

# Maven 学习笔记

2010-6-25

# 目录

目录	1
前言	1
一、知识准备	2
1.1 项目模型 POM	2
1.2 生命周期	2
1.3 安装及测试	2
1.4 帮助命令	2
二、MAVEN 入门和初步进阶	4
2.1 入门例子	4
2.2 进阶例子——Simple Weather	5
2.2.1 项目说明	5
2.2.2 创建项目	5
2.2.3 为 pom 增加组织、法律和开发人员信息	5
2.2.4 Eclipse 版本——为项目增加新的依赖——Maven 镜像	5
2.2.5 Eclipse 版本——为项目增加新的依赖——Eclipse 导入 Nexus Index	6
2.2.6 Eclipse 版本——为项目增加新的依赖——可视化导入包	7
2.2.7 Eclipse 版本——为项目增加新的依赖——实现依赖包导入	8
2.2.8 手工版本——为项目增加新的依赖——POM 写入依赖包	9
2.2.9 手工版本——为项目增加新的依赖——远程下载包	10
2.3 源码	10
2.3.1 源码功能介绍	10
2.3.2 依次加入类	11
2.3.3 添加外部项目资源	12
2.3.4 运行 main	14
2.3.5 关于 exec 插件和项目依赖介绍	15
2.3.6 单元测试	15
2.3.7 构建一个大包好的命令行应用程序	15
三、MAVEN——WEB 简单应用	17
3.1 生成项目	17
3.2 配置 jetty——命令行启动和 Eclipse 启动	18

3.3 添加一个 servlet——添加 J2EE 依赖 -----20

3.3.1 添加 servlet2.5 规格说明作为依赖 -----20

3.3.2 添加 jsp2.0 规格说明作为依赖-----20

3.3.3 添加一个 servlet -----21

3.3.4 配置 web.xml-----21

3.3.5 运行 -----22

四、MAVEN——WEB 应用进阶——简单多模块项目 ----- 23

4.1 创建项目 -----23

4.2 父级项目 -----26

4.3 加入 module-----27

4.4 simple\_weather 模块-----30

4.5 webapp 模块-----31

附录 ----- 41

1、手工安装第三方 jar-----41

2、更新本地索引-----41

# 前言

本文针对 Maven2.1.0 。其余版本不再本文叙述范围内，特此声明。

本文内容为是一份名为《Maven 权威指南中文版》的文档为基础进行的学习记录。

# 一、知识准备

## 1.1 项目模型 POM

groupId: artifactId: packaging: version

对应 Maven 仓库存放目录结构:

/groupId/artifactId/version/artifactId-version.packaging

## 1.2 生命周期

lifecycle——phase——goal

## 1.3 安装及测试

解压 Maven2.1.0 至任何目录，配置环境变量:

新增 M2\_HOME=Maven 的解压目录

修改 PATH=Maven 的 bin 目录

新增 MAVEN\_OPTS=-Xms256m -Xmx512m

重启电脑，在命令行输入 mvn -version，如果显示版本号即为安装成功。

## 1.4 帮助命令

mvn help:describe {[-Dcmd=][-Dplugin=][-DgroupId= -DartifactID=]}

如果需要详细信息可在最后加上“-Ddetail”。

## 二、Maven 入门和初步进阶

### 2.1 入门例子

用 Maven 命令生成一个最简单的项目。

命令行如下：

```
#####  
D:\NewWorld>mvn archetype:create -DgroupId=com.mycom.test  
  
-DartifactId=simple  
  
#####
```

在【NewWorld】目录下生成名为“simple”的 Maven 项目。为方便查看，把生成的项目转为 Eclipse 项目（Eclipse 插件已安装相关插件）。

但是，在 Maven 不熟悉的情况下，还是通过最普通的资源管理器方式查看以及用命令行方式执行 Maven 命令，不建议现在就使用 Eclipse!!!

进入到带有【.pom】文件的目录级别，基本上就是在项目的跟目录下，之后对项目处理的命名均是在这个目录下执行，执行命令行如下：

```
#####  
D:\NewWorld\simple>mvn eclipse:eclipse  
  
#####
```

使用 Eclipse 直接导入 simple 项目即可。

在此，可以用命令行输入“mvn site”用于生成项目站点（实际上就是一个网页格式项目说明书）、“mvn package”用于项目打包、“mvn install”用于项目安装至 Maven 仓库（目录结构在第一部分以提及）。

## 2.2 进阶例子——Simple Weather

### 2.2.1 项目说明

一个基本命令行驱动的应用程序，接受邮件编码输入，然后从“Yahoo!Weather RSS”源获取数据，然后解析数据并把结果打印输出。

### 2.2.2 创建项目

命令行如下：

```
#####  
D:\NewWorld>mvn archetype:create -DgroupId=com.mycom.test  
-DartifactId=Simple-Weather  
#####
```

### 2.2.3 为 pom 增加组织、法律和开发人员信息

在 pom 的根中包含标签<licenses><organisation><developers>（具体内容略）

### 2.2.4 Eclipse 版本——为项目增加新的依赖——Maven 镜像

参看同系列文档《nexus 仓库管理》，本文在这里配置了一个 Nexus 镜像，用户级 setting.xml 镜像部分增加代码如下：

```
#####  
<mirrors>
```



```
<mirror>

  <id>Nexus</id>

  <mirrorOf>central</mirrorOf>

  <name>Nexus-My-Test</name>

  <url> http://127.0.0.1:8081/nexus/content/groups/public/</url>

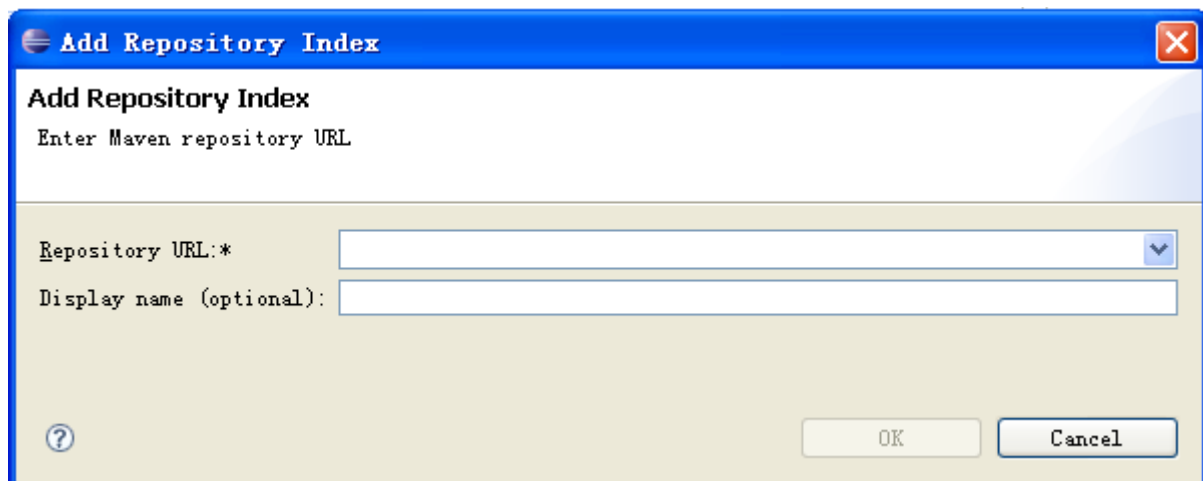
</mirror>

</mirrors>
```

#####

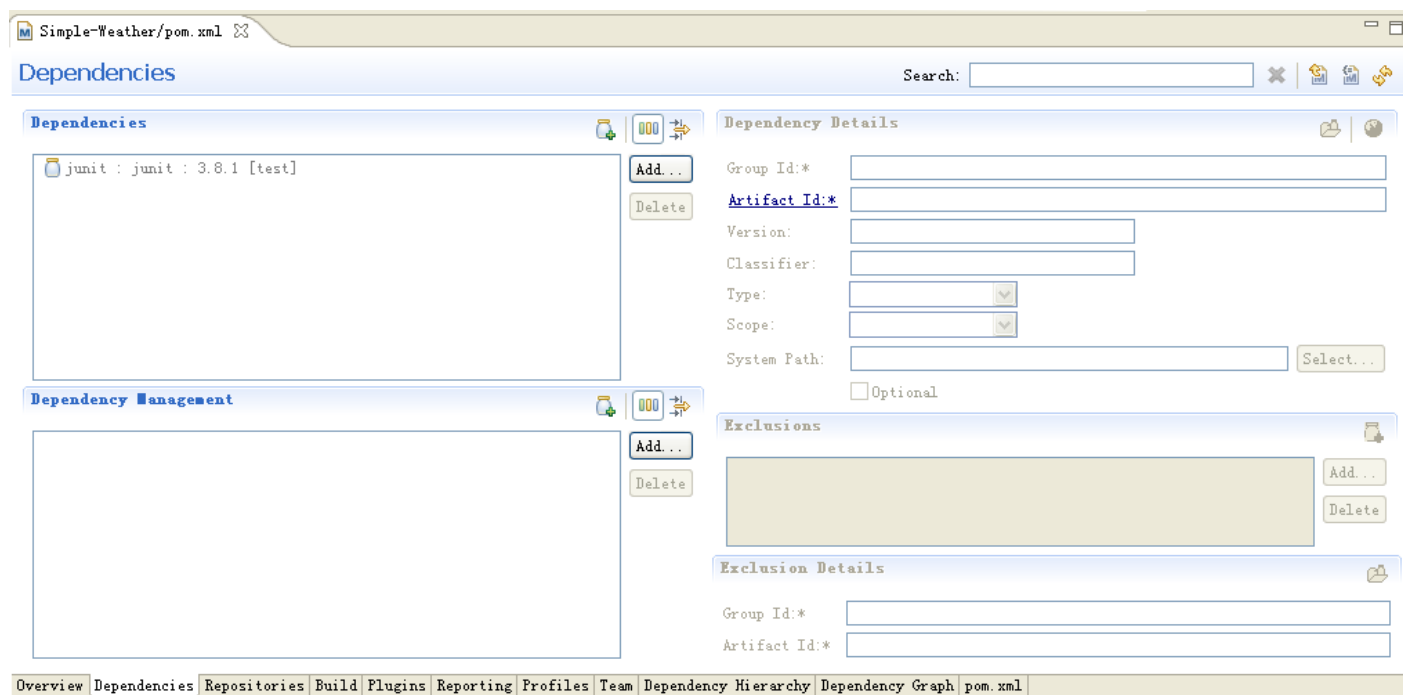
## 2.2.5 Eclipse 版本——为项目增加新的依赖——Eclipse 导入 Nexus Index

在 Eclipse **【windows】 - 【show view】 - 【other】** 找到 **【Maven】 - 【Index】**。  
将出现一个 Maven-Index 控制台。在控制台中右击选择添加索引。

