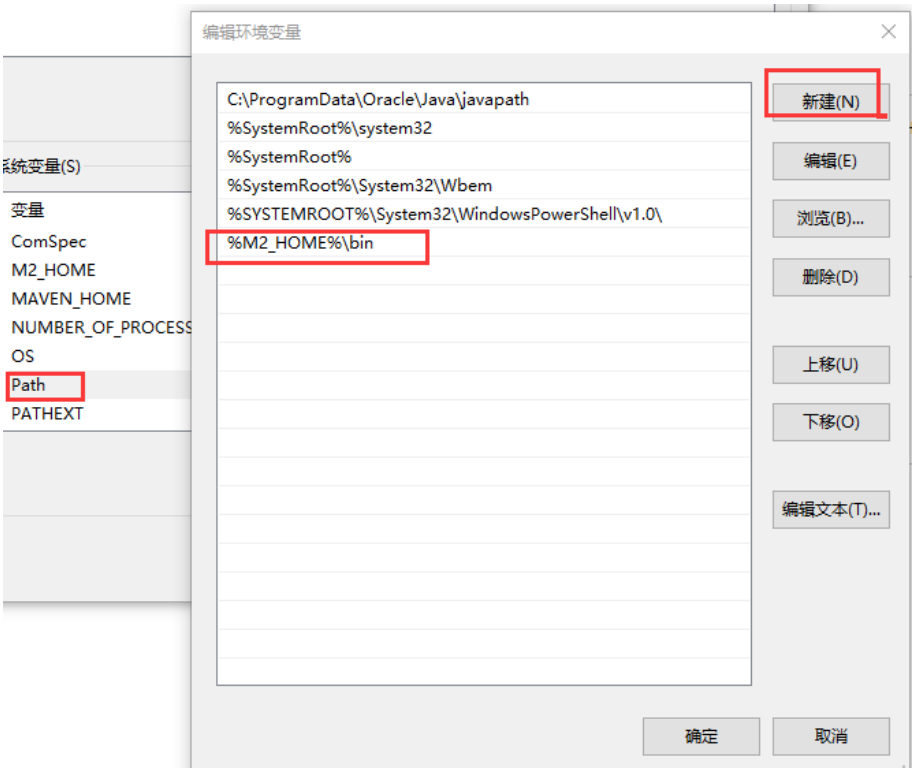
Maven_3.5.3

参考：易百教程(很乱)
制作日期：2018-6-11
制作人：小桅[yw_forgit@163.com]

1、安装配置

Windows 下载 apache-maven-3.5.3-bin.zip ，版本号自己看着办，然后解压到一个目录（相当于安装目录），eg：D:\Software\apache-maven-3.5.3 ，然后添加 Maven 环境变量，当然也要添加 JDK 的 JAVA_HOME 环境变量（但是我用的都是最新的 JDK 和 eclipse，所以不用都不用配置，尽管环境变量没有 JAVA_HOME）：

M2_HOME = D:\Software\apache-maven-3.5.3
MAVEN_HOME = D:\Software\apache-maven-3.5.3
Path = %M2_HOME%\bin // win10 以前的添加到后面去，与其他变量值之前使用分号隔开；win10 在编辑 Path 的时候，需要新建变量值了，如下图，
JAVA_HOME = D:\Program Files\Java\jdk1.8.0_161 // （等我学着学着然后就用到了，还是配置吧(使用 jdk)）



配置完，使用命令行，执行，如下图，表示成功了。

```
C:\Users\WIN>mvn -version
Apache Maven 3.5.3 (3383c37e1f9e9b3bc3df5050c29c8aff9f295297; 2018-02-25T03:49:05+08:00)
Maven home: D:\Software\apache-maven-3.5.3\bin\..
Java version: 1.8.0_161, vendor: Oracle Corporation
Java home: D:\Program Files\Java\jre1.8.0_161
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

2、启用代理访问

如果你的公司正在建立一个防火墙，并使用 HTTP 代理服务器来阻止用户直接连接到互联网。如果您使用代理，Maven 将无法下载任何依赖。为了使它工作，你必须声明在 Maven 的配置文件中设置代理服务器。
这不是必须的，要环境，我就公司就不用设置，我设置之后就无法访问了报错了，如图：

```
C:\Users\WIN>mvn archetype:generate -DgroupId=com.jluzh -DartifactId=NumberGenerator -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom
[WARNING] Failed to retrieve plugin descriptor for org.apache.maven.plugins:maven-clean-plugin:2.5: Plugin org.apache.maven.plugins:maven-clean-plugin:2.5 or one of its dependencies could not be resolved: Failed to read artifact descriptor for org.apache.maven.plugins:maven-clean-plugin:jar:2.5
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-install-plugin/2.4/maven-install-plugin-2.4.pom
[WARNING] Failed to retrieve plugin descriptor for org.apache.maven.plugins:maven-install-plugin:2.4: Plugin org.apache.maven.plugins:maven-install-plugin:2.4 or one of its dependencies could not be resolved: Failed to read artifact descriptor for org.apache.maven.plugins:maven-install-plugin:jar:2.4
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-deploy-plugin/2.7/maven-deploy-plugin-2.7.pom
[WARNING] Failed to retrieve plugin descriptor for org.apache.maven.plugins:maven-deploy-plugin:2.7: Plugin org.apache.maven.plugins:maven-deploy-plugin:2.7 or one of its dependencies could not be resolved: Failed to read artifact descriptor for org.apache.maven.plugins:maven-deploy-plugin:jar:2.7
```

```
po.maven.apache.org/maven2): proxy.host.net
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.312 s
[INFO] Finished at: 2018-06-11T10:17:18+08:00
[INFO] ------------------------------------------------------------------------
[ERROR] No plugin found for prefix 'archetype' in the current project and in the plugin groups [org.apache.maven.plugins, org.codehaus.mojo] available from the repositories [local (E:\SoftwareData\MavenLocalRepository), central (https://repo.maven.apache.org/maven2)] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/NoPluginFoundForPrefixException
```

打开配置文件 D:\Software\apache-maven-3.5.3\conf\settings.xml ，把注释的代理配置去掉注释。

```
<!-- proxies
 | This is a list of proxies which can be used on this machine to connect to the network.
 | Unless otherwise specified (by system property or command-line switch), the first proxy
 | specification in this list marked as active will be used.
 |-->
<proxies>
 <!-- proxy
  | Specification for one proxy, to be used in connecting to the network.
  |
 <proxy>
   <id>optional</id>
   <active>true</active>
   <protocol>http</protocol>
   <username>proxyuser</username>
   <password>proxypass</password>
   <host>proxy.host.net</host>
   <port>80</port>
   <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
 </proxy>
 -->
</proxies>
```

被注释掉了

```
<!-- proxies
 | This is a list of proxies which can be used on this machine to connect to the network.
 | Unless otherwise specified (by system property or command-line switch), the first proxy
 | specification in this list marked as active will be used.
 |-->
<proxies>
  <proxy>
    <id>optional</id>
    <active>true</active>
    <protocol>http</protocol>
    <username>proxyuser</username>
    <password>proxypass</password>
    <host>proxy.host.net</host>
    <port>80</port>
    <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
  </proxy>
</proxies>
```

使用代理下载maven依赖包

3、Maven 存储库

Maven 库中有三种类型

- local - 本地库
- central - 中央库
- remote - 远程库

3.1、Maven 本地资源库

Maven 的本地资源库是用来存储所有项目的依赖关系(插件 jar 和其他文件，这些文件被 Maven 下载)到本地文件夹。

更改默认的路径，打开配置文件 D:\Software\apache-maven-3.5.3\conf\settings.xml ，找到 localRepository ，添加安装格式添加新的路径即可，修改配置文件可能要重新打开 cmd。

```
<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>/path/to/local/repo</localRepository>
  -->
  <localRepository>E:\SoftwareData\MavenLocalRepository</localRepository>
```

本地资源库的位置

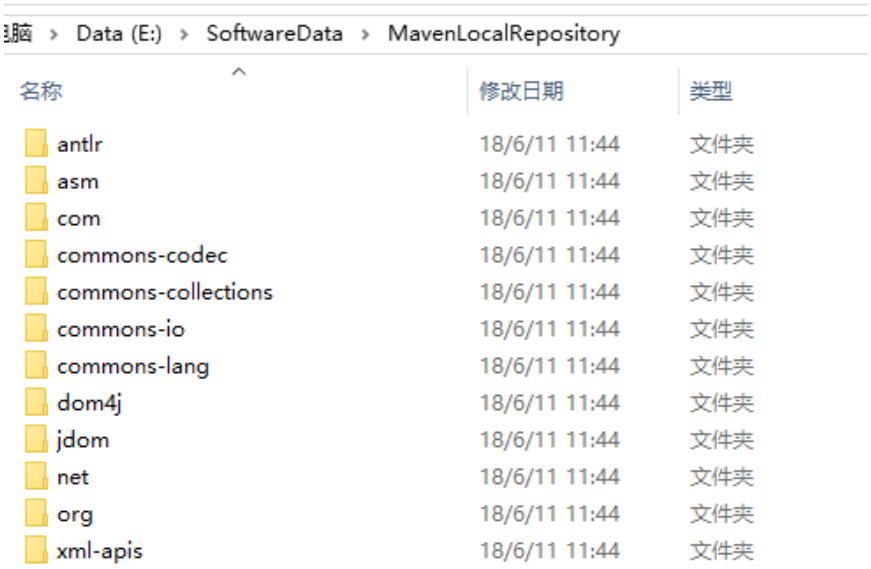
执行命令：C:\Users\WIN> mvn archetype:generate -DgroupId=com.jluzh -DartifactId=NumberGenerator -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

