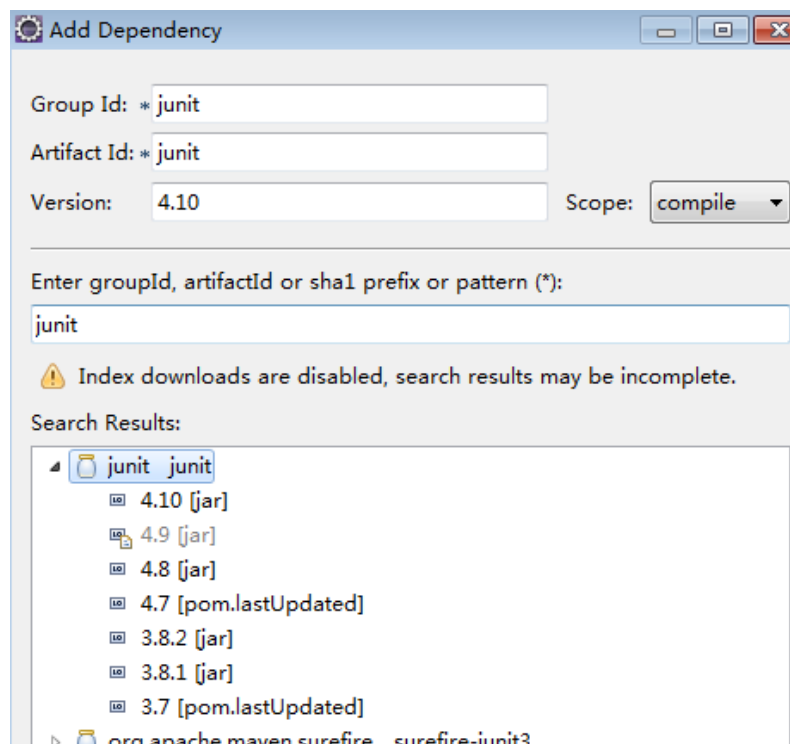
```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>3.2.0.RELEASE</version>
</dependency>
```

■ 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



The screenshot shows the Maven context menu. The menu items are: 'Team', 'Compare With', 'Replace With', 'Maven', and 'Preferences...'. The 'Maven' item is highlighted, and a sub-menu is open with the following options: 'Add Dependency', 'Add Plugin', and 'New Maven Module Project'. The 'Add Dependency' option is highlighted with a red box.



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	对于编译 classpath 有效	对于测试 classpath 有效	对于运行时 classpath 有效	例子
compile	Y	Y	Y	spring-core
test	-	Y	-	Junit
provided	Y	Y	-	servlet-api
runtime	-	Y	Y	JDBC驱动
system	Y	Y	-	本地的， Maven仓库之 外的类库