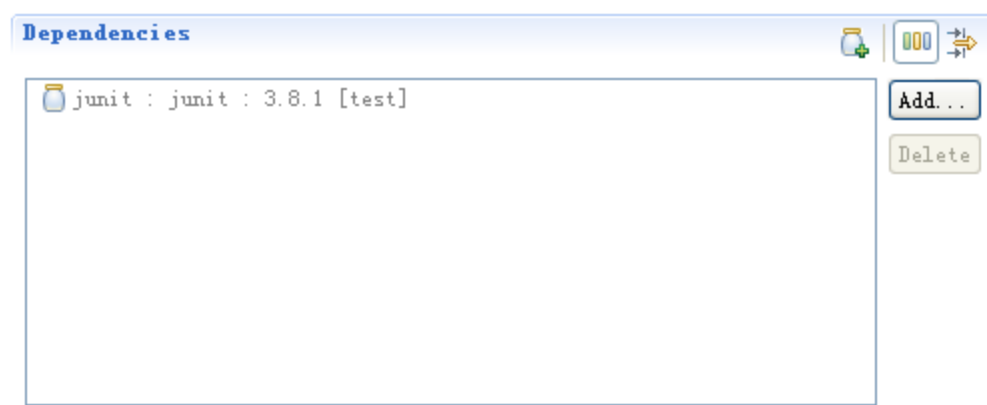


输入 Nexus 对应的仓库地址。最后刷新即可看到一堆包。这个时候打开 pom 就可以方便地有提示地输入相应的依赖包了。

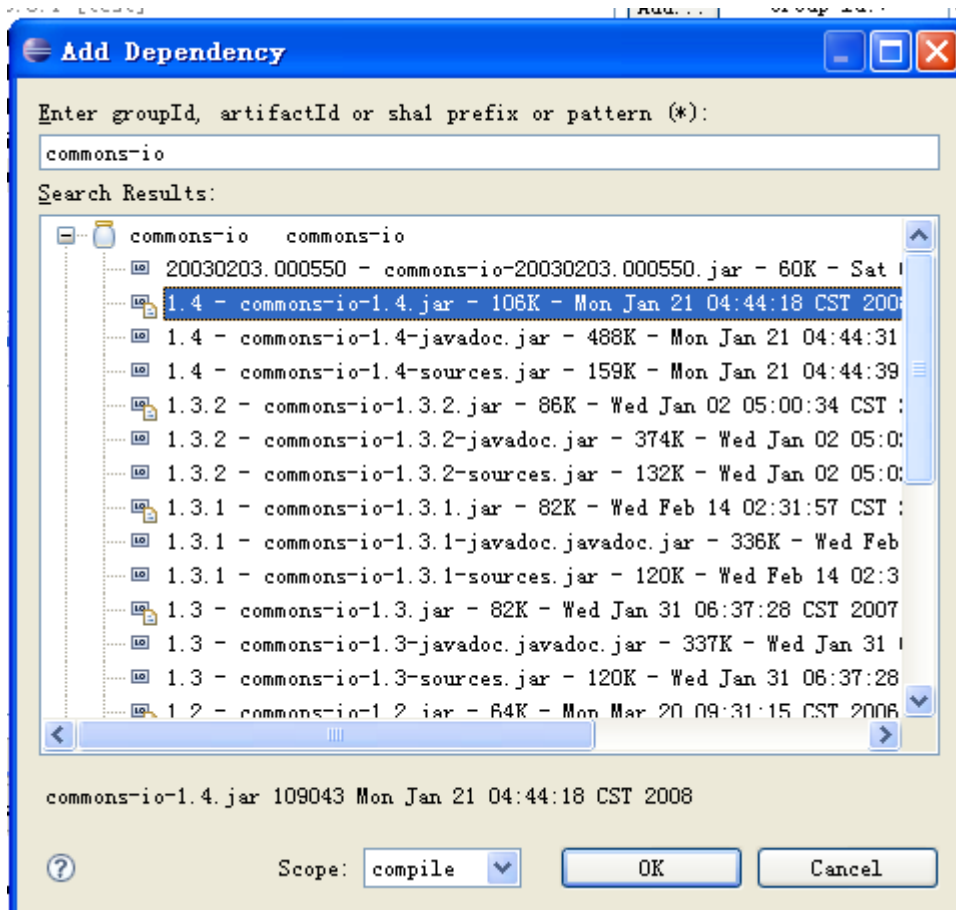
## 2.2.6 Eclipse 版本——为项目增加新的依赖——可视化导入包



点击下图右上角那个小瓶子增加。



在增加的时候注意，默认会选择最新的版本，但是经常最新版本是一个非正式版，这种版本通常不能正常使用，所以，要注意版本号，然后选择。



## 2.2.7 Eclipse 版本——为项目增加新的依赖——实现依赖包导入

实际上，上面在 pom 中加入依赖并非一定要使用 Eclipse，只是这样加入更加的方便和准确，其实是可以手工慢慢敲入的。

pom 加入依赖之后，在命令行，执行

```
#####  
D:\NewWorld\Simple-Weather>mvn install  
#####
```

如果你 Maven 本地仓库中没有这些依赖包，则会提示下载。这样便实现了依赖导入了。

## 2.2.8 手工版本——为项目增加新的依赖——POM 写入依赖包

在已有的 pom 文件里面找到 junit 依赖的地方，放入如下代码：

```
#####
```

```
<dependency>
```

```
    <groupId>log4j</groupId>
```

```
    <artifactId>log4j</artifactId>
```

```
    <version>1.2.14</version>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>dom4j</groupId>
```

```
    <artifactId>dom4j</artifactId>
```

```
    <version>1.6.1</version>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>jaxen</groupId>
```

```
    <artifactId>jaxen</artifactId>
```

```
    <version>1.1.1</version>
```

```
</dependency>
```

```
<dependency>
```

```
    <groupId>velocity</groupId>
```

```
    <artifactId>velocity</artifactId>
```

```
    <version>1.5</version>
```

```
</dependency>
```

```

<dependency>

    <groupId>org.apache.commons</groupId>

    <artifactId>commons-io</artifactId>

    <version>1.3.2</version>

    <scope>test</scope>

</dependency>

```

```
#####
```

## 2.2.9 手工版本——为项目增加新的依赖——远程下载包

上面的 Eclipse 版本在 maven 中建了镜像，使用了 nexus 作为远程仓库（也有人叫私服），新增的依赖包会先从镜像中搜寻下载。现在不使用镜像，即把 maven 配置中的镜像配置删除，之后，直接使用

```
#####
```

```
mvn install
```

```
#####
```

这样 maven 会从远程库，也就是默认配置中的 <http://repol.maven.org/maven2/> 搜寻下载。

## 2.3 源码

### 2.3.1 源码功能介绍

这里由于更改了包名，所以学习需要对应各个包。

Simple Weather 命令行应用程序包含五个 Java 类。

`org.sonatype.mavenbook.weather.Main`

这个类包含了一个静态的 `main()` 函数，即系统的入口。

`org.sonatype.mavenbook.weather.Weather`

`Weather` 类是个很简单的 **Java Bean**，它保存了天气报告的地点和其它一些关键元素，如气温和湿度。

`org.sonatype.mavenbook.weather.YahooRetriever`

`YahooRetriever` 连接到 **Yahoo! Weather** 并且返回来自数据源数据的 `InputStream`。

`org.sonatype.mavenbook.weather.YahooParser`

`YahooParser` 解析来自 **Yahoo! Weather** 的 XML，返回 `Weather` 对象。

`org.sonatype.mavenbook.weather.WeatherFormatter`

`WeatherFormatter` 接受 `Weather` 对象，创建 `VelocityContext`，根据 **Velocity** 模板生成结果。

由于方便检查错误以及编译方便，这里还是使用 Eclipse，但是注意，使用的是上一步骤里面的手工版本添加依赖，即不使用 maven 镜像。

执行：

```
#####  
mvn eclipse:eclipse  
#####
```

然后导入到 Eclipse 中。

## 2.3.2 依次加入类

从源码中复制到项目。

- 1、`weather.java`
- 2、`main.java`

3、YahooRetriever.java

4、YahooParser.java

5、WeatherFormatter.java

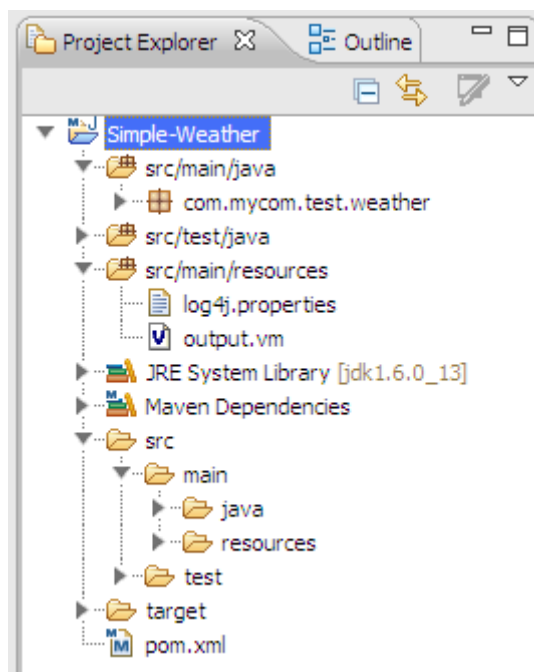
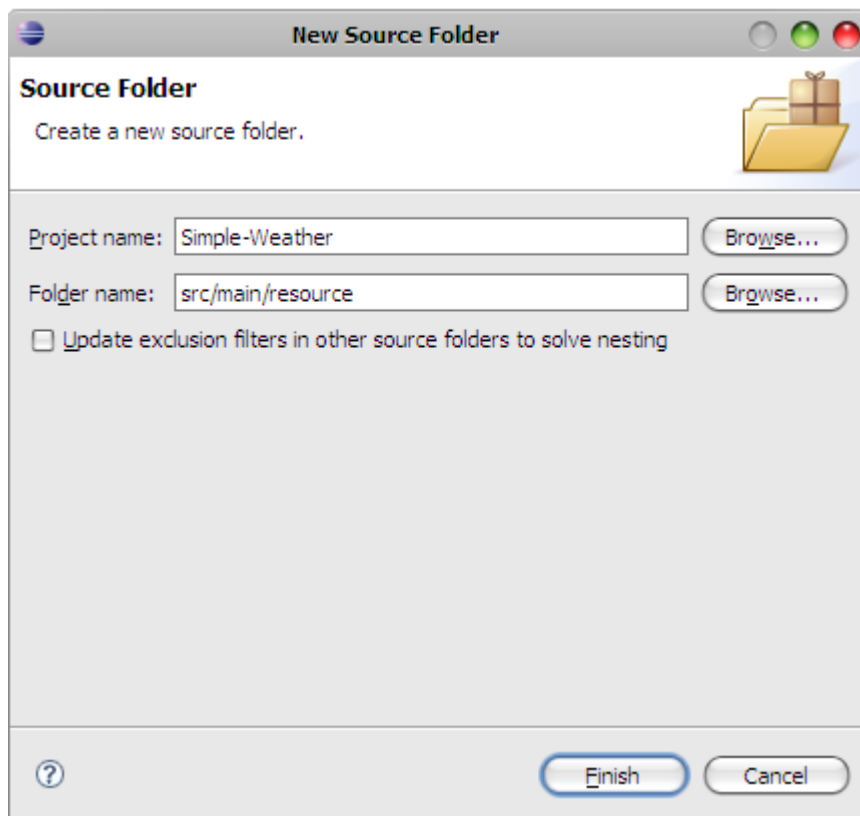
### 2.3.3 添加外部项目资源

本项目依赖于两个 `classpath` 资源： `Main` 类通过 `classpath` 资源 `log4j.preoperties` 来配置 `Log4J`， `WeatherFormatter` 引用了一个在 `classpath` 中的名为 `output.vm` 的 `Velocity` 模板。这两个资源都需要在默认包中（或者 `classpath` 的根目录）。

为了添加这些资源，我们需要在项目的基础目录下创建一个新的目录——`src/main/resources`。由于任务 `archetype:create` 没有创建这个目录，我们需要通过在项目的基础目录下运行下面的命令来创建它：

在【`main`】目录下新建文件夹【`resources`】，把对应的【`log4j.properties`】、【`output.vm`】文件放入到该文件夹中。

注意，如果使用 `Eclipse`，对于新手，很容易误认为直接在 `src/main` 下面加个 `folder` 就 OK 的，实际上应该是增加一个“`source folder`”！然后再把文件复制进来。



如上，那个 source folder 是类似 package 有个包的！当 source folder 加入以后，在下面的 src 普通 folder 会同步出现一个名叫【resources】的文件夹。



## 2.3.4 运行 main

执行：

```
#####
```

```
mvn install
```

```
#####
```

之后执行(这里的空间由于更换了电脑关系，名字变了，但不会有影响。)