```
C:\Users\WIN>mvn archetype:generate -DgroupId=com.jluzh -DartifactId=NumberGenerator -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-clean-plugin/2.5/maven-clean-plugin-2.5.pom
```

然后要看到成功的信息，如图

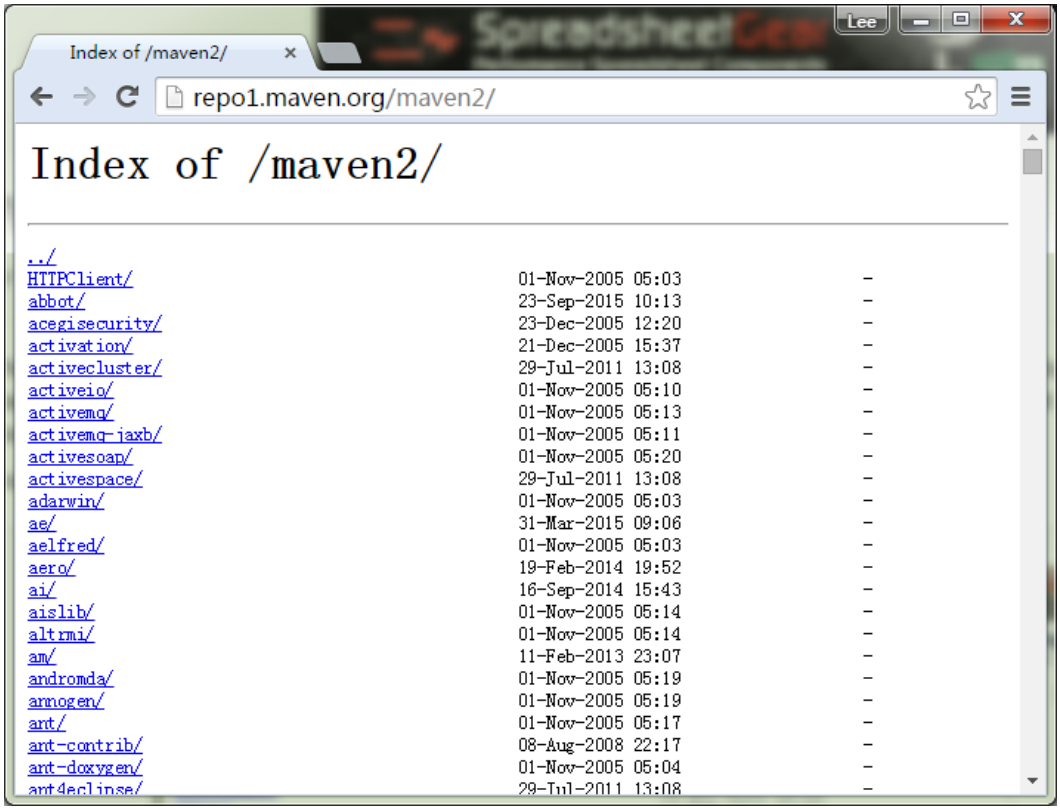
```
0/maven-archetype-quickstart-1.0.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstar
/maven-archetype-quickstart-1.0.jar (4.3 kB at 10 kB/s)
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: C:\Users\WIN
[INFO] Parameter: package, Value: com.jluzh
[INFO] Parameter: groupId, Value: com.jluzh
[INFO] Parameter: artifactId, Value: NumberGenerator
[INFO] Parameter: packageName, Value: com.jluzh
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\WIN\NumberGenerator
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 05:42 min
[INFO] Finished at: 2018-06-11T11:49:03+08:00
[INFO] ------------------------------------------------------------------------
□[0m□[0m
```

然后去 E:\SoftwareData\MavenLocalRepository 看看，就有了很多东西(com.jluzh 是我学校)。



3.2、Maven 中央存储库

当你建立一个 Maven 的项目，Maven 会检查你的 pom.xml 文件，以确定哪些依赖下载。首先，Maven 将从本地资源库获得 Maven 的本地资源库依赖资源，如果没有找到，然后把它会从默认的 Maven 中央存储库 https://repo1.maven.org/maven2/ 查找下载。
Maven 中央存储库是由 Maven 社区提供的资源库。



要浏览中央 Maven 仓库的内容，Maven 社区提供了一个网址：http://search.maven.org/ 使用这个库，开发人员可以在中央存储库中搜索所有可用的库。

3.3、远程下载

在 Maven 中，当你声明的库不存在于本地存储库中，也没有不存在于 Maven 中心储存库，该过程将停止并将错误消息输出到 Maven 控制台。

1). 示例

org.jvnet.localizer 只在于于 Java.net 资源库

pom.xml 正常配置

```
<dependency>
        <groupId>org.jvnet.localizer</groupId>
        <artifactId>localizer</artifactId>
        <version>1.8</version>
</dependency>
```

但是当你建立这个 Maven 项目，它将依赖找不到失败并输出错误消息。

2). 声明 Java.net 储存库

这时候就要告诉 Maven 来获得 Java.net 的依赖，你需要在 pom.xml 文件声明远程仓库，如下图的 URL：

pom.xml

```
 <repositories>
        <repository>
            <id>java.net</id>
            <url>https://maven.java.net/content/repositories/public/</url>
        </repository>
</repositories>
```

Maven 的依赖库查询顺序为：

1. 在 Maven 本地资源库中搜索，如果没有找到，进入第 2 步。
2. 在 Maven 中央存储库搜索，如果没有找到，进入第 3 步，若没有第 3 步，则提示错误信息，结束。
3. 在 java.net Maven 的远程存储库(url)搜索，如果没有找到，提示错误信息，结束。

JBoss Maven 存储库

添加 JBoss 远程仓库。

pom.xml

```
<project ...>
    <repositories>
      <repository>
        <id>JBoss repository</id>
        <url>http://repository.jboss.org/nexus/content/groups/public/</url>
      </repository>
    </repositories>
```

```
</project>
```

4、Maven 以来机制

在 Maven 依赖机制的帮助下自动下载所有必需的依赖库，并保持版本升级。

案例分析

让我们看一个案例研究，以了解它是如何工作的。假设你想使用 Log4j 作为项目的日志。

1.在传统方式

- 访问 http://logging.apache.org/log4j/
- 下载 Log4 j 的 jar 库
- 复制 jar 到项目类路径
- 手动将其包含到项目的依赖
- 所有的管理需要一切由自己做

如果有 Log4j 版本升级，则需要重复上述步骤一次。

2. 在 Maven 的方式

- 你需要知道 log4j 的 Maven 坐标，例如：
- ```
  <groupId>log4j</groupId>
  ```
- ```
  <artifactId>log4j</artifactId>
  ```

  ```
  <version>1.2.14</version>
  ```
- 它会自动下载 log4j 的 1.2.14 版本库。如果“version”标签被忽略，它会在自动升级库时升级 log4j 为最新的版本。

- 声明 Maven 的坐标转换成 pom.xml 文件。

```
<dependencies>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.14</version>
    </dependency>
</dependencies>
```

- 当 Maven 编译或构建，log4j 的 jar 会自动下载，并把它放到 Maven 本地存储库

5、定制库安装到 Maven 本地资源库

2 个案例(有很多 jar 不支持 Maven 的)，需要手动发出 Maven 命令包括一个 jar 到 Maven 的本地资源库。

1. 要使用的 jar 不存在于 Maven 的中心储存库中。
2. 创建了一个自定义的 jar ，而另一个 Maven 项目需要使用。

案例学习

例如，kaptcha，它是一个流行的第三方 Java 库，但它不在 Maven 的中央仓库中。

1. mvn 安装

下载 “kaptcha” (此库已经不在 com.google.code 了，但是不影响我们演示，示例时随便重命名一个文件 kaptcha-2.3.jar)，将其解压缩并将 kaptcha-xx.xx.jar 复制到一个地方，比如：C 盘。命令：

```
// 注意 xx.xx 是具体的版本号，看着来哈

mvn install:install-file -Dfile=e:\kaptcha-xx.xx.jar -DgroupId=com.google.code -DartifactId=kaptcha -Dversion=xx.xx -Dpackaging=jar
```

示例：

```
C:\Users\WIN>mvn install:install-file -Dfile=e:\kaptcha-2.3.jar -DgroupId=com.google.code -DartifactId=kaptcha -Dv
=2.3 -Dpackaging=jar
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] --- maven-install-plugin:2.4:install-file (default-cli) @ standalone-pom ---
```

```
// 看到此目录就知道上面执行的命令的所对应的参数了
```

```
[INFO] Installing e:\kaptcha-2.3.jar to E:\SoftwareData\MavenLocalRepository\com\google\code\kaptcha\2.3\kaptcha-2.3.jar
[INFO] Installing C:\Users\WIN\AppData\Local\Temp\mvninstall1689871421461981 3618.pom to E:\SoftwareData\MavenLocalReposit
ory\com\google\code\kaptcha\2.3\kaptcha-2.3.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 32.109 s
[INFO] Finished at: 2018-06-11T11:52:21+08:00
[INFO] ------------------------------------------------------------------------
```

现在，“kaptcha” jar 被复制到 Maven 本地存储库，可以去对应的目录看看。

记得删掉，因为我是随便找一个文件替代的。

2. pom.xml

安装完毕后，就在 pom.xml 中声明 kaptcha 的坐标。

```
<dependency>
      <groupId>com.google.code</groupId>
      <artifactId>kaptcha</artifactId>
      <version>2.3</version>
 </dependency>
```

构建它，现在 “kaptcha” jar 能够从你的 Maven 本地存储库检索了。

6、使用 Maven 创建 Java 项目

如何使用 Maven 来创建一个 Java 项目，并导入到 Eclipse IDE，并打包 Java 项目到一个 JAR 文件。

1. 从 Maven 模板创建一个项目

Maven 使用 **archetype** 来创建项目。要创建一个简单的 Java 应用程序，我们使用 maven-archetype-quickstart 插件。

在命令行中，浏览到要创建 Java 项目的文件夹。键入以下命令：

```
// project-packaging 是域名类型的包名；project-name 是项目名字；使用 maven-archetype-quickstart 插件来创建
mvn archetype:generate -DgroupId={project-packaging} -DartifactId={project-name}-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

这告诉 Maven ，从 maven-archetype-quickstart 模板创建 Java 项目。如果忽视 archetypeArtifactId 选项，一个巨大的 Maven 模板列表将列出。

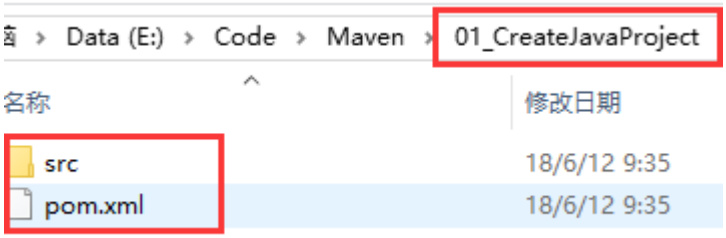
例如，工作目录是：E:\Code\Maven。

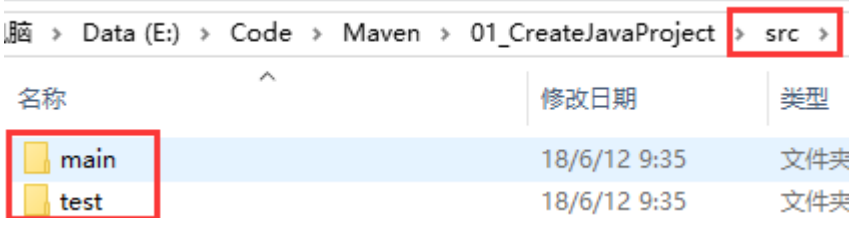
```
E:\Code\Maven>mvn archetype:generate -DgroupId=com.jluzh -DartifactId=01_CreateJavaProject -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.0.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.0.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.0.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.maven.archetypes:maven-archetype-quickstart:1.0)
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: E:\Code\Maven
[INFO] Parameter: package, Value: com.jluzh
[INFO] Parameter: groupId, Value: com.jluzh
[INFO] Parameter: artifactId, Value: 01_CreateJavaProject
[INFO] Parameter: packageName, Value: com.jluzh
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: E:\Code\Maven\01_CreateJavaProject
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

在上述情况下，一个新的 Java 项目命名 “01_CreateJavaProject”，而整个项目的目录结构会自动创建。

2. Maven 目录布局

使用 mvn archetype:generate + maven-archetype-quickstart 模板，以下项目的目录结构被创建。





很简单的，所有的源代码放在文件夹 /src/main/java/，所有的单元测试代码放入 /src/test/java/.

注意，

附加的一个标准的 pom.xml 被生成。这个 POM 文件描述了整个项目的信息。

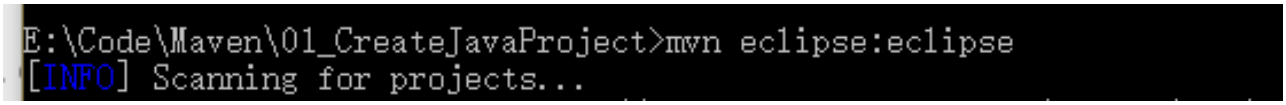
pom.xml

3. Eclipse IDE 的支持

使用 Maven 创建了一个 Java 项目，但是这个项目不能导入到 Eclipse IDE 中，因为它不是 Eclipse 风格的项目。

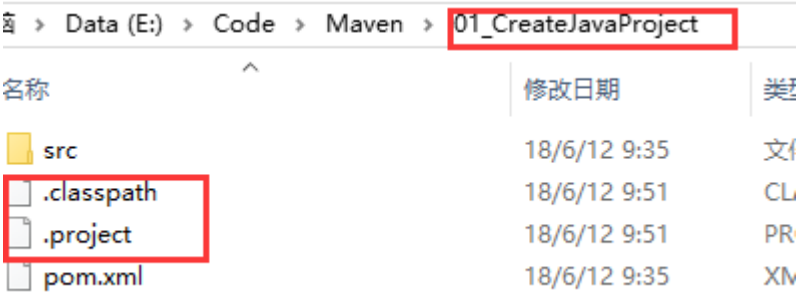
为了使它成为一个 Eclipse 项目，在终端进入到 “01_CreateJavaProject” 项目，键入以下命令：