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

6 maven 工程运行调试

6.1 端口占用处理

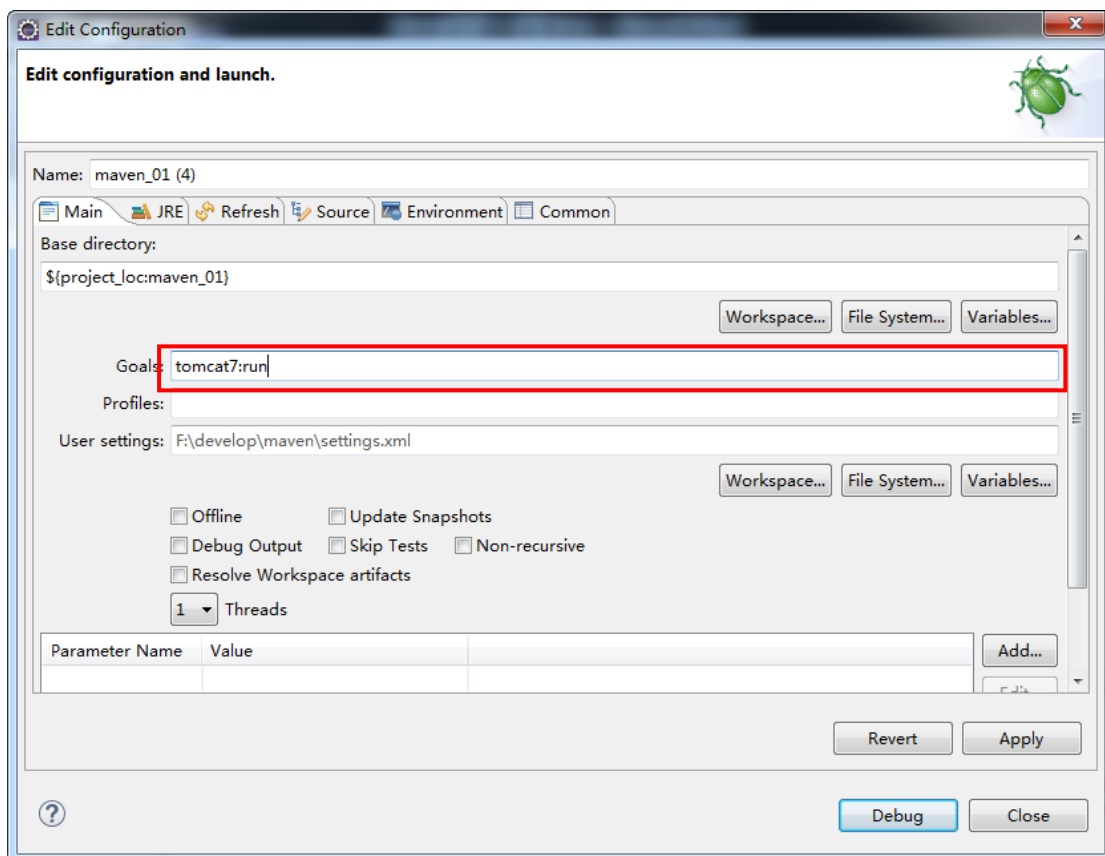
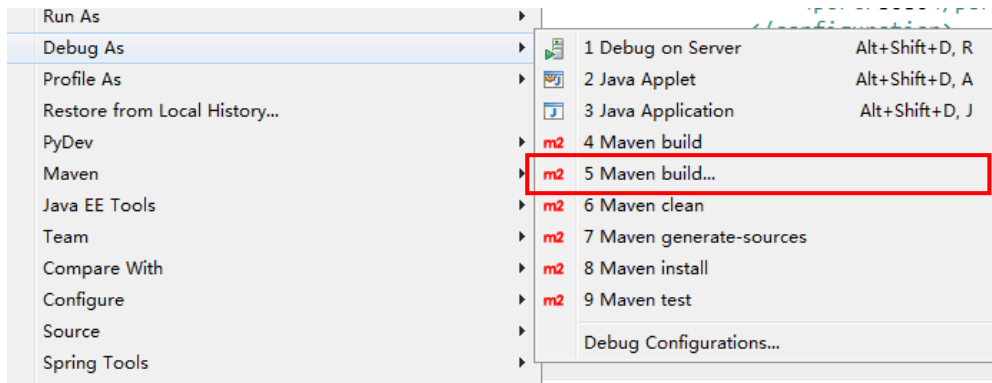
重新执行 tomcat:run 命令重启工程，重启之前需手动停止 tomcat，否则报下边的错