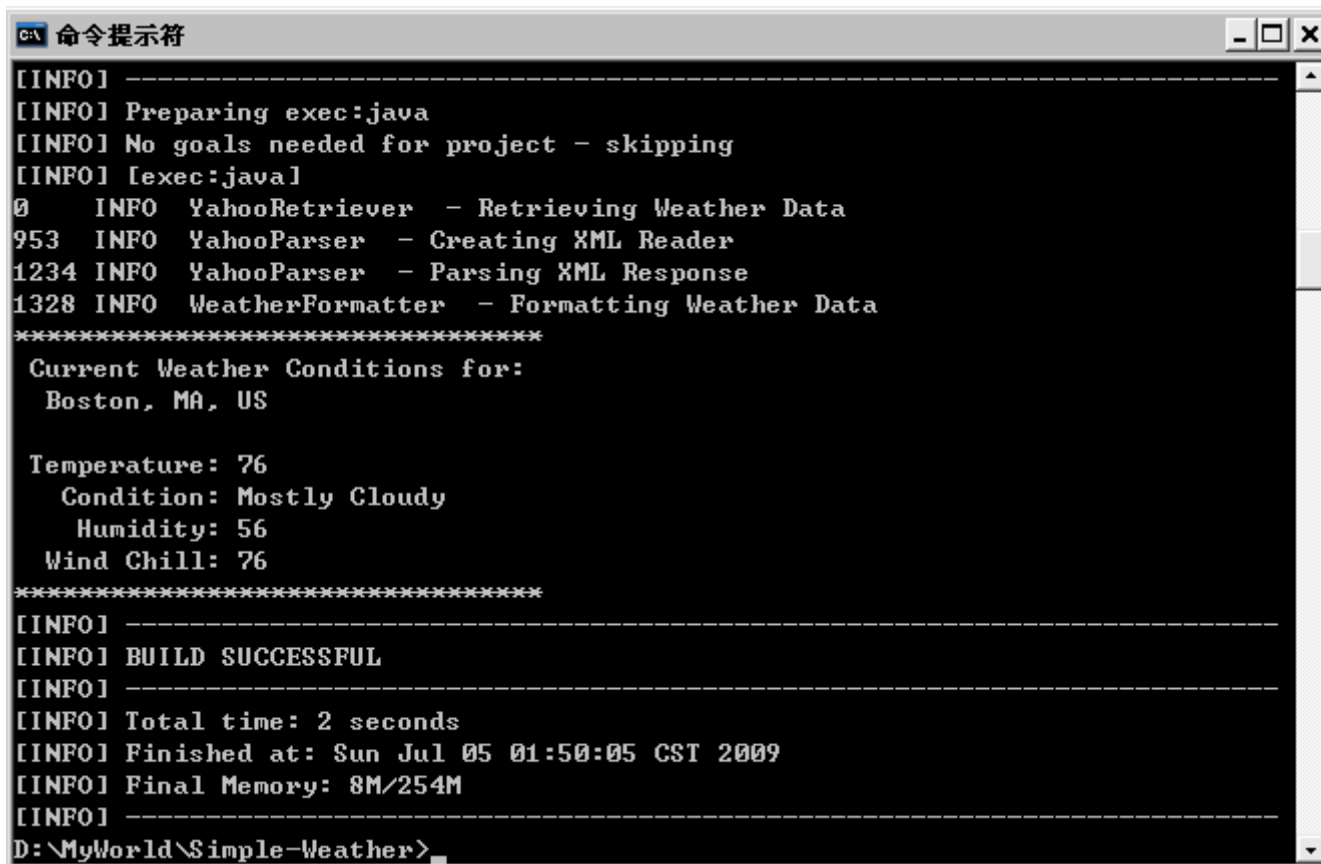
```
#####
```

```
D:\MyWorld\Simple-Weather>mvn exec:java
```

```
-Dexec.mainClass=com.mycom.test.weather.Main
```

```
#####
```

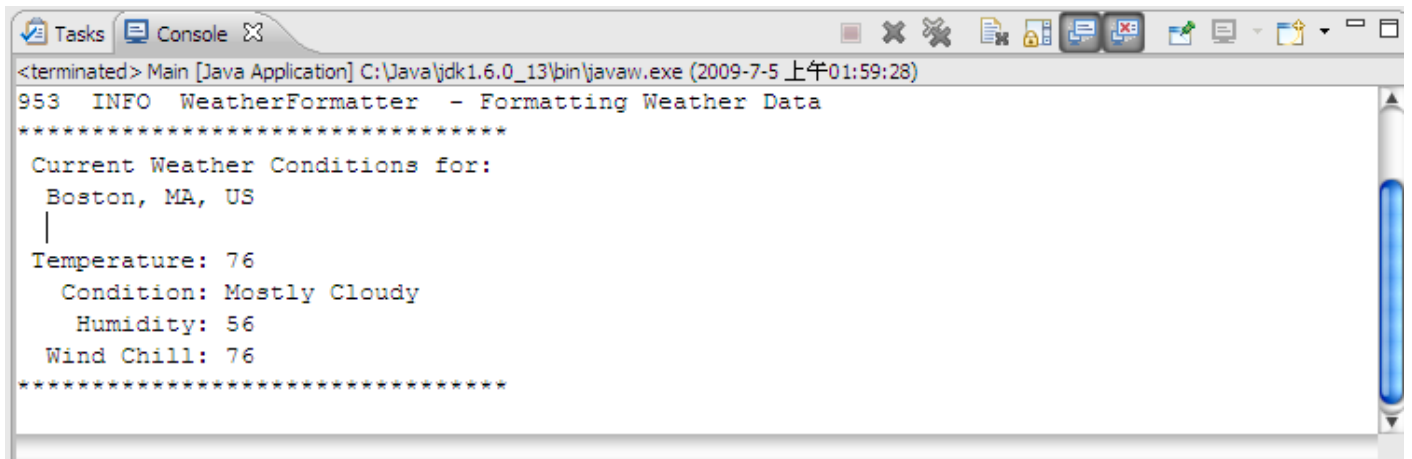
如果正确会出现



```
命令提示符
[INFO] -----
[INFO] Preparing exec:java
[INFO] No goals needed for project - skipping
[INFO] [exec:java]
0      INFO  YahooRetriever - Retrieving Weather Data
953    INFO  YahooParser - Creating XML Reader
1234   INFO  YahooParser - Parsing XML Response
1328   INFO  WeatherFormatter - Formatting Weather Data
*****
Current Weather Conditions for:
  Boston, MA, US

Temperature: 76
  Condition: Mostly Cloudy
    Humidity: 56
  Wind Chill: 76
*****
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 2 seconds
[INFO] Finished at: Sun Jul 05 01:50:05 CST 2009
[INFO] Final Memory: 8M/254M
[INFO] -----
D:\MyWorld\Simple-Weather>
```

如果用 Eclipse，在 install 之后，也可以像普通 Java 应用程序一样运行



```
<terminated> Main [Java Application] C:\Java\jdk1.6.0_13\bin\javaw.exe (2009-7-5 上午01:59:28)
953 INFO WeatherFormatter - Formatting Weather Data
*****
Current Weather Conditions for:
Boston, MA, US
|
Temperature: 76
Condition: Mostly Cloudy
Humidity: 56
Wind Chill: 76
*****
```

### 2.3.5 关于 exec 插件和项目依赖介绍

具体对照《maven 权威指南》相应的部分。

### 2.3.6 单元测试

(略)

### 2.3.7 构建一个大包好的命令行应用程序

从上面看到，要运行这个项目，要么要使用 maven，要么要在 Eclipse 里面，外部依赖是没有加入到 jar 中的。

maven assembly 插件就有这个功能。

在 pom 中配置 maven assembly 描述符（具体位置查看源码）

```
#####

<build>

    <plugins>
```

```
<plugin>

  <artifactId>maven-assembly-plugin</artifactId>

  <configuration>

    <descriptorRefs>

      <descriptorRef>jar-with-dependencies</descriptorRef>

    </descriptorRefs>

  </configuration>

</plugin>

</plugins>

</build>
```

#####

之后执行

#####

```
mvn install assembly:assembly
```

#####

如果成功，会生成一个比较大的包，会把依赖都加进来了。但是这个不是很好的做法！据说后面会有改进做法。

## 三、Maven——web 简单应用

### 3.1 生成项目

执行：

```
#####
```

```
D:\NewWorld>mvn archetype:generate
```

```
#####
```

之后选择 18，就是那个 web 应用，默认是 15，也就是我们上面那个例子默认的模式，之后是 groupId、artifactId 等：

```
#####
```

```
Define value for groupId: : com.mycom.test
```

```
Define value for artifactId: : simple-webapp
```

```
#####
```

生成的项目注意在 pom 中有两个地方不同

```
<packaging>war</packaging>
```

这个是打包方式

以及

```
<build>
```

```
    <finalName>simple-webapp</finalName>
```

```
</build>
```

这个是打包名字。

## 3.2 配置 jetty——命令行启动和 Eclipse 启动

在 pom 中配置（具体查阅源代码）

```
#####

<build>

    <finalName>simple-webapp</finalName>

    <plugins>

        <plugin>

            <groupId>org.mortbay.jetty</groupId>

            <artifactId>jetty-maven-plugin</artifactId>

            <version>6.1.18</version>

        </plugin>

    </plugins>

</build>
```

```
#####

#####

mvn install
```

```
#####

因为 jetty 默认端口为 8080，通常这个端口会被占用，所以这样启动

#####

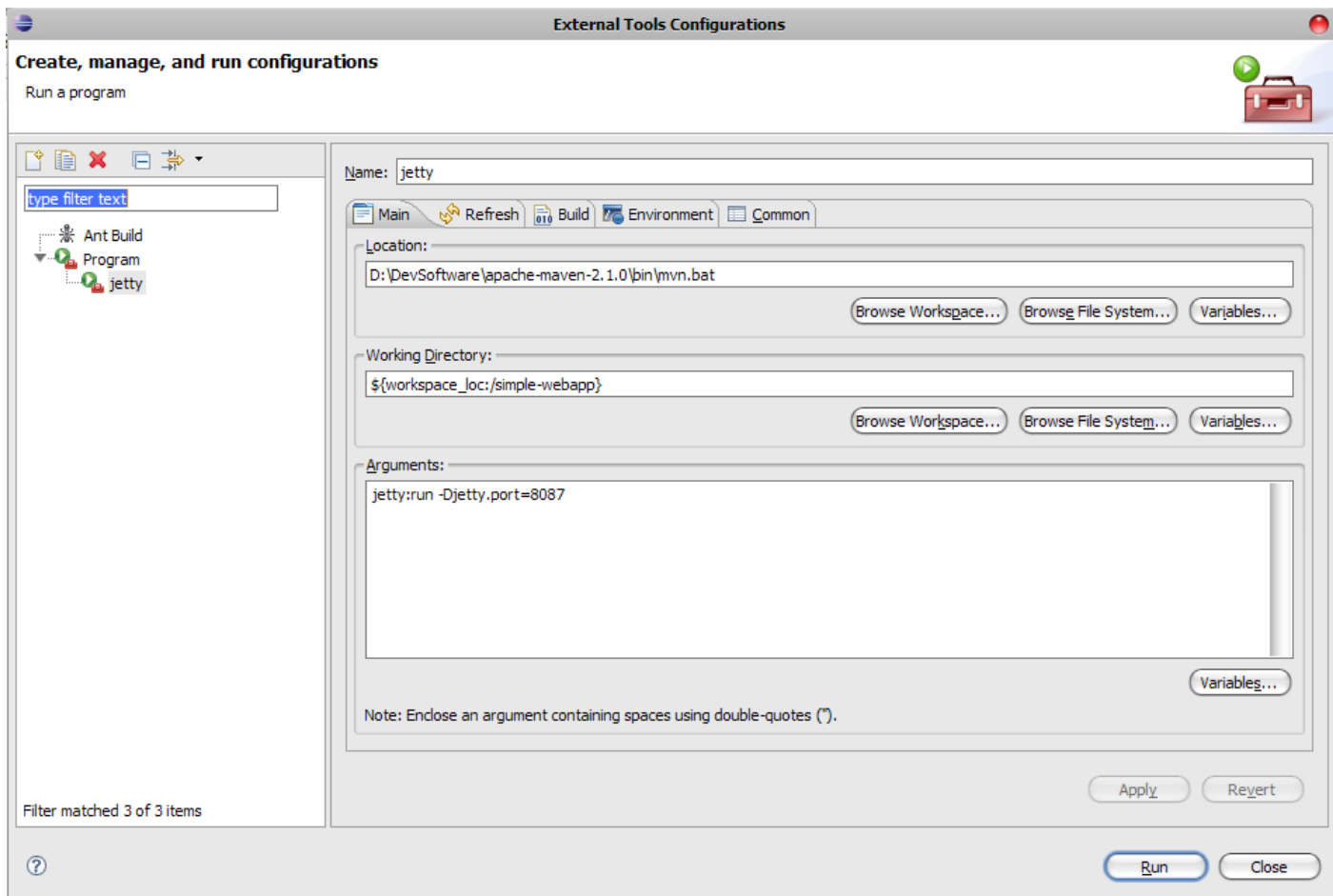
mvn -Djetty.port=8087 jetty:run
```


```
#####

用 Eclipse，导入的是 6.1.18 版本的 Maven-jetty-plugin。
```