```
mvn eclipse:eclipse
```

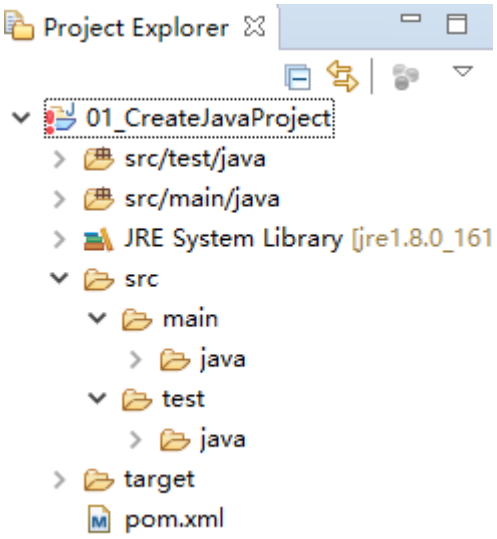


执行以上命令后，它自动下载更新相关资源和配置信息（需要等待一段时间），并产生 Eclipse IDE 所要求的所有项目文件。

你会发现创建了两个新文件 - “.classpath”和“.project”。这两个文件都为 Eclipse IDE 所创建。

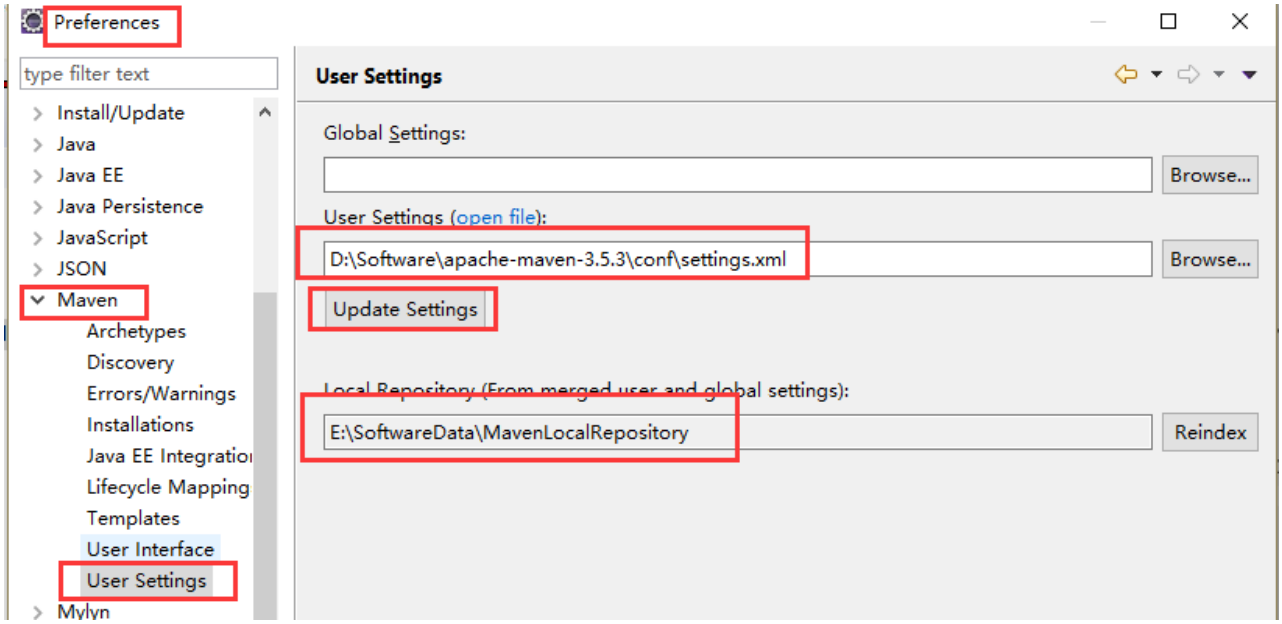


要导入项目到 Eclipse IDE 中，选择 “File -- Import… -- General -- Existing Projects into Workspace”，然后选择 root 目录为 01_CreateJavaProject，点击完成即可，



4. 更改 eclipse 中 Maven 的本地仓库

如下图，进入 Preferences，然后到 User Settings，定位到 settings.xml 文件，然后点击 Update Settings，Local Repository 就能显示 settings.xml 文件设置的本地仓库的目录了。

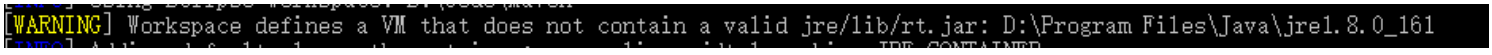


5. 更新 POM

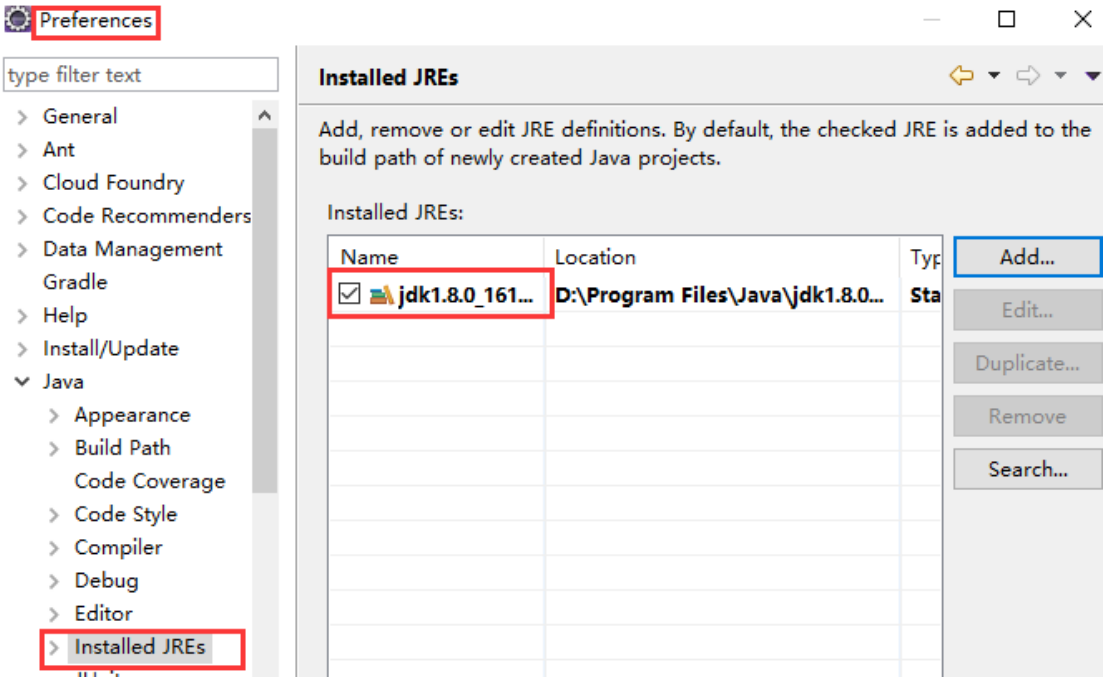
1）默认的 pom.xml 太简单了，很多时候，你需要添加编译器插件来告诉 Maven 使用哪个 JDK(包含 JRE) 版本是用来编译项目。（默认 JDK1.4，这的确太旧了点，不要使用 JRE 哦）

使用 JRE 会在你再一次执行 mvn eclipse:eclipse 时警告找不到 rt 包：

Workspace defines a VM that does not contain a valid jre/lib/rt.jar: D:\Program Files\Java\jre1.8.0_161



改 JDK：



2）从更新 JUnit 3.8.1 到最新的 4.11。

3）pom.xml

```
<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">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.jluzh</groupId>
    <artifactId>01_CreateJavaProject</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>01_CreateJavaProject</name>
    <url>http://maven.apache.org</url>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
                <configuration>
                    <!-- 指定java源码开发使用jdk和编译后的运行环境jre -->
                    <source>1.8</source>
                    <target>1.8</target>
```

```
            </configuration>
          </plugin>
        </plugins>
    </build>

</project>
```

在终端，再次执行同样的命令 mvn eclipse:eclipse ,Maven 将从 Maven 中心储存库下载插件项目依赖关系（JUnit），它会自动保存到你的本地仓库。

6. 更新业务逻辑

AppTest.java
App.java
这两个都是自动生成的代码，可以删除，逻辑代码就是在 java 目录写的。

7. Maven 打包

现在，我们将使用 Maven 这个项目，并输出编译成一个 “jar” 的文件。 请参考 pom.xml 文件，包元素定义应该包应该输出什么。

pom.xml

```
<project ...>

    <modelVersion>4.0.0</modelVersion>
  <groupId>com.jluzh</groupId>
  <artifactId>01_CreateJavaProject</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
```

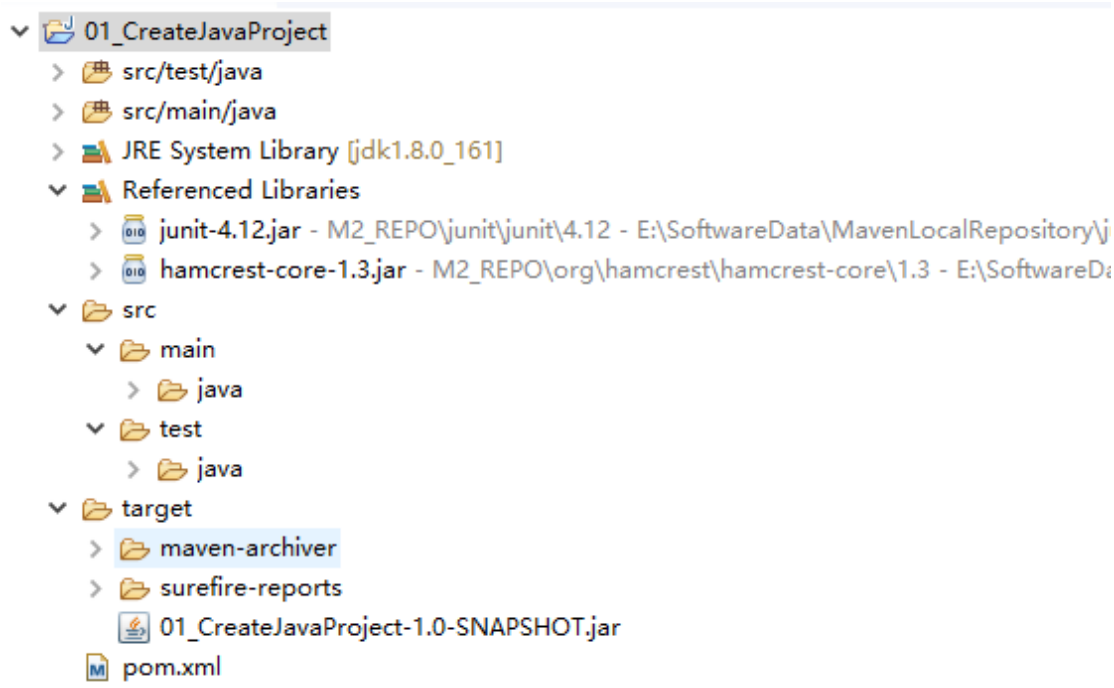
在终端输入 mvn package 或者 mvn clean package(先清理旧的，再打包)：