误：Caused by: [java.net.BindException](#): Address already in use: JVM_Bind

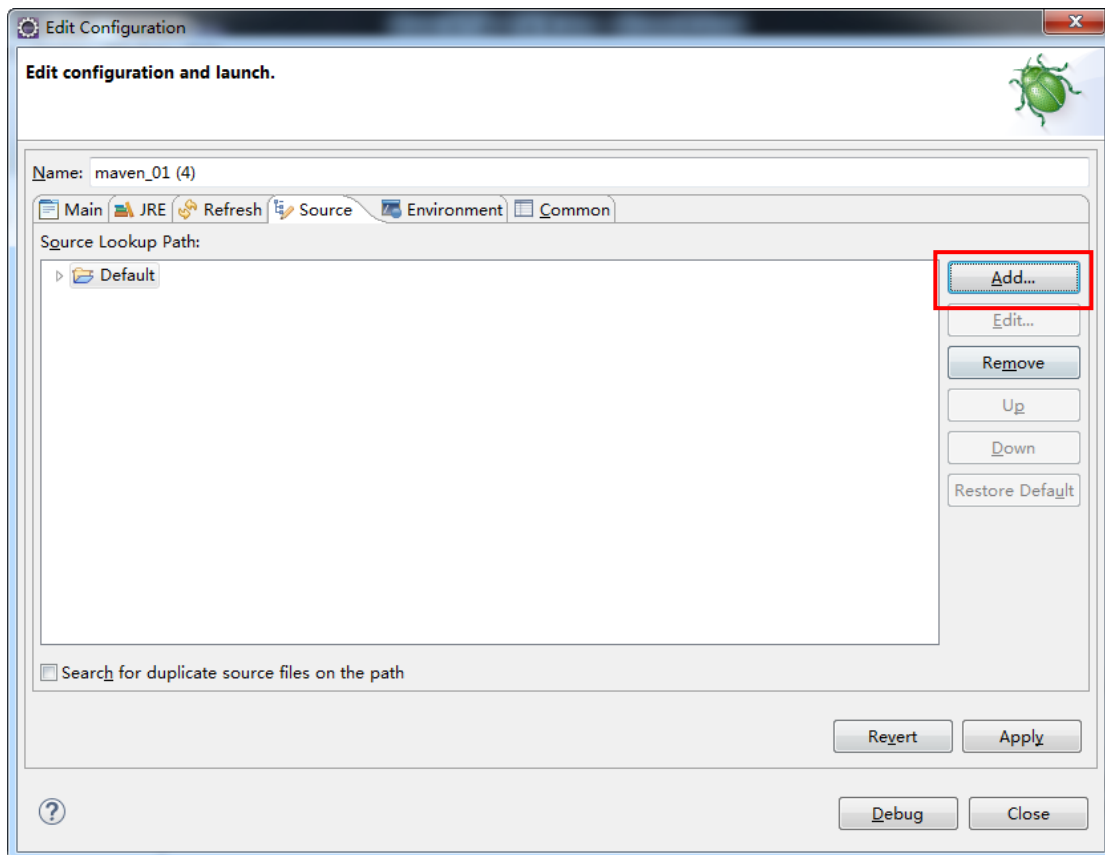
6.2 断点调试

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

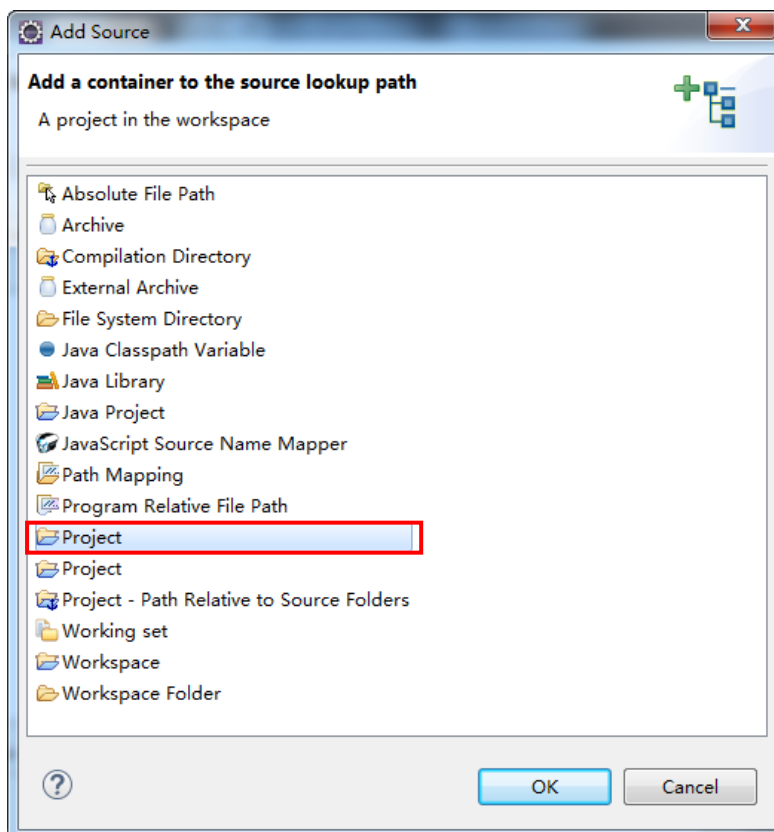
跟踪：

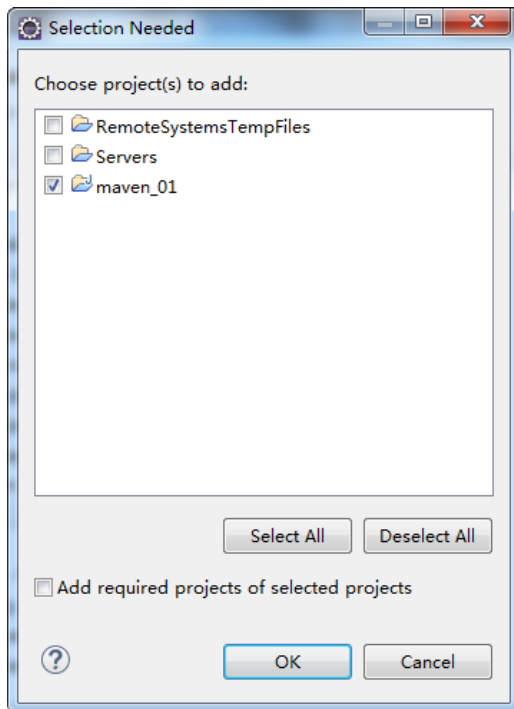


引入源码：



添加，选择本工程：





以 debug 方式运行：