用 Eclipse 启动则可以通过以下方法方便启动：**【run】 - 【external tools】 -**

【external tools configuration】弹出以下界面：



在左边菜单【program】右击【new】，出现一个新配置项目，点击。之后在右边的配置界面中，name 部分为改写配置名字（现在已经改为 jetty），在 location 中选择当前 Maven 的【mvn.bat】安装路径。在 working directory 中选择当前项目。在下面的 argument 中输入 mvn 的命令代码，注意里面不需要再添加“mvn”。最后点击 apply，然后 run 即可。之后在  这里便是快捷方式。

配置好之后，mvn install ，然后启动 jetty 服务器，在浏览器中便可以浏览那个 hello world 了。

## 3.3 添加一个 **servlet**——添加 J2EE 依赖

### 3.3.1 添加 **servlet2.5** 规格说明作为依赖

这里我是使用了 Eclipse 来导入的，版本与《指南》不一致。

```
#####  
<dependency>  
  
    <groupId>org.apache.geronimo.specs</groupId>  
  
    <artifactId>geronimo-servlet_2.5_spec</artifactId>  
  
    <version>1.2</version>  
  
    <type>jar</type>  
  
    <scope>compile</scope>  
  
</dependency>
```

```
#####
```

注意，用 Eclipse 可以使用相同的方法加入索引，但是 Maven 下载不一定要使用 Nexus 镜像，也就是说，Eclipse 里面可以配置 Nexus 的索引，但是 Maven 的用户级配置文件里面不需要配置镜像，这样，下载的时候还是会直接在默认官网上下载，因为我碰到过一种情况，通过 Nexus 下载的包出现问题。同时下载东西，最好还是用 Eclipse 下载，Eclipse 能够检测出下载是否完整，如果出错会自动重新下载的。

### 3.3.2 添加 **jsp2.0** 规格说明作为依赖

```
#####
```

```
<dependency>  
  
    <groupId>org.apache.geronimo.specs</groupId>
```

```

    <artifactId>geronimo-jsp_2.0_spec</artifactId>

    <version>1.1</version>

    <type>jar</type>

    <scope>compile</scope>

</dependency>

```

```
#####
```

### 3.3.3 添加一个 servlet

新建一个 source folder **【src/main/java】**, 新建一个 package

**【com.mycom.test.web】**, 从这里可以看到这个与 java app 有点不一样, java app 会自动根据 groupId 生成一个 package, 而 web app 是没有这个的。

复制源代码的 **【SimpleServlet.java】** 到包中。

### 3.3.4 配置 web.xml

在 **【web.xml】** 中配置对应的为 servlet。

```
#####
```

```

<servlet>

    <servlet-name>simple</servlet-name>


<servlet-class>org.sonatype.mavenbook.web.SimpleServlet</servlet-class>

</servlet>

<servlet-mapping>

```



```
<servlet-name>simple</servlet-name>
```

```
<url-pattern>/simple</url-pattern>
```

```
</servlet-mapping>
```

```
#####
```

### 3.3.5 运行

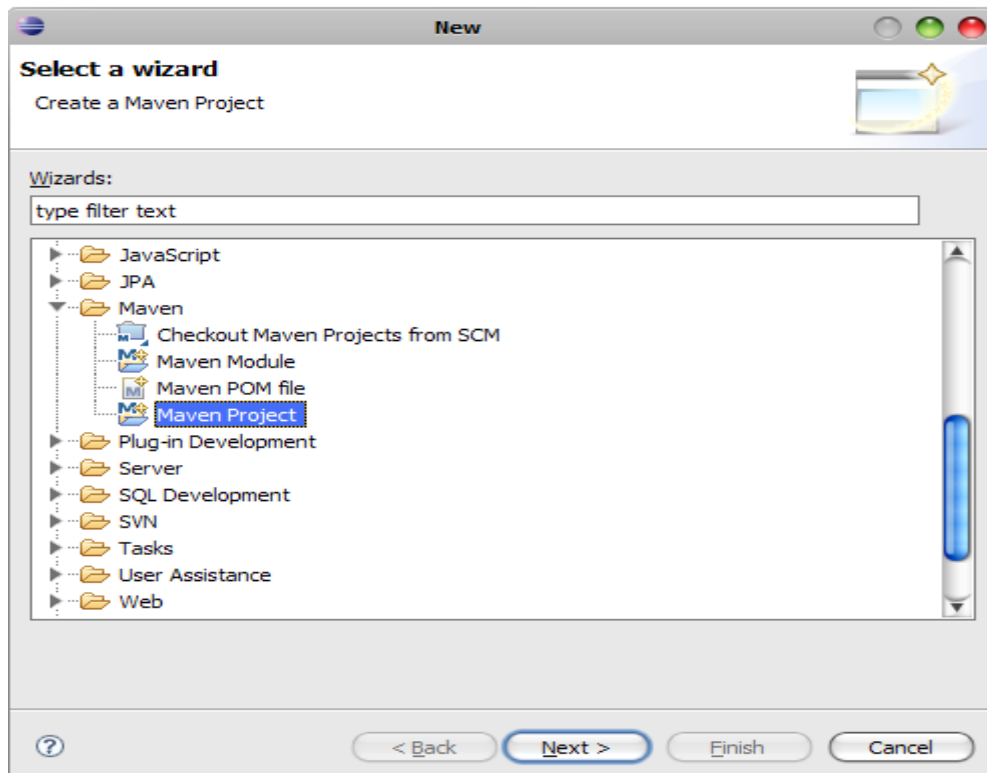
执行 `mvn clean install`

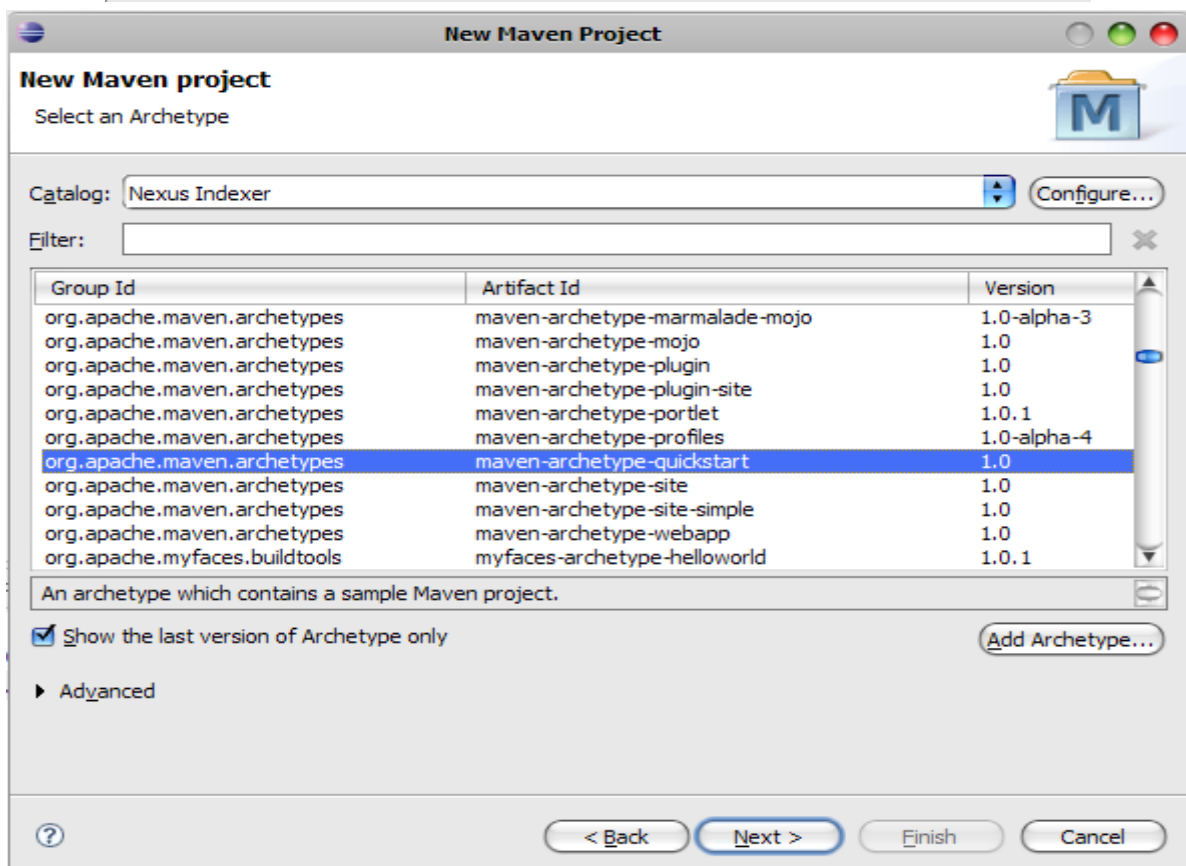
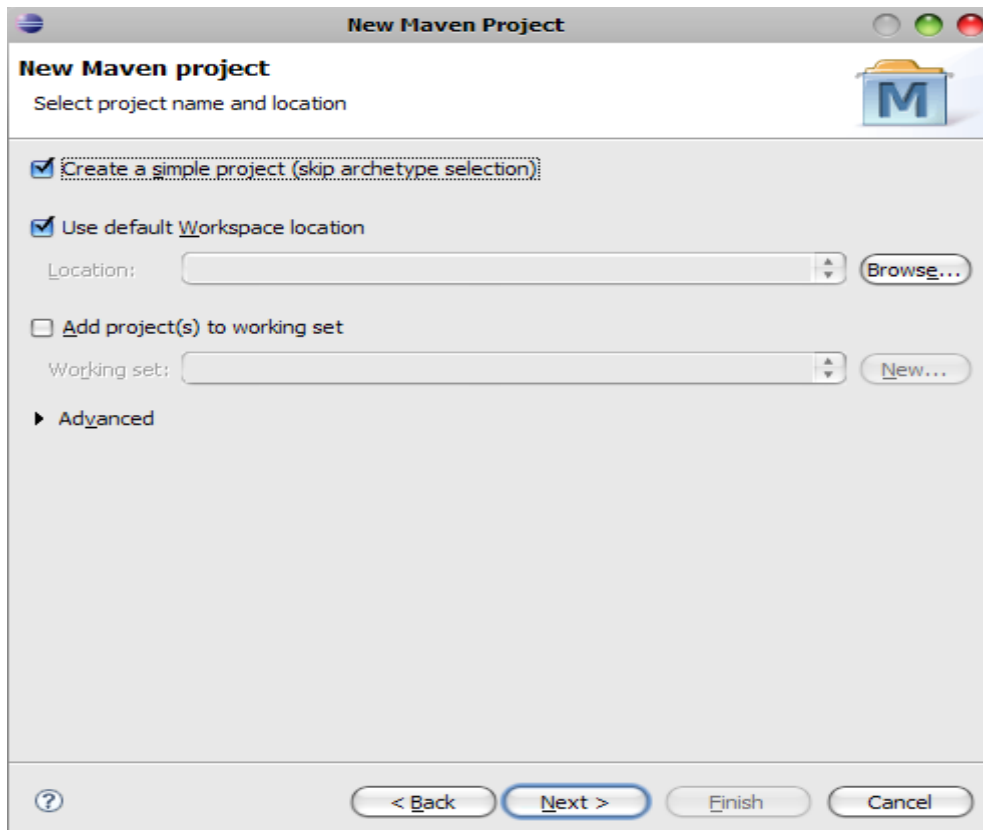
启动服务器（这里如上面的做法）