```
E:\Code\Maven\01_CreateJavaProject>mvn package
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< com.jluzh:01_CreateJavaProject >-------------------
[INFO] Building 01_CreateJavaProject 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ 01_CreateJavaProject ---
```

```
[INFO] Building jar: E:\Code\Maven\01_CreateJavaProject\target\01_CreateJavaProject-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

它编译，运行单元测试并打包项目成一个 jar 文件，并把它放在 project/target 文件夹。

最终项目的目录结构, 如下图片 ：



8. 示例

从项目的 jar 文件运行应用程序示例

```
E:\Code\Maven\01_CreateJavaProject>java -cp target/01_CreateJavaProject-1.0-SNAPSHOT.jar com.jluzh.App
Hello World!
```

其实就是一个 main 方法输出一个 Hello World。

7、使用 Maven 创建 Web 应用程序项目

使用 Maven 创建一个 Java Web 项目。

1. 从 Maven 模板创建 Web 项目

您可以通过使用 Maven 的 maven-archetype-webapp 模板来创建一个快速启动 Java Web 应用程序的项目。在终端或命令提示符中，导航至您想要创建项目的文件夹。

键入以下命令：

```
$ mvn archetype:generate -DgroupId=com.jluzh -DartifactId=CounterWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

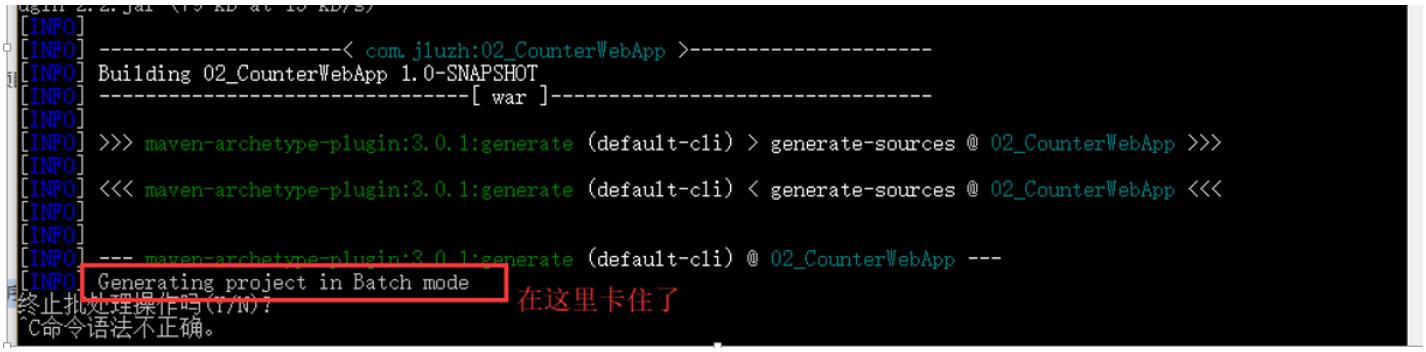
具体示例 ：

（注意要在 E:\Code\Maven 操作，不要在上一节的项目里面操作）

mvn archetype:generate -DgroupId=com.jluzh -DartifactId=02_JavaWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false

```
E:\Code\Maven>mvn archetype:generate -DgroupId=com.jluzh -DartifactId=02_JavaWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

在 Generating project in Batch mode 卡住了，Ctrl + C 可以强制退出，如下



可以在后面加一个参数-X 查看具体执行到哪一步卡住的。

添加-X 执行后你会发现，在这里卡住了，原因是需要下载 https://repo.maven.apache.org/maven2/archetype-catalog.xml 这个文件。详情如下：

解决方法就是：

1.直接从浏览器上下载该文件；

通过文件夹链接 https://repo.maven.apache.org/maven2/ 然后找到 archetype-catalog.xml（在最后面）右键保存链接进行下载（推荐！）。

2.然后复制到： 本地仓库+ \org\apache\maven\archetype\archetype-catalog\x.x.x（版本号） 下面；

3.然后在执行的命令后面加上增加参数**-DarchetypeCatalog=local**，变成读取本地文件即可。

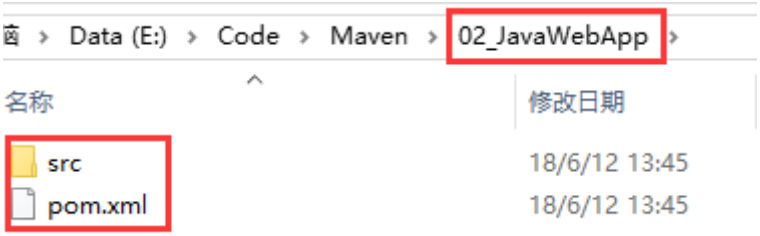
mvn archetype:generate -DgroupId=com.jluzh -DartifactId=02_JavaWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false -DarchetypeCatalog=local

```
E:\Code\Maven>mvn archetype:generate -DgroupId=com.jluzh -DartifactId=02_JavaWebApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false -DarchetypeCatalog=local
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.0.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.0.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.0.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-webapp:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: E:\Code\Maven
[INFO] Parameter: package, Value: com.jluzh
[INFO] Parameter: groupId, Value: com.jluzh
[INFO] Parameter: artifactId, Value: 02_JavaWebApp
[INFO] Parameter: packageName, Value: com.jluzh
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: E:\Code\Maven\02_JavaWebApp
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

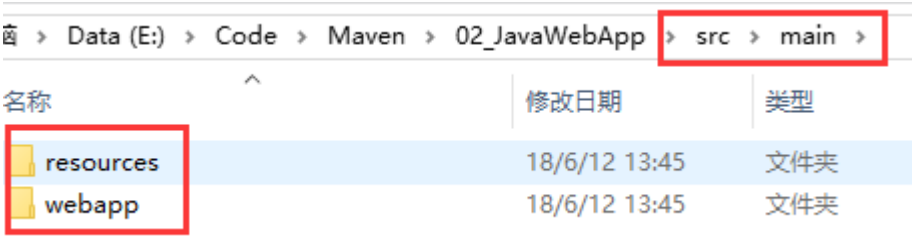
新的 Web 项目命名为 “02_JavaWebApp”，以及一些标准的 web 目录结构也会自动创建。

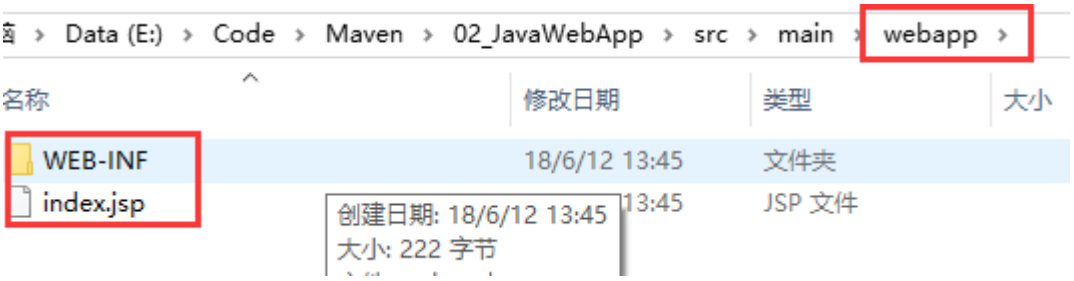
2. 项目目录布局

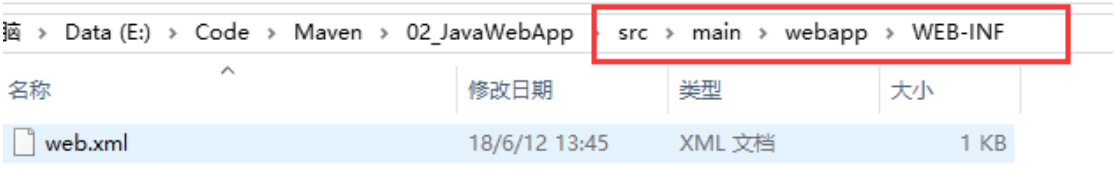
查看生成的项目结构布局：



resource 目录是空的，主要是放配置文件的，比如 Hibernate、Spring、MyBatis 的 xml 配置文件







Maven 产生了一些文件夹，一个部署描述符 web.xml，pom.xml 和 index.jsp。

pom.xml

```
<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">

  <modelVersion>4.0.0</modelVersion>
  <groupId>com.jluzh</groupId>
  <artifactId>02_JavaWebApp</artifactId>

  <!-- 不写packaging，默认打包是jar，而web项目打包是war包，故这里是war -->

  <packaging>war</packaging>

  <version>1.0-SNAPSHOT</version>
  <name>02_JavaWebApp Maven Webapp</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>02_JavaWebApp</finalName>
  </build>

</project>
```

web.xml - Servlet 2.3 已经比较旧，建议升级

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
  <display-name>Archetype Created Web Application</display-name>
</web-app>
```

index.jsp - 一个简单的 hello world html 页面文件

```
<html>
<body>
```

```
<h2>Hello World!</h2>
</body>
</html>
```