浏览器地址：<http://127.0.0.1:8087/simple-webapp/simple>

## 四、Maven——web 应用进阶——简单多模块项目

### 4.1 创建项目





注意这个分类 catalog，如果没有配 Nexus，可以选择 Internet。

**New Maven Project**

Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

Properties:

Name	Value

► Advanced

**New Maven Project**

Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

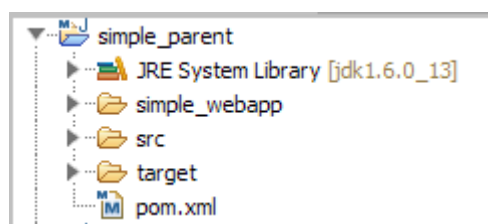
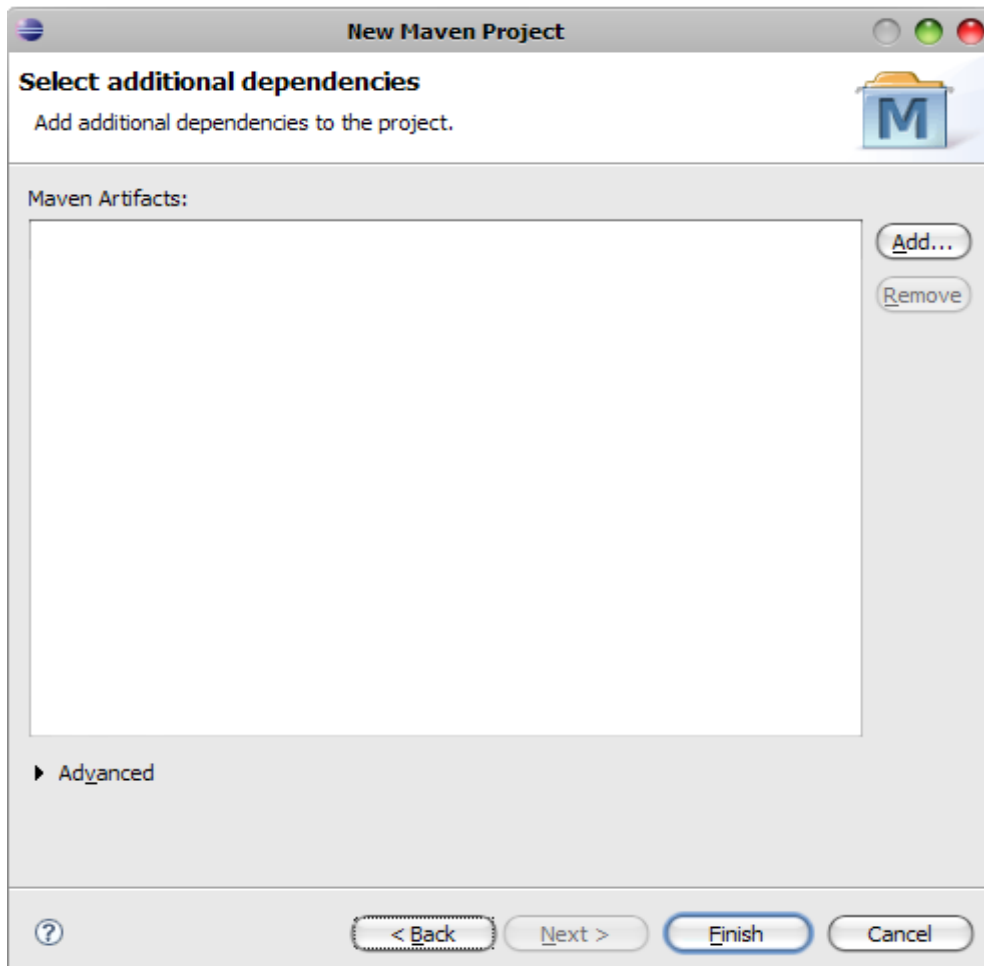
Parent Project

Group Id:

Artifact Id:

Version:

► Advanced



## 4.2 父级项目

修改 pom 中打包方式（有时候没有改）

#####

```
<packaging>pom</packaging>
```

#####

另外为了接近源码加了这段

#####

```
<dependencies>

  <dependency>

    <groupId>junit</groupId>

    <artifactId>junit</artifactId>

    <version>3.8.1</version>

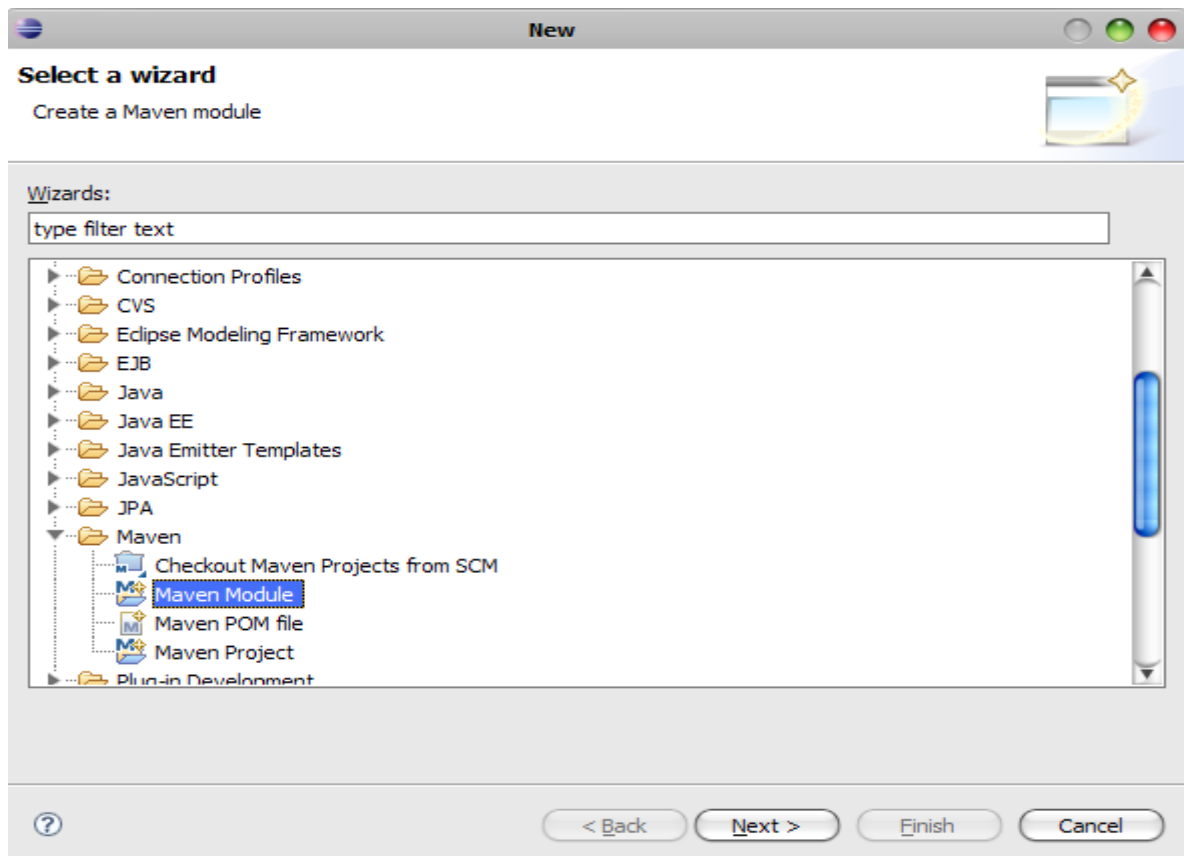
    <scope>test</scope>

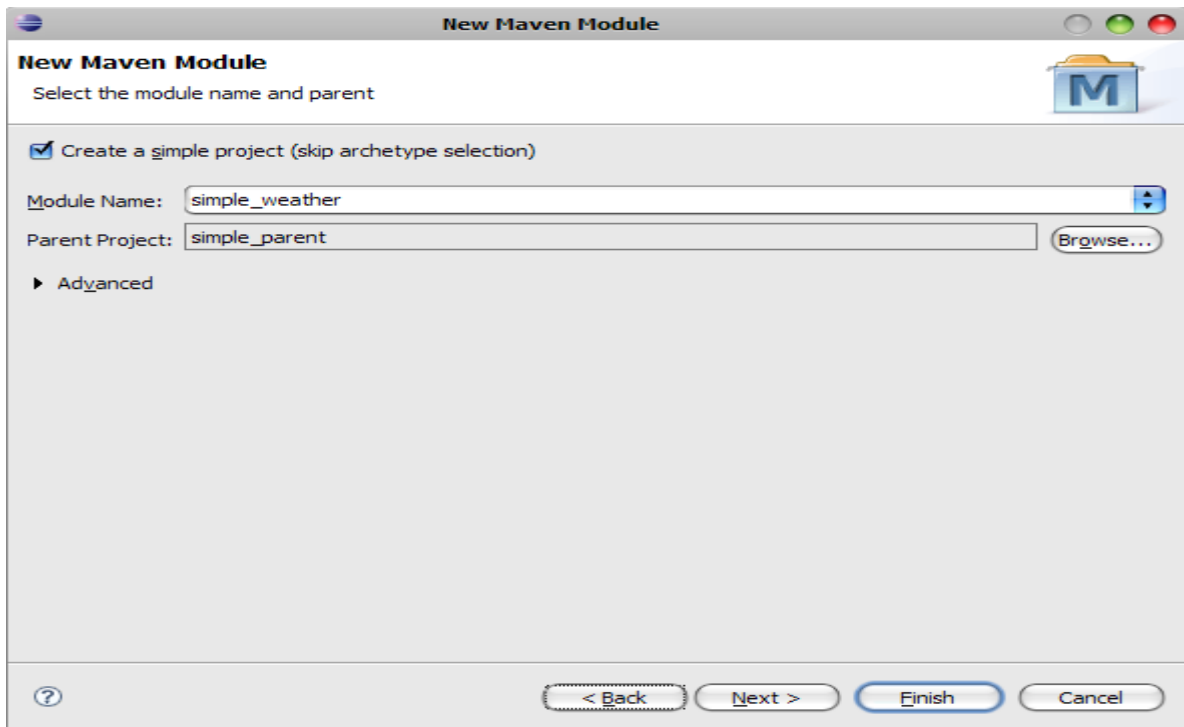
  </dependency>

</dependencies>
```

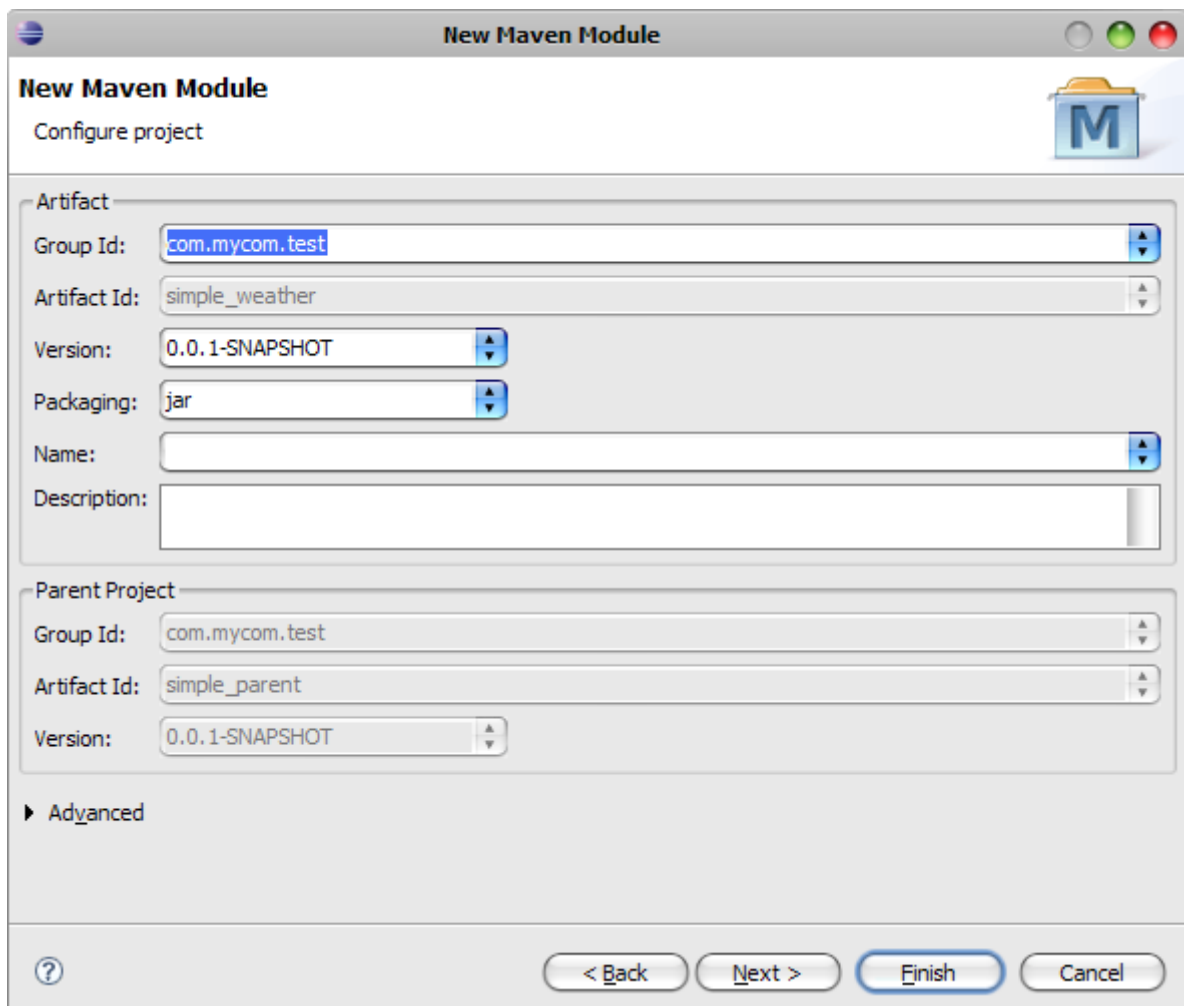
#####

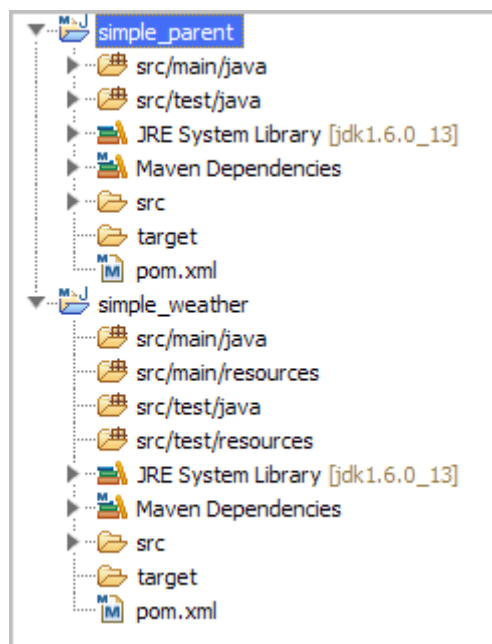
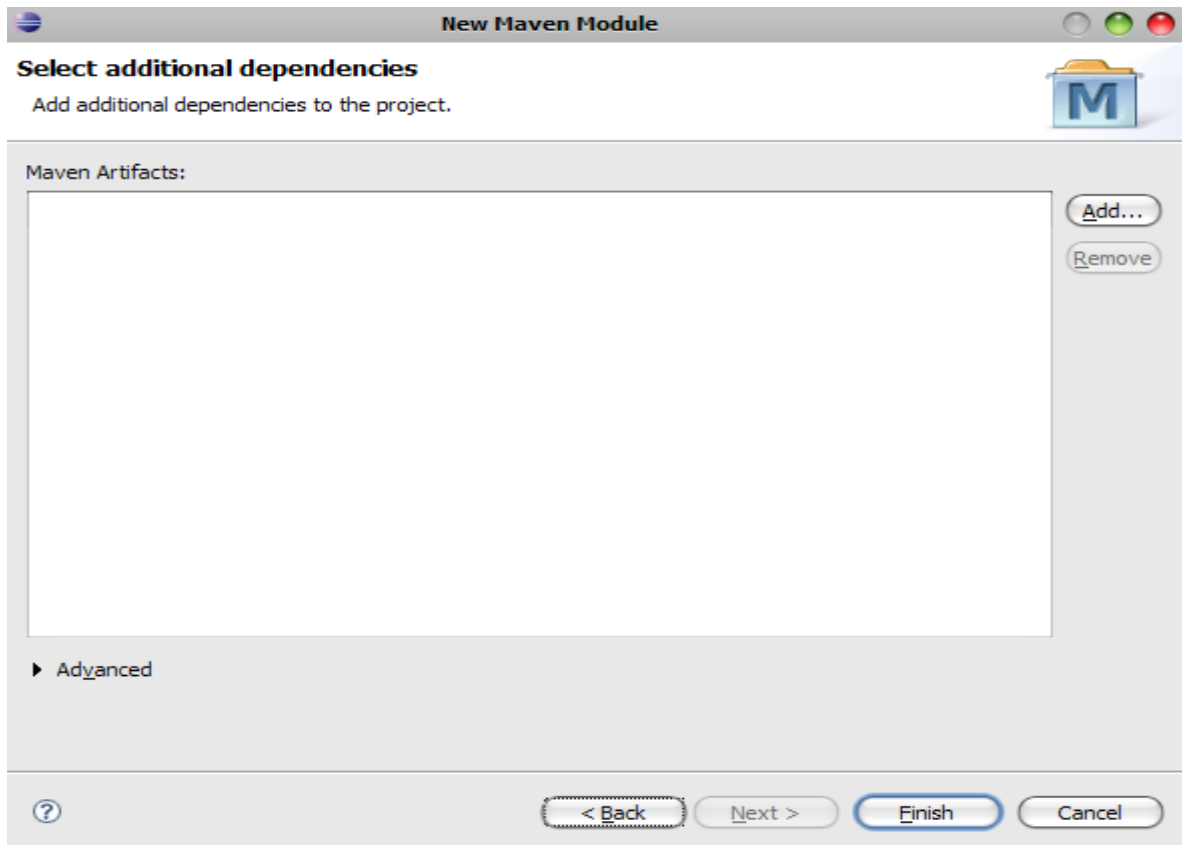
## 4.3 加入 module





这个钩钩与上面建立的 parent 的钩钩也可以勾上,这样生成的项目就是最简单的,这个看个人习惯。





在 parent 的 pom 中会自动出现

#####

<modules>

    <module>simple\_weather</module>

</modules>



#####

以同样方法再增加一个 module **【simple\_webapp】**，注意这个选择上面用到的那个 webapp 插件来生成，其他的在我的测试中有问题，生成之后那个 pom 的头有点不完整，在其他 module 中复制过来就可以了，也可以直接复制下面的。

#####

```
<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/maven-v4_0_0.xsd">
```

#####

## 4.4 simple\_weather 模块

增加 maven-surefire-plugin 插件，具体用途看《指南》，用于测试的。

#####

```
<build>
```

```
    <plugins>
```

```
        <plugin>
```

```
            <groupId>org.apache.maven.plugins</groupId>
```

```
            <artifactId>maven-surefire-plugin</artifactId>
```

```
            <version>2.4.3</version>
```

```
        </plugin>
```

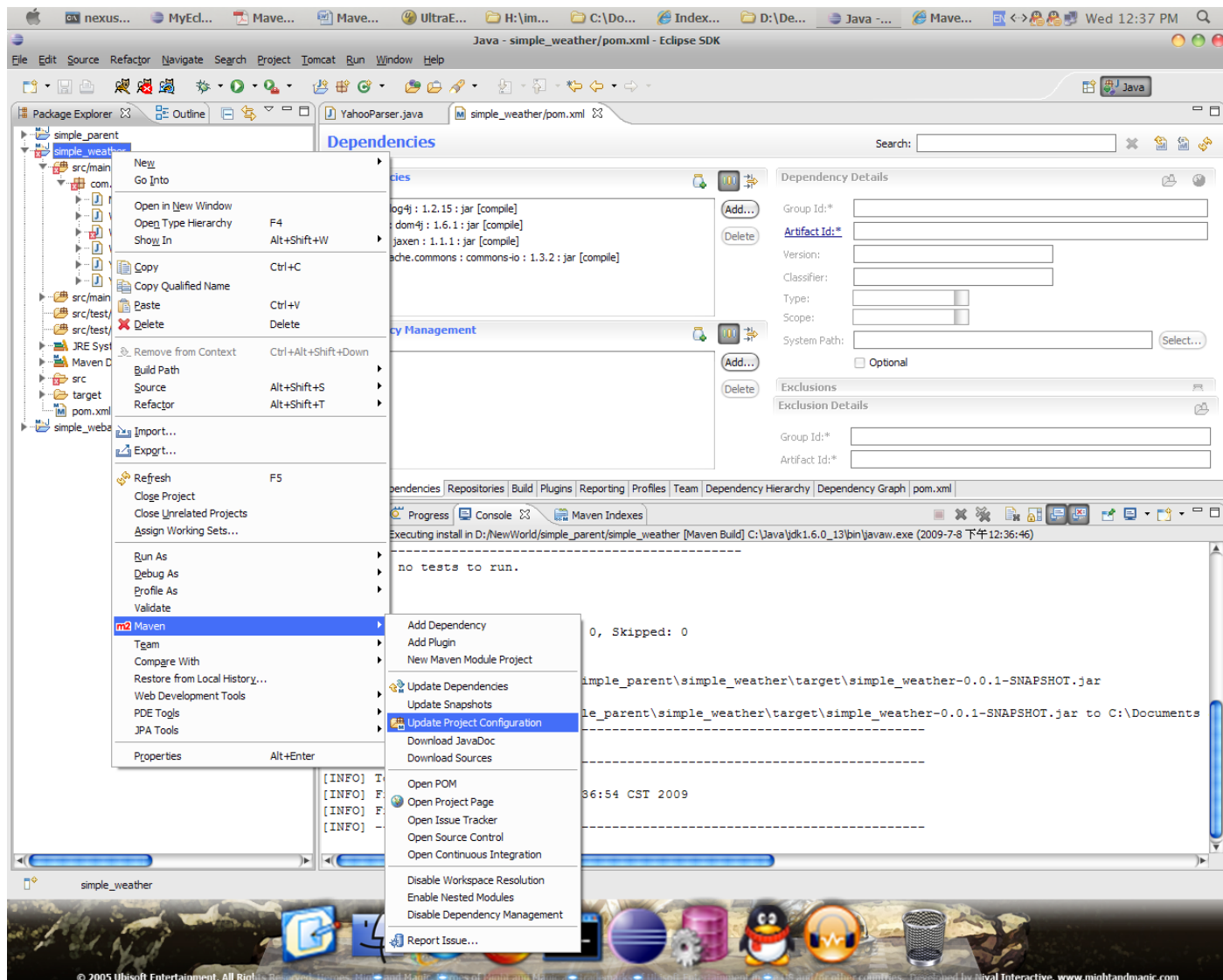
</plugins>

</build>

#####

增加依赖，和上面 simple\_weather 项目一样。

加入依赖以后点击，用于增加关联，一般保存过后会自动加入。



最后是复制源代码。

## 4.5 webapp 模块

在 pom 中加入 packaging 为 war

#####

<packaging>war</packaging>

#####

像上面的 webapp 项目加入相应的依赖和 jetty 插件。

增加一个 **servlet**，复制源码即可。

这里面有个特别注意的地方，那个 **Weather** 的 **servlet** 复制过来后，会发现有个类也就是那个 **simple\_weather** 项目 **module** 里面的类不能直接 **import**，需要把 **simple\_weather** 打包 **jar** 后像依赖一样引入即可。