其实在这一步，就可以直接用 `mvn package` 打包了发布了，后面只是为了提供 `Eclipse` 的支持而已。

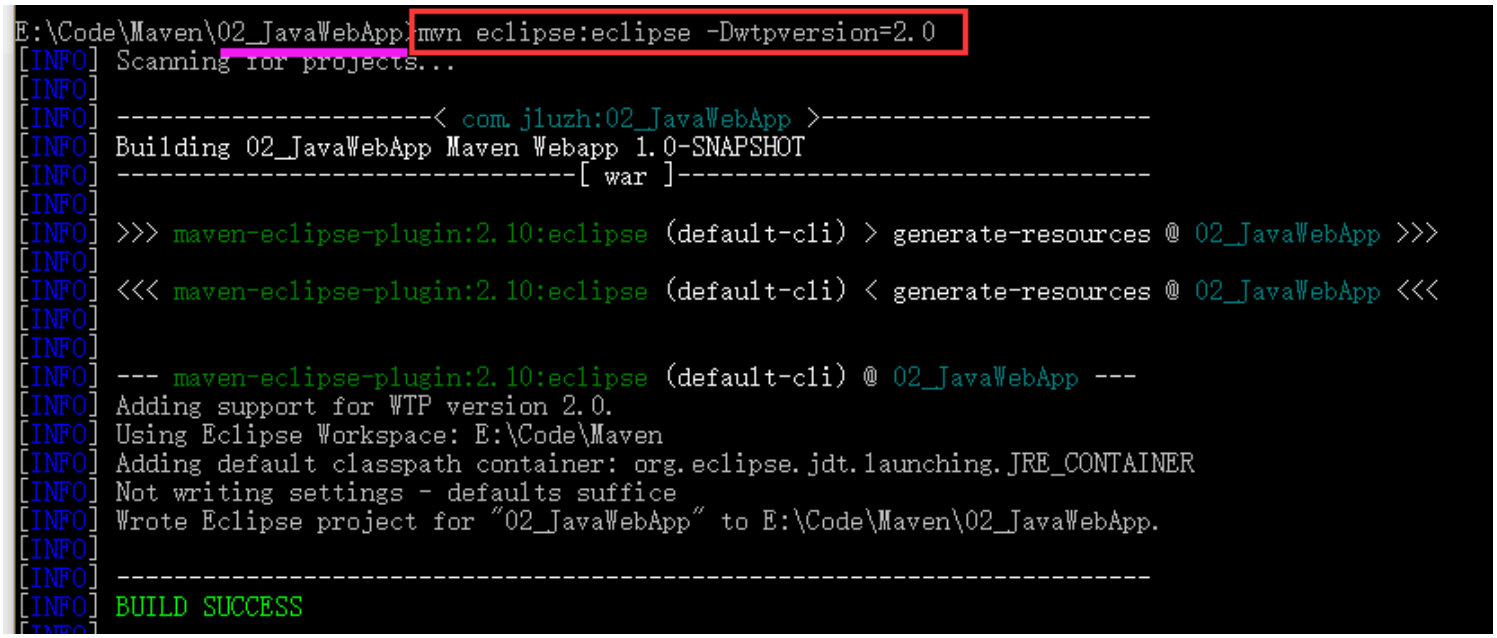
3. Eclipse IDE 的支持

注意，通过 WTP 工具 Eclipse IDE 支持 Web 应用程序，所以需要让基于 Maven 的项目支持它。

要导入这个项目到 Eclipse 中，需要生成一些 Eclipse 项目的配置文件：

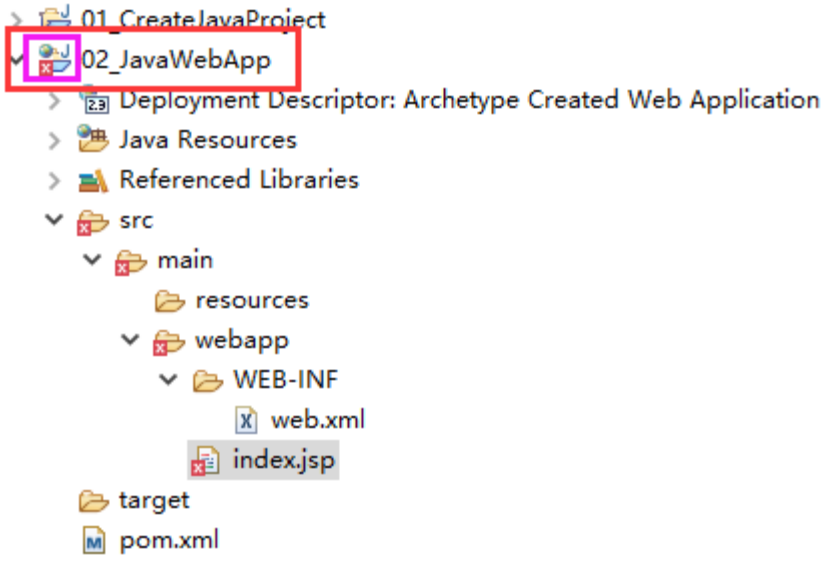
3.1、在终端，进入到 “02_JavaWebApp” 文件夹中，键入以下命令：

```
E:\Code\Maven\02_JavaWebApp>mvn eclipse:eclipse -Dwtpversion=2.0
```



（对于 Web 应用程序，需要额外的参数 -Dwtpversion=2.0 ，使其支持 Eclipse WTP）
注意，此选项 -Dwtpversion=2.0 告诉 Maven 将项目转换到 Eclipse 的 Web 项目(WAR)，而不是默认的 Java 项目(JAR)。为方便起见，以后我们会告诉你如何配置 pom.xml 中的这个 WTP 选项。

3.2 导入到 Eclipse IDE。File -- Import… -- General -- Existing Projects into workspace.



在 Eclipse 中，如果看到项目顶部有地球图标，意味着这是一个 Web 项目。

4. 更新 POM

在 Maven 中，Web 项目的设置都通过这个单一的 pom.xml 文件配置。

pom.xml

```
<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">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.jluzh</groupId>
    <artifactId>02_JavaWebApp</artifactId>
    <!-- 不写packaging，默认打包是jar，而web项目打包是war包，故这里是war -->
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>02_JavaWebApp Maven Webapp</name>
    <url>http://maven.apache.org</url>

    <!-- 属性值定义，供后面的version使用，这样可以修改方便，还一目了然 -->
    <properties>
        <jdk.version>1.8</jdk.version>
        <junit.version>4.11</junit.version>
    </properties>

    <dependencies>
        <!-- Unit Test -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <!-- scope是依赖的生命周期，设置scope为test表示此依赖只会在测试时能用，发布等时期不可用，此jar也不会被拷贝过
去 -->
            <scope>test</scope>
        </dependency>
```

```
    </dependencies>

    <build>
        <finalName>02_JavaWebApp</finalName>

        <plugins>

            <!-- Eclipse project -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-eclipse-plugin</artifactId>
                <version>2.9</version>
                <configuration>
                    <!-- 下载依赖的时候，同时也下载源码，但是javadocs文档不下载 -->
                    <downloadSources>true</downloadSources>
                    <downloadJavadocs>false</downloadJavadocs>
                    <!-- 设置wtpversion，可以避免输入  mvn eclipse:eclipse -Dwtpversion=2.0  来创建Eclipse Web项目，
可以直接输入 mvn eclipse:eclipse 即可-->
                    <wtpversion>2.0</wtpversion>
                </configuration>
            </plugin>

            <!-- Set JDK Compiler Level -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
                <configuration>
                    <source>${jdk.version}</source>
                    <target>${jdk.version}</target>
                </configuration>
            </plugin>

            <!-- maven的Tomcat插件声明 -->
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <version>2.2</version>
                <configuration>
                    <path>/02_JavaWebApp</path>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

注意，为方便起见，声明 maven-eclipse-plugin，并配置 wtpversion 来避免输入参数 -Dwtpversion=2.0。现在，执行命令 mvn eclipse:eclipse，将这个项目转换为 Eclipse Web 项目，并且会下载相关的依赖。

```
E:\Code\Maven\02_JavaWebApp>mvn eclipse:eclipse
[INFO] Scanning for projects...
```

```
rces.jar (364 kB at 13 kB/s)
[INFO] Wrote settings to E:\Code\Maven\02_JavaWebApp\.settings\org.eclipse.jdt.core.prefs
[INFO] File E:\Code\Maven\02_JavaWebApp\.project already exists.
       Additional settings will be preserved, run mvn eclipse:clean if you want old settings to be removed.
[INFO] Wrote Eclipse project for "02_JavaWebApp" to E:\Code\Maven\02_JavaWebApp.
[INFO]
       Sources for some artifacts are not available.
       List of artifacts without a source archive:
         o jstl:jstl:1.2

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

5. 更新源代码

这里不写代码

6. Eclipse + Tomcat

三种启动项目的方法：（开发使用第二种，发布部署使用第一种(一般使用 eclipse 图形界面来打包，后续说 运行 Maven 命令)）

6.1 要编译，测试和项目打包成一个 WAR 文件，输入：

```
mvn package 或者 mvn clean package(清理、打包)
```

```
E:\Code\Maven\02_JavaWebApp>mvn package
[INFO] Scanning for projects...
```

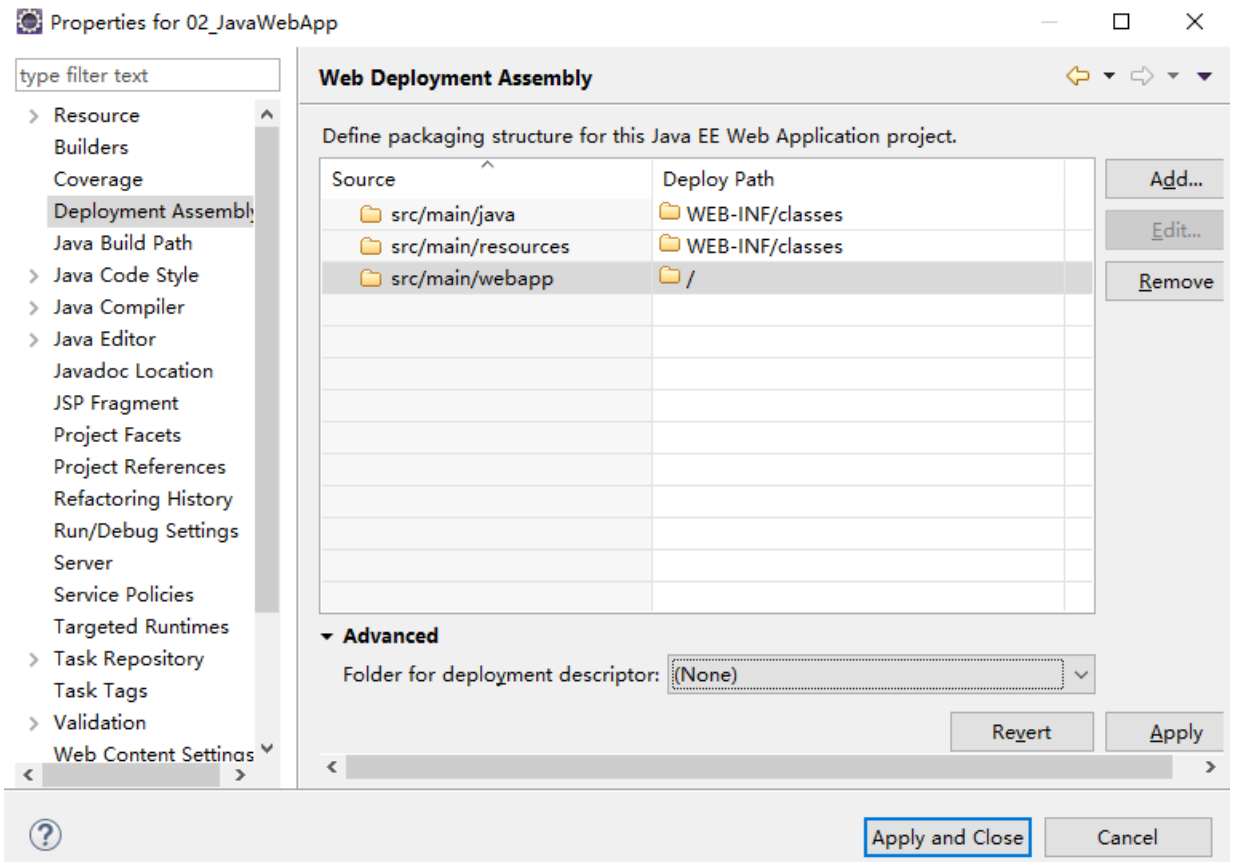
```
[INFO] Packaging webapp
[INFO] Assembling webapp [02_JavaWebApp] in [E:\Code\Maven\02_JavaWebApp\target\02_JavaWebApp]
[INFO] Processing war project
[INFO] Copying webapp resources [E:\Code\Maven\02_JavaWebApp\src\main\webapp]
[INFO] Webapp assembled in [177 msecs]
[INFO] Building war: E:\Code\Maven\02_JavaWebApp\target\02_JavaWebApp.war
[INFO] WEB-INF\web.xml already added, skipping
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

一个新的 WAR 文件将在 target/02_JavaWebApp.war 产生，只需复制并部署到 Tomcat 发布的目录。

6.2 如果想通过 Eclipse 服务器这个项目插件(Tomcat 或其它容器)调试，这里再输入：

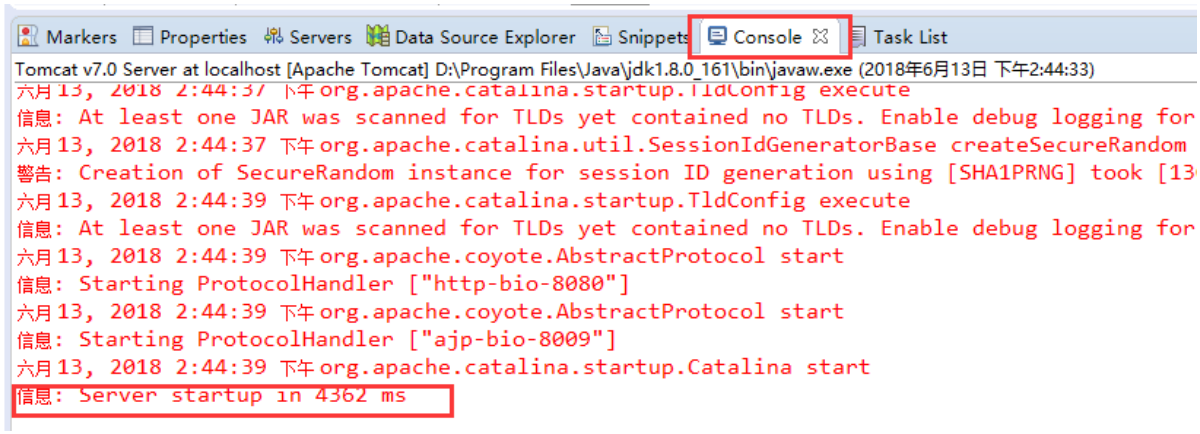
```
mvn eclipse:eclipse
```

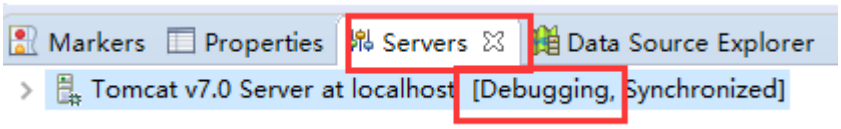
如果一切顺利，该项目的依赖将被装配附加到 Web 部署项目。



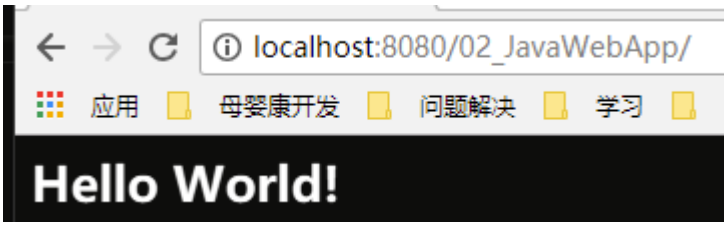
右键点击 project -> Properties -> Deployment Assembly。

然后在 eclipse 配置你的 Tomcat，然后启动它





访问



6.3 Maven 的 Tomcat 插件声明(加入到 pom.xml)：

pom.xml

```
<!-- maven的Tomcat插件声明 -->
<plugin>
    <groupId>org.apache.tomcat.maven</groupId>
    <artifactId>tomcat7-maven-plugin</artifactId>
    <version>2.2</version>
    <configuration>
        <path>/02_JavaWebApp</path>
    </configuration>
</plugin>
```

执行：

```
mvn tomcat:run
```

这将启动 Tomcat，部署项目默认在端口 8080，如果其他端口占用，需要先关闭它。

8、Maven POM

POM 代表项目对象模型。它是 Maven 中工作的基本单位，这是一个 XML 文件。它始终保存在该项目基本目录中的 `pom.xml` 文件。
POM 包含的项目是使用 Maven 来构建的，它用来包含各种配置信息。
POM 也包含了目标和插件。在执行任务或目标时，Maven 会使用当前目录中的 POM。它读取 POM 得到所需要的配置信息，然后执行目标。部分的配置可以在 POM 使用如下：

- project dependencies
- plugins
- goals
- build profiles
- project version
- developers
- mailing list

创建一个 POM 之前，应该要先决定项目组(groupId)，它的名字(artifactId)和版本，因为这些属性在项目仓库是唯一标识的。

POM 的例子

```
<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>com.jluzh.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>
<project>
```

```
XML
```

要注意的是，每个项目只有一个 POM 文件。

- 所有的 POM 文件要项目元素必须有三个必填字段: `groupId`，`artifactId`，`version`
- 在库中的项目符号是：`groupId:artifactId:version`
- pom.xml 的根元素是 `project`，它有三个主要的子节点。

节点	描述
groupId	这是项目组的编号，这在组织或项目中通常是独一无二的。 例如，一家银行集团 `com.company.bank` 拥有所有银行相关项目。
artifactId	这是项目的 ID。这通常是项目的名称。 例如，`consumer-banking`。 除了 `groupId` 之外，`artifactId` 还定义了 `artifact` 在存储库中的位置。
`version`	这是项目的版本。与 `groupId` 一起使用，`artifact` 在存储库中用于将版本彼此分离。 例如：`com.company.bank:consumer-banking:1.0`

超级 POM

所有的 POM 继承自父类(尽管明确界定)。这个基础的 POM 被称为超级 POM，并包含继承默认值。 Maven 使用有效的 POM(超级 POM 加项目配置的配置)执行有关目标。它可以帮助开发人员指定最低配置的详细信息写在 `pom.xml` 中。虽然配置可以很容易被覆盖。 一个简单的方法来看看超级 POM 的默认配置，通过运行下面的命令：`mvn help:effective-pom` 。

进入带 pom.xml 的项目目录，执行以下 `mvn` 命令：

```
E:\Code\Maven\02_JavaWebApp>mvn help:effective-pom
```

显示结果：有效 POM，继承，插值，应用配置文件。

```
<?xml version="1.0" encoding="GBK"?>
<!-- ====================================================================== -->
<!--                                                                        -->
<!-- Generated by Maven Help Plugin on 2018-06-13T14:16:02+08:00            -->
<!-- See: http://maven.apache.org/plugins/maven-help-plugin/                -->
<!--                                                                        -->
<!-- ====================================================================== -->
<!-- ====================================================================== -->
<!--                                                                        -->
<!-- Effective POM for project 'com.jluzh:02_JavaWebApp:war:1.0-SNAPSHOT'   -->
<!--                                                                        -->
<!-- ====================================================================== -->
<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>com.jluzh</groupId>
  <artifactId>02_JavaWebApp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>02_JavaWebApp Maven Webapp</name>
```

```
<url>http://maven.apache.org</url>
<properties>
  <jdk.version>1.8</jdk.version>
  <junit.version>4.11</junit.version>
</properties>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
<repositories>
  <repository>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <id>central</id>
    <name>Central Repository</name>
    <url>https://repo.maven.apache.org/maven2</url>
  </repository>
</repositories>
<pluginRepositories>
  <pluginRepository>
    <releases>
      <updatePolicy>never</updatePolicy>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <id>central</id>
    <name>Central Repository</name>
    <url>https://repo.maven.apache.org/maven2</url>
  </pluginRepository>
</pluginRepositories>
<build>
  <sourceDirectory>E:\Code\Maven\02_JavaWebApp\src\main\java</sourceDirectory>
  <scriptSourceDirectory>E:\Code\Maven\02_JavaWebApp\src\main\scripts</scriptSourceDirectory>
  <testSourceDirectory>E:\Code\Maven\02_JavaWebApp\src\test\java</testSourceDirectory>
  <outputDirectory>E:\Code\Maven\02_JavaWebApp\target\classes</outputDirectory>
  <testOutputDirectory>E:\Code\Maven\02_JavaWebApp\target\test-classes</testOutputDirectory>
  <resources>
    <resource>
      <directory>E:\Code\Maven\02_JavaWebApp\src\main\resources</directory>
    </resource>
  </resources>
  <testResources>
    <testResource>
      <directory>E:\Code\Maven\02_JavaWebApp\src\test\resources</directory>
    </testResource>
  </testResources>
```

```
<directory>E:\Code\Maven\02_JavaWebApp\target</directory>
<finalName>02_JavaWebApp</finalName>
<pluginManagement>
  <plugins>
    <plugin>
      <artifactId>maven-antrun-plugin</artifactId>
      <version>1.3</version>
    </plugin>
    <plugin>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>2.2-beta-5</version>
    </plugin>
    <plugin>
      <artifactId>maven-dependency-plugin</artifactId>
      <version>2.8</version>
    </plugin>
    <plugin>
      <artifactId>maven-release-plugin</artifactId>
      <version>2.5.3</version>
    </plugin>
  </plugins>
</pluginManagement>
<plugins>
  <plugin>
    <artifactId>maven-eclipse-plugin</artifactId>
    <version>2.9</version>
    <configuration>
      <downloadSources>true</downloadSources>
      <downloadJavadocs>false</downloadJavadocs>
      <wtpversion>2.0</wtpversion>
    </configuration>
  </plugin>
  <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>2.3.2</version>
    <executions>
      <execution>
        <id>default-compile</id>
        <phase>compile</phase>
        <goals>
          <goal>compile</goal>
        </goals>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </execution>
      <execution>
        <id>default-testCompile</id>
        <phase>test-compile</phase>
        <goals>
          <goal>testCompile</goal>
```

```
      </goals>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </execution>
  </executions>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
  </configuration>
</plugin>
<plugin>
  <groupId>org.apache.tomcat.maven</groupId>
  <artifactId>tomcat7-maven-plugin</artifactId>
  <version>2.2</version>
  <configuration>
    <path>/02_JavaWebApp</path>
  </configuration>
</plugin>
<plugin>
  <artifactId>maven-clean-plugin</artifactId>
  <version>2.5</version>
  <executions>
    <execution>
      <id>default-clean</id>
      <phase>clean</phase>
      <goals>
        <goal>clean</goal>
      </goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <artifactId>maven-resources-plugin</artifactId>
  <version>2.6</version>
  <executions>
    <execution>
      <id>default-testResources</id>
      <phase>process-test-resources</phase>
      <goals>
        <goal>testResources</goal>
      </goals>
    </execution>
    <execution>
      <id>default-resources</id>
      <phase>process-resources</phase>
      <goals>
        <goal>resources</goal>
      </goals>
    </execution>
  </executions>
```

```
</plugin>
<plugin>
  <artifactId>maven-war-plugin</artifactId>
  <version>2.2</version>
  <executions>
    <execution>
      <id>default-war</id>
      <phase>package</phase>
      <goals>
        <goal>war</goal>
      </goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.12.4</version>
  <executions>
    <execution>
      <id>default-test</id>
      <phase>test</phase>
      <goals>
        <goal>test</goal>
      </goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <artifactId>maven-install-plugin</artifactId>
  <version>2.4</version>
  <executions>
    <execution>
      <id>default-install</id>
      <phase>install</phase>
      <goals>
        <goal>install</goal>
      </goals>
    </execution>
  </executions>
</plugin>
<plugin>
  <artifactId>maven-deploy-plugin</artifactId>
  <version>2.7</version>
  <executions>
    <execution>
      <id>default-deploy</id>
      <phase>deploy</phase>
      <goals>
        <goal>deploy</goal>
      </goals>
    </execution>
  </executions>
```