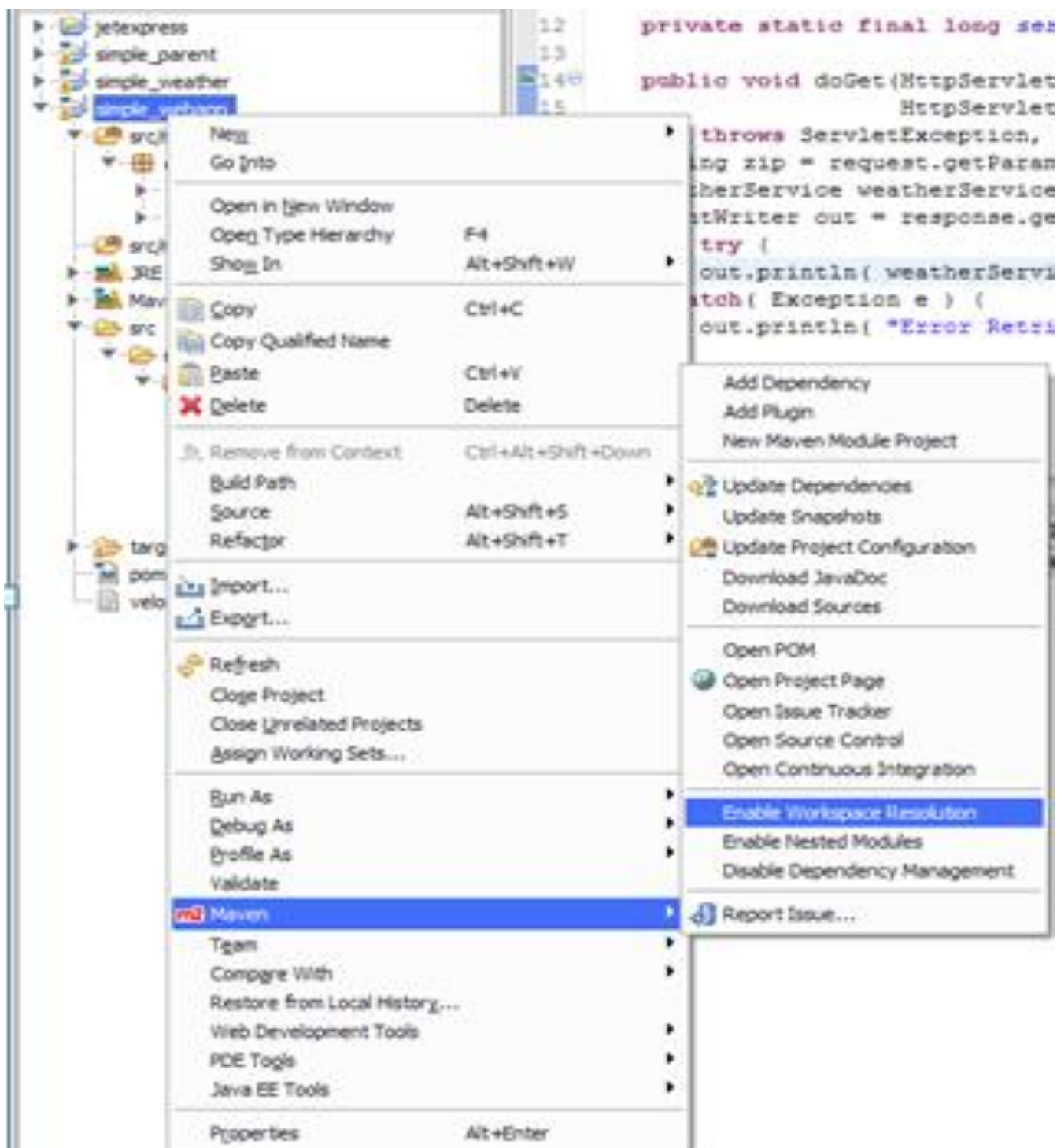
另外，不知道什么时候那个 **simple\_weather** 出现了一个警告，网上的做法只是把他去掉，但没说为什么。去掉的方法是按 “**ctrl+1**”。

引入之后，有个很奇怪的现象，就是用 Eclipse 进行 **install** 不能够成功了，说什么 “拒绝访问”。但是用命令行又可以，至今没有找到原因。

现在增加个索引。在 Nexus 中增加一个 **hosted** 仓库。地址指向本地目录。在后面增加一个自己的包，就是那个 **simple\_Weather**，包括 **pom** 和 **jar**，然后保存，这样就把这个包加入到这个仓库中。注意，记得在加入之后右击选择 “**reIndex**”。之后在 Eclipse 里面加入他的仓库地址就能加入一个新的索引了。

哇哈哈，拒绝访问的问题解决了！

在 Eclipse 里面的选项如下，



就是由于我一开始不知道这个 workspace resolution 有什么用，于是把他点上了，成为“disabled workspace resolution”，即 workspace resolution 为 enable，因此导致了 install 不成功。只要这个选择显示为“enable workspace resolution”的时候就能够 install 了。这时候，simple\_weather 的依赖也就加入到 class lib 下面。

由于 jetty 我实在不会用，因此转为使用 tomcat。在 tomcat 的 Catalina 下面加入项目部署文件。对应到相应的地方就可以了，这样就和普通的 web 项目一样了，可

以断点调试之类。注意，在更新文件之后需要 install 一次，即更新 web-inf 下面的文件，这时候 tomcat 会自动更新的。实际上和 MyEclipse 的重新发布时一样的。

有个有趣的现象，maven 有个伪热发布的效果。就是不论更新什么，服务器都不需要重启就能看到更新，但是在重启服务器后实际上还是没有变的。这个可以利用。在正式发布的时候在 install 一下就 OK。

至于下面的 nested modules 是在父级 pom 中使用的，如果是 enable 的话就是在父级项目中显示出自 module 的 source folder，反之则不显示。

## 五、多模块企业级应用

### 5.1 module 说明及所用主要技术

#### **simple-model**

该模块定义了一个简单的对象模型，对从 **Yahoo! Weather** 信息源返回的数据建模。该对象模型包含了 **Weather**, **Condition**, **Atmosphere**, **Location**, 和 **Wind** 对象。当我们的应用程序解析 **Yahoo! Weather** 信息源的时候，**simple-weather** 中定义的解析器会解析 XML 并创建供应用程序使用的 **Weather** 对象。该项目还包含了使用 **Hibernate 3** 标注符标注的模型对象，它们在 **simple-persist** 的逻辑中被用来映射每个模型对象至关系数据库中对应的表。

#### **simple-weather**

该模块包含了所有用来从 **Yahoo! Weather** 数据源获取数据并解析结果 XML 的逻辑。从数据源返回的 XML 被转换成 **simple-model** 中定义的模型对象。**simple-weather** 有一个对 **simple-model** 的依赖。**simple-weather** 定义了一个 **WeatherService** 对象，该对象会被 **simple-command** 和 **simple-webapp** 项目引用。

#### **simple-persist**

该模块包含了一些数据访问对象(DAO)，这些对象将 **Weather** 对象存储在一个内嵌数据库中。这个多模块项目中的两个应用都会使用 **simple-persist** 中定义的 **DAO** 来将数据存储至内嵌数据库中。本项目中定义的 **DAO** 能理解并返回 **simple-model** 定义的模型对象。**simple-persist** 有一个对 **simple-model** 的依赖，它也依赖于模型对象上的 **Hibernate** 标注。

#### **simple-webapp**

这个 **web** 应用项目包含了两个 **Spring MVC** 控制器实现，控制器使用了 **simple-weather** 中定义的 **WeatherService**，以及 **simple-persist** 中定义的 **DAO**。**simple-webapp** 有对于 **simple-weather** 和 **simple-persist** 的直接依赖；还有一个对于 **simple-model** 的传递性依赖。

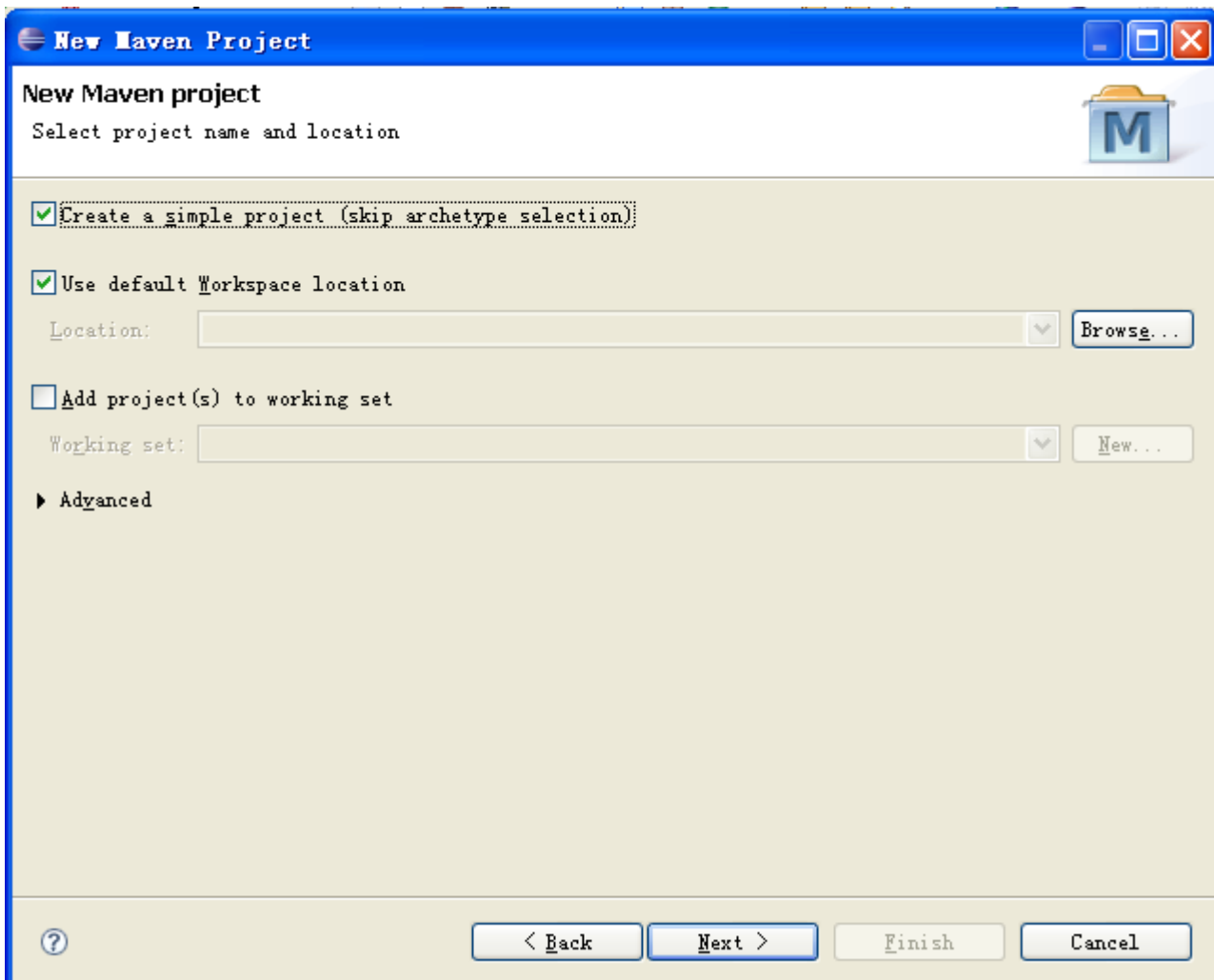
## simple-command

该模块包含了一个用来查询 Yahoo! Weather 信息源的简单命令行工具。它包含了一个带有静态 `main()` 方法的类，与 `simple-weather` 中定义的 `WeatherService` 和 `simple-persist` 中定义的 `DAO` 交互。`simple-command` 有对于 `simple-weather` 和 `simple-persist` 的直接依赖；还有一个对于 `simple-model` 的传递性依赖。