```
    </plugin>
    <plugin>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.3</version>
      <executions>
        <execution>
          <id>default-site</id>
          <phase>site</phase>
          <goals>
            <goal>site</goal>
          </goals>
          <configuration>
            <outputDirectory>E:\Code\Maven\02_JavaWebApp\target\site</outputDirectory>
            <reportPlugins>
              <reportPlugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-project-info-reports-plugin</artifactId>
              </reportPlugin>
            </reportPlugins>
          </configuration>
        </execution>
        <execution>
          <id>default-deploy</id>
          <phase>site-deploy</phase>
          <goals>
            <goal>deploy</goal>
          </goals>
          <configuration>
            <outputDirectory>E:\Code\Maven\02_JavaWebApp\target\site</outputDirectory>
            <reportPlugins>
              <reportPlugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-project-info-reports-plugin</artifactId>
              </reportPlugin>
            </reportPlugins>
          </configuration>
        </execution>
      </executions>
      <configuration>
        <outputDirectory>E:\Code\Maven\02_JavaWebApp\target\site</outputDirectory>
        <reportPlugins>
          <reportPlugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-project-info-reports-plugin</artifactId>
          </reportPlugin>
        </reportPlugins>
      </configuration>
    </plugin>
  </plugins>
</build>
<reporting>
  <outputDirectory>E:\Code\Maven\02_JavaWebApp\target\site</outputDirectory>
```

```
  </reporting>
</project>
```

可以看到默认的项目源文件夹结构，输出目录，插件，资料库，报表目录，Maven 将使用它们来执行预期的目标。

9、Maven 构建生命周期

构建生命周期是什么？

构建生命周期阶段的目标是执行顺序是一个良好定义的序列。
这里使用一个例子，一个典型的 Maven 构建生命周期是由下列顺序的阶段：

阶段	处理	描述
prepare-resources	资源复制	资源复制可以进行定制
compile	执行编译	源代码编译在此阶段完成
package	打包	创建 JAR/WAR 包如在 pom.xml 中定义提及的包
install	安装	这一阶段在本地/远程 Maven 仓库安装程序包

可用于注册必须执行一个特定的阶段之前或之后的目标，有之前处理和之后阶段。
当 Maven 开始建立一个项目，它通过定义序列阶段步骤和执行注册的每个阶段的目标。

Maven 有以下三种标准的生命周期：

- clean
- default(或 build)
- site

目标代表一个特定的任务，它有助于项目的建设和管理。可以被绑定到零个或多个生成阶段。一个没有绑定到任何构建阶段的目标，它的构建生命周期可以直接调用执行。
执行的顺序取决于目标和构建阶段折调用顺序。例如，考虑下面的命令。清理和打包（mvn clean）参数的构建阶段，而 dependency:copy-dependencies package 是一个目标。

```
mvn clean dependency:copy-dependencies package
```

在这里，清洁的阶段，将首先执行，然后是依赖关系：复制依赖性的目标将被执行，并终于将执行包阶段。

清洁生命周期

当我们执行命令 mvn clean 命令后，Maven 调用清洁的生命周期由以下几个阶段组成：

- pre-clean
- clean

- post-clean

Maven 清洁目标（clean:clean）被绑定清洁干净的生命周期阶段。clean:clean 目标删除 build 目录下的构建输出。因此，当 mvn clean 命令执行时，Maven 会删除编译目录。

目标清洁生命周期在上述阶段，我们可以自定义此行为。
在下面的示例中，我们将附加 maven-antrun-plugin:run 对目标进行预清洁，清洁和清洁后这三个阶段。这将使我们能够调用的信息显示清理生命周期的各个阶段。
现在来创建了一个 pom.xml 文件在 ***C:\MVN*** **项目文件夹中，具体内容如下：**

```
<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>com.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

     <plugin>

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

     <artifactId>maven-antrun-plugin</artifactId>

     <version>1.1</version>

     <executions>

         <execution>

              <id>id.pre-clean</id>

              <phase>pre-clean</phase>

              <goals>

                  <goal>run</goal>

              </goals>

              <configuration>

                  <tasks>

                       <echo>pre-clean phase</echo>

                  </tasks>

              </configuration>

         </execution>

         <execution>

              <id>id.clean</id>

              <phase>clean</phase>
```

```
            <goals>
             <goal>run</goal>
            </goals>
            <configuration>
                <tasks>
                     <echo>clean phase</echo>
                </tasks>
            </configuration>
        </execution>
        <execution>
            <id>id.post-clean</id>
            <phase>post-clean</phase>
            <goals>
                <goal>run</goal>
            </goals>
            <configuration>
                <tasks>
                     <echo>post-clean phase</echo>
                </tasks>
            </configuration>
        </execution>
    </executions>
    </plugin>
</plugins>
</build>
</project>
```

现在，打开命令控制台，到该文件夹包含 pom.xml 并执行以下 mvn 命令。

```
C:\MVN\project>mvn post-clean
```

Maven 将开始处理并显示清理生命周期的所有阶段。

你可以尝试调整 mvn 清洁命令，该命令将显示清洁前什么都不会被执行。

默认（或生成）生命周期

这是 Maven 主要的生命周期，用于构建应用程序。它有以下 23 个阶段。

生命周期阶段	描述

validate	验证项目是否正确，并且所有必要的信息可用于完成构建过程
initialize	建立初始化状态，例如设置属性
generate-sources	产生任何的源代码包含在编译阶段
process-sources	处理源代码，例如，过滤器值
generate-resources	包含在包中产生的资源
process-resources	复制和处理资源到目标目录，准备打包阶段
compile	编译该项目的源代码
process-classes	从编译生成的文件提交处理，例如：Java 类的字节码增强/优化
generate-test-sources	生成任何测试的源代码包含在编译阶段
process-test-sources	处理测试源代码，例如，过滤器任何值
test-compile	编译测试源代码到测试目标目录
process-test-classes	处理测试代码文件编译生成的文件
test	运行测试使用合适的单元测试框架（JUnit）
prepare-package	执行必要的任何操作的实际打包之前准备一个包
package	提取编译后的代码，并在其分发格式打包，如 JAR，WAR 或 EAR 文件
pre-integration-test	完成执行集成测试之前所需操作。例如，设置所需的环境
integration-test	处理并在必要时部署软件包到集成测试可以运行的环境
pre-integration-test	完成集成测试已全部执行后所需操作。例如，清理环境
verify	运行任何检查，验证包是有效的，符合质量审核规定
install	将包安装到本地存储库，它可以用作当地其他项目的依赖
deploy	复制最终的包到远程仓库与其他开发者和项目共享

有涉及到 Maven 生命周期值得一提几个重要概念：

- 当一个阶段是通过 Maven 命令调用，例如：mvn compile，只有阶段到达并包括这个阶段才会被执行。
- 不同的 Maven 目标绑定到 Maven 生命周期的不同阶段这是这取决于包类型(JAR/WAR/EAR)。

在下面的示例中，将附加 Maven 的 antrun 插件：运行目标构建生命周期的几个阶段。这将使我们能够回显的信息显示生命周期的各个阶段。
我们已经更新了在 C:\MVN\ 项目文件夹中的 pom.xml 文件。

```
<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>com.companyname.projectgroup</groupId>

<artifactId>project</artifactId>

<version>1.0</version>

<build>

<plugins>

<plugin>

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

<artifactId>maven-antrun-plugin</artifactId>

<version>1.1</version>

<executions>

     <execution>

         <id>id.validate</id>

         <phase>validate</phase>

         <goals>

              <goal>run</goal>          </goals>

         <configuration>

              <tasks>

                  <echo>validate phase</echo>

              </tasks>

         </configuration>

     </execution>

     <execution>

         <id>id.compile</id>

         <phase>compile</phase>

         <goals>

              <goal>run</goal>
```

```
        </goals>
        <configuration>
            <tasks>
                <echo>compile phase</echo>
            </tasks>
        </configuration>
    </execution>
    <execution>
        <id>id.test</id>
        <phase>test</phase>
        <goals>
            <goal>run</goal>
        </goals>
        <configuration>
            <tasks>
                <echo>test phase</echo>
            </tasks>
        </configuration>
    </execution>
    <execution>
            <id>id.package</id>
            <phase>package</phase>
            <goals>
                <goal>run</goal>
            </goals>
            <configuration>
            <tasks>
                <echo>package phase</echo>
            </tasks>
        </configuration>
    </execution>
    <execution>
        <id>id.deploy</id>
        <phase>deploy</phase>
        <goals>
```

```
            <goal>run</goal>
        </goals>
        <configuration>
        <tasks>
            <echo>deploy phase</echo>
        </tasks>
        </configuration>
    </execution>
</executions>
</plugin>
</plugins>
</build>
</project>
```

现在，打开命令控制台，进入包含 pom.xml 并执行以下 mvn 命令。

```
C:\MVN\project>mvn compile
```

编译阶段，Maven 将开始构建生命周期的阶段处理并显示。

网站的生命周期

Maven 的网站插件通常用于创建新的文档，创建报告，部署网站等。
阶段

- pre-site
- site
- post-site
- site-deploy

在下面的示例中，我们将附加 maven-antrun-plugin:run 目标网站的生命周期的所有阶段。这将使我们能够调用短信显示的生命周期的各个阶段。
现在更新 pom.xml 文件在 C:\MVN\ 项目文件夹中。

```
<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>com.companyname.projectgroup</groupId>
<artifactId>project</artifactId>
<version>1.0</version>
<build>
```

```
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-antrun-plugin</artifactId>
<version>1.1</version>
     <executions>
         <execution>
              <id>id.pre-site</id>
              <phase>pre-site</phase>
              <goals>
                  <goal>run</goal>
              </goals>
              <configuration>
                  <tasks>
                       <echo>pre-site phase</echo>
                  </tasks>
              </configuration>
         </execution>
         <execution>
              <id>id.site</id>
              <phase>site</phase>
              <goals>
              <goal>run</goal>
              </goals>
              <configuration><tasks>
                       <echo>site phase</echo>
                  </tasks>
              </configuration>
         </execution>
         <execution>
              <id>id.post-site</id>
              <phase>post-site</phase>
              <goals>
                  <goal>run</goal>
              </goals>
```

```
            <configuration>
                <tasks>
                    <echo>post-site phase</echo>
                </tasks>
            </configuration>
        </execution>
        <execution>
            <id>id.site-deploy</id>
            <phase>site-deploy</phase>
            <goals>
                <goal>run</goal>
            </goals>
            <configuration>
                <tasks>
                    <echo>site-deploy phase</echo>
                </tasks>
            </configuration>
        </execution>
    </executions>
</plugin>
</plugins>
</build>
</project>
```

打开命令控制台，进入该文件夹包含 pom.xml 并执行以下 mvn 命令。

```
C:\MVN\project>mvn site
```

Maven 将开始处理并显示网站的生命周期阶段的各个阶段。

10、Maven 插件

什么是 Maven 的插件？

Maven 是一个执行插件的框架，每一个任务实际上是由插件完成的。Maven 插件通常用于：

- 创建 jar 文件
- 创建 war 文件
- 编译代码文件

- 进行代码单元测试
- 创建项目文档
- 创建项目报告

一个插件通常提供了一组目标，可使用以下语法来执行：