所用技术 `spring framework` 和 `hibernate`。

## 5.2 构建过程

### 5.2.1 父级项目



**New Maven Project**

New Maven project  
Configure project

**Artifact**

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

**Parent Project**

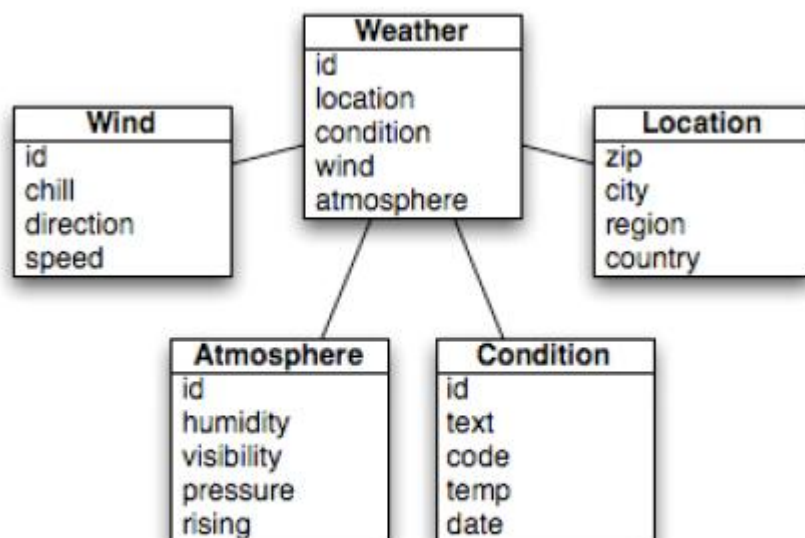
Group Id:

Artifact Id:

Version:

► Advanced

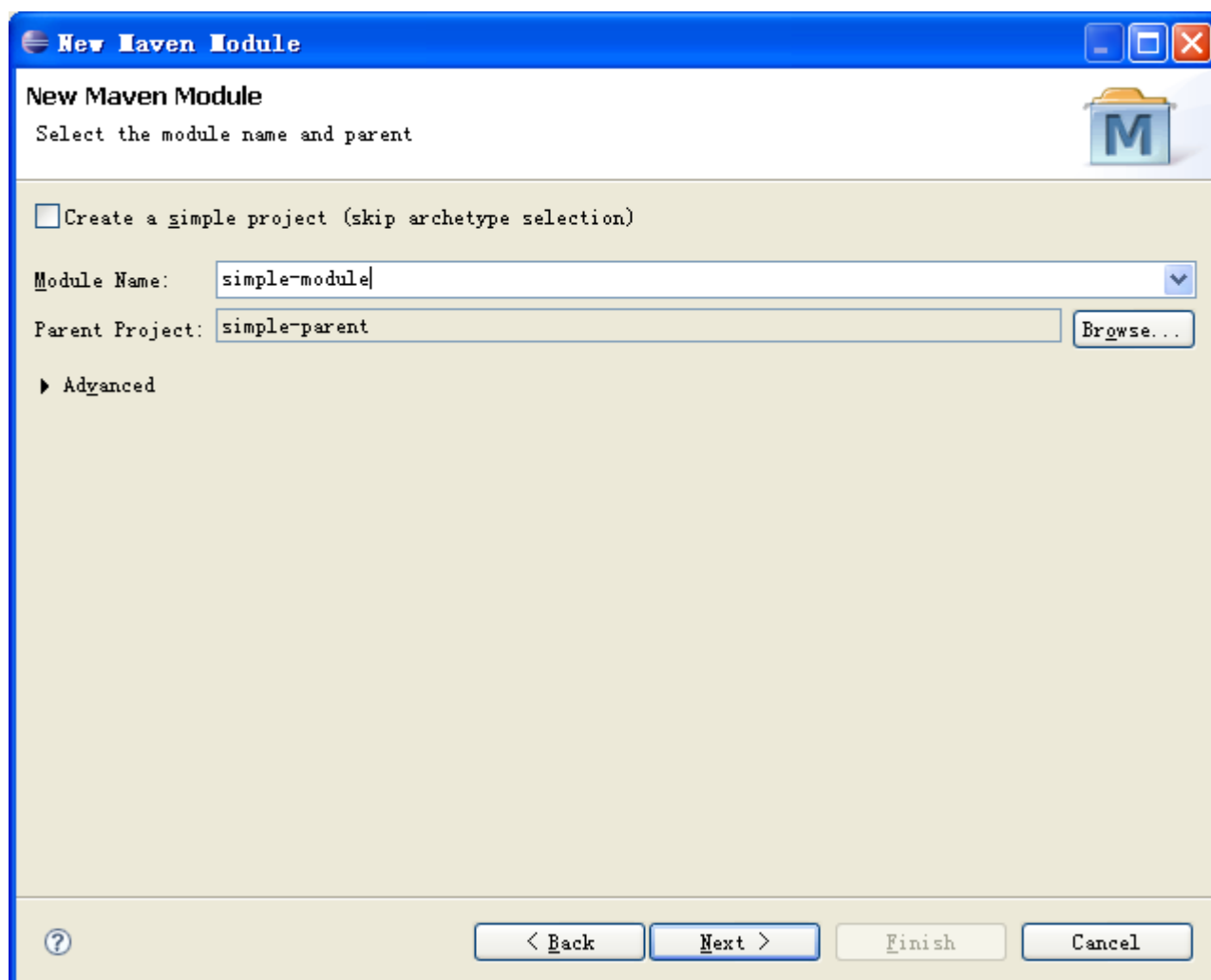
## 5.2.2 simple-module



**Figure 7.2.** 天气数据的简单对象模型



## 建立项目



The image shows a 'New Maven Module' dialog box. The title bar is blue with the text 'New Maven Module' and standard window controls. The main area has a light beige background. At the top, it says 'New Maven Module' and 'Select the module name and parent'. There is a small icon of a folder with an 'M' on it. Below this, there is a checkbox labeled 'Create a simple project (skip archetype selection)'. Underneath, there are two text input fields: 'Module Name:' with the value 'simple-module' and 'Parent Project:' with the value 'simple-parent'. To the right of the 'Parent Project' field is a 'Browse...' button. Below these fields is a collapsed section labeled 'Advanced'. At the bottom of the dialog, there is a row of buttons: a help button (question mark), '< Back', 'Next >', 'Finish', and 'Cancel'.

New Maven Module

Select the module name and parent

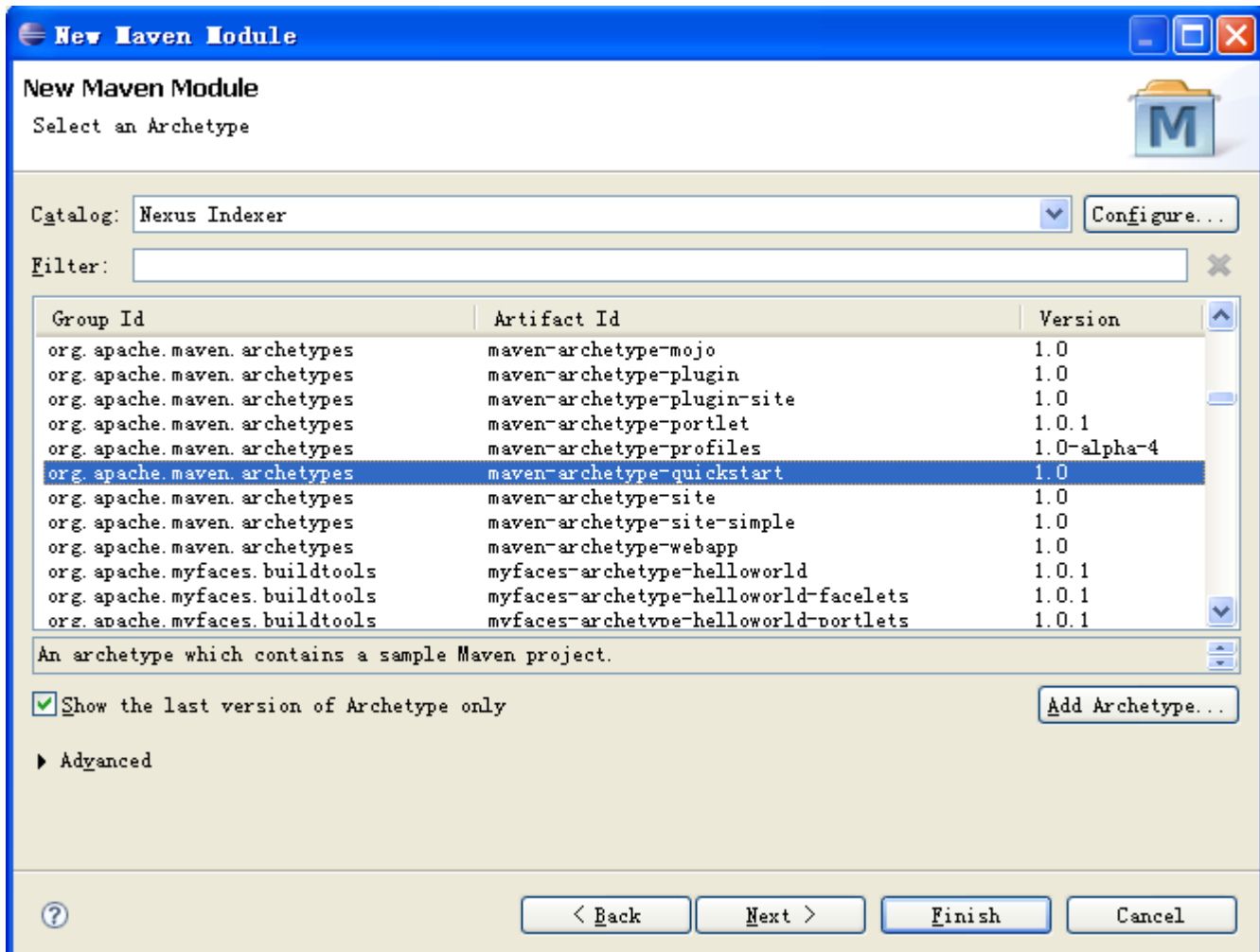
☐ Create a simple project (skip archetype selection)

Module Name: simple-module

Parent Project: simple-parent Browse...

Advanced

? < Back Next > Finish Cancel



添加 hibernate 依赖（参看源码，注意，源码中排除了一个类！按照流程，暂时不加  
上）

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-commons-annotations</artifactId>
  <version>3.3.0.ga</version>
  <type>jar</type>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-annotations</artifactId>
  <version>3.3.0.ga</version>
  <type>jar</type>
  <scope>compile</scope>
</dependency>
```



# 附录

## 1、手工安装第三方 jar

```
mvn install:install-file -DgroupId=easymock -DartifactId=easymock -Dversion=2.2 -Dpackaging=jar -Dfile=/easymock.jar  
-DgeneratePom=true
```

## 2、更新本地索引

```
mvn archetype:crawl
```

## 3、maven 默认编辑级别

maven 默认编译级别是 1.4 或者 1.3，需要在 pom 中配置。

```
<build>  
  <pluginManagement>  
    <plugins>  
      <plugin>  
        <groupId>org.apache.maven.plugins</groupId>  
        <artifactId>maven-compiler-plugin</artifactId>  
        <version>2.0.2</version>  
        <configuration>  
          <source>1.6</source>  
          <target>1.6</target>  
        </configuration>  
      </plugin>  
    </plugins>  
  </pluginManagement>  
</build>
```