```
mvn [plugin-name]:[goal-name]
```

例如，一个 Java 项目可以使用 Maven 编译器插件来编译目标，通过运行以下命令编译

```
mvn compiler:compile
```

插件类型

Maven 提供以下两种类型插件：

类型	描述
构建插件	在生成过程中执行，并在 pom.xml 中的<build/> 元素进行配置
报告插件	在网站生成期间执行，在 pom.xml 中的 <reporting/> 元素进行配置

以下是一些常见的插件列表：

插件	描述
clean	编译后的清理目标，删除目标目录
compiler	编译 Java 源文件
surefile	运行 JUnit 单元测试，创建测试报告
jar	从当前项目构建 JAR 文件
war	从当前项目构建 WAR 文件
javadoc	产生用于该项目的 Javadoc
antrun	从构建所述的任何阶段运行一组 Ant 任务

例子

我们使用 maven-antrun-plugin 插件在例子中来在控制台打印数据。现在在 C:\MVN\project 文件夹 创建一个 pom.xml 文件，内容如下：

```
<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>com.companyname.projectgroup</groupId>
<artifactId>project</artifactId>
<version>1.0</version>
<build>
<plugins>
   <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-antrun-plugin</artifactId>
   <version>1.1</version>
   <executions>
      <execution>
         <id>id.clean</id>
         <phase>clean</phase>
         <goals>
            <goal>run</goal>
         </goals>
         <configuration>
            <tasks>
               <echo>clean phase</echo>
            </tasks>
         </configuration>
      </execution>
   </executions>
   </plugin>
</plugins>
</build>
</project>
```

接下来，打开命令控制台，并转到包含 pom.xml 的文件夹并执行以下命令 mvn 命令。

```
C:\MVN\project>mvn clean
```

Maven 将开始处理并显示清洁周期/阶段

上面的例子说明了以下关键概念：

- 插件可在 pom.xml 使用的 plugin 元素来指定；

- 每个插件可以有多个目标；
- 从插件应使用它的相位元素开始处理定义阶段。这里已经使用 clean 阶段；
- 可以通过将它们绑定到插件的目标来执行配置任务。这里已经绑定 echo 任务到 maven-antrun-plugin 的运行目标；
- 就这样，Maven 将处理其余部分。如果没有可用的本地存储库，它会下载这个插件；

11、Maven 外部依赖

正如大家所了解的那样，Maven 确实使用 Maven 库的概念作依赖管理。但是，如果依赖是在远程存储库和中央存储库不提供那会怎么样？ Maven 提供为使用外部依赖的概念，就是应用在这样的场景中的。

举一个例子，在一个标准的 Maven 项目做以下的修改。

- 添加 lib 文件夹到 src 文件夹
- 复制任何的 jar 到 lib 文件夹。这里使用的是 ldapjdk.jar，这是 LDAP 操作的辅助库。

现在我们的项目结构看起来应该类似下面这样：

在这里，在项目中指定自己所用的库，它可以包含 jar 文件，但是可能无法在任何 Maven 存储库找到，那么需要从外部下载。如果代码使用这个 Maven 库但没有办法找到，那么 Maven 构建将会失败，因为它在编译阶段使用指这个库无法下载或无法找到。

要处理这种情况，需要添加外部依赖项，如使用下列方式在 Maven 的 pom.xml 。

```
<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">

   <modelVersion>4.0.0</modelVersion>

   <groupId>com.companyname.bank</groupId>

   <artifactId>consumerBanking</artifactId>

   <packaging>jar</packaging>

   <version>1.0-SNAPSHOT</version>

   <name>consumerBanking</name>
```

```
  <url>http://maven.apache.org</url>


  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>


    <dependency>
      <groupId>ldapjdk</groupId>
      <artifactId>ldapjdk</artifactId>
      <!--  依赖范围使用本系统 -->
      <scope>system</scope>
      <version>1.0</version>
      <systemPath>${basedir}\src\lib\ldapjdk.jar</systemPath>
    </dependency>
  </dependencies>


</project>
```

再看上面例子中的第二个依赖元素（dependency），它清除以下有关外部依赖的重要概念。

- 外部依赖（JAR 库的位置）可以在 pom.xml 中配置为与其他依赖的方式相同；
- 指定 groupId 同样作为库的名称；
- 指定 artifactId 同样作为库的名称
- 指定范围的系统；
- 指定相系统项目的位置；

12、Maven 项目文档

本教程学习如何一步到位地创建应用程序的文档。因此现在开始我们进入到 C:\MVN 创建 java 应用程序项目：consumerBanking。 进入到项目文件夹中执行以下命令 mvn 命令。

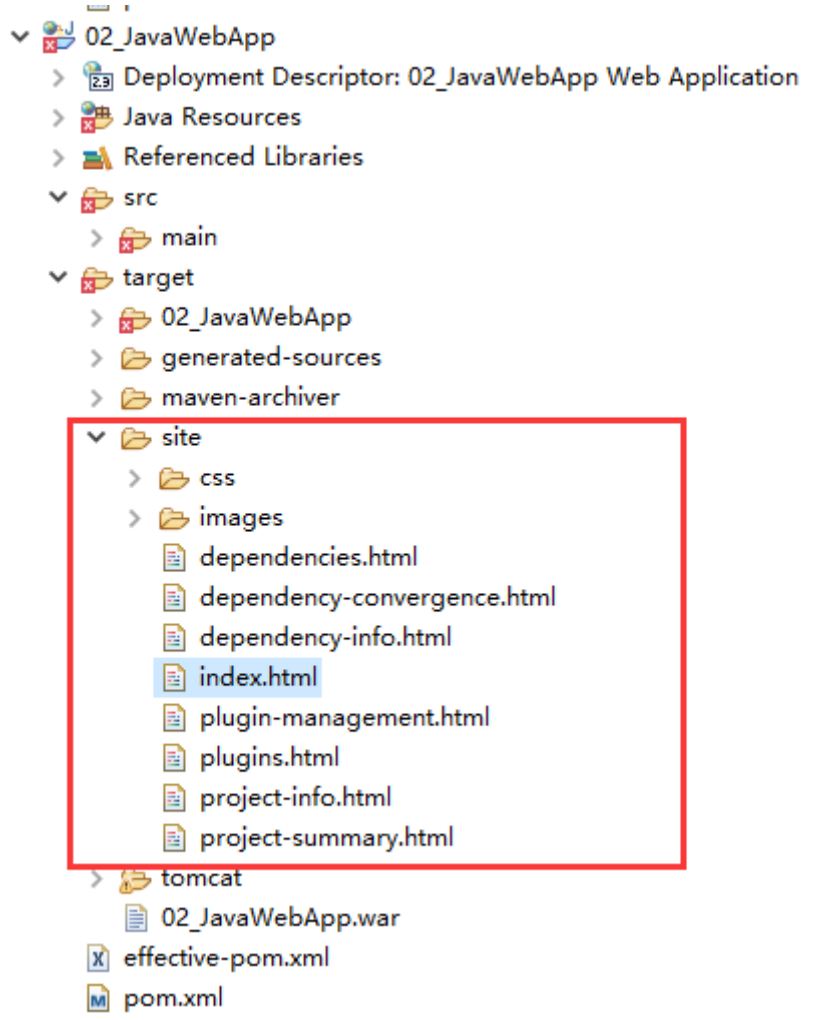
```
mvn site
```

Maven 将开始构建这个项目，输出结果如下：

```
E:\Code\Maven\02_JavaWebApp>mvn site
[INFO] Scanning for projects...
[INFO]
```

```
[INFO] Rendering site with org.apache.maven.skins:maven-default-skin:jar:1.0 skin.
[INFO] Generating "Dependencies" report    --- maven-project-info-reports-plugin:2.9
[INFO] Generating "Dependency Convergence" report    --- maven-project-info-reports-plugin:2.9
[INFO] Generating "Dependency Information" report    --- maven-project-info-reports-plugin:2.9
[INFO] Generating "About" report    --- maven-project-info-reports-plugin:2.9
[INFO] Generating "Plugin Management" report    --- maven-project-info-reports-plugin:2.9
[INFO] Generating "Plugins" report    --- maven-project-info-reports-plugin:2.9
[INFO] Generating "Summary" report    --- maven-project-info-reports-plugin:2.9
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

完成后，在你的项目文件已构建完成。 Maven 会在 target 目录中创建一个 site 目录，如下图所示：



打开 ..\target\site 文件夹。点击 index.html 打开此文档，看到结果如下图所示：

Maven 会使用一个文件处理引擎： Doxia，它将会读取多个源格式并将它们转换为通用文档模型文档。以下几个是常用的格式使用来编写项目文档，这是由 Doxia 解析编写内容。

格式名称	描述	参考
APT	纯文本文档格式	http://maven.apache.org/doxia/format.html
XDoc	Maven1.x 的文档格式	http://jakarta.apache.org/site/jakarta-site2.html
FML	用于常问问题(FQA)文件	http://maven.apache.org/doxia/references/fml-format.html
XHTML	可扩展 HTML	http://en.wikipedia.org/wiki/XHTML

13、Maven 项目模板(了解)

maven 使用 Archetype 为用户提供不同类型的项目模板，它是一个非常大的列表（614 个数字）。 maven 使用下面的命令来帮助用户快速开始构建一个新的 Java 项目。

```
mvn archetype:generate
```

什么是 Archetype?

Archetype 是一个 Maven 插件，其任务是按照其模板来创建一个项目结构。在这里我们将使用 *quickstart* 原型插件来创建一个简单的 Java 应用程序。

(其实前面我们已经做过，现在是拆分来了解一下)

使用项目模板

让我们打开命令控制台，进入到 **C:\>MVN** 目录，然后执行以下命令 mvn 命令，如下代码所示：

```
C:\MVN>mvn archetype:generate
```

Maven 开始处理，并按要求选择所需的原型，执行结果如下图中所示：

```
INFO] Scanning for projects...

[INFO] Searching repository for plugin with prefix: 'archetype'.

[INFO] ------------------------------------------------------------------------

[INFO] Building Maven Default Project

[INFO]    task-segment: [archetype:generate] (aggregator-style)

[INFO] ------------------------------------------------------------------------

[INFO] Preparing archetype:generate

...
```

```
600: remote -> org.trailsframework:trails-archetype (-)
601: remote -> org.trailsframework:trails-secure-archetype (-)
602: remote -> org.tynamo:tynamo-archetype (-)
603: remote -> org.wicketstuff.scala:wicket-scala-archetype (-)
604: remote -> org.wicketstuff.scala:wicketstuff-scala-archetype
Basic setup for a project that combines Scala and Wicket,
depending on the Wicket-Scala project.
Includes an example Specs test.)
605: remote -> org.wikbook:wikbook.archetype (-)
606: remote -> org.xaloon.archetype:xaloon-archetype-wicket-jpa-glassfish (-)
607: remote -> org.xaloon.archetype:xaloon-archetype-wicket-jpa-spring (-)
608: remote -> org.xwiki.commons:xwiki-commons-component-archetype
(Make it easy to create a maven project for creating XWiki Components.)
609: remote -> org.xwiki.rendering:xwiki-rendering-archetype-macro
(Make it easy to create a maven project for creating XWiki Rendering Macros.)
610: remote -> org.zkoss:zk-archetype-component (The ZK Component archetype)
611: remote -> org.zkoss:zk-archetype-webapp (The ZK wepapp archetype)
612: remote -> ru.circumflex:circumflex-archetype (-)
613: remote -> se.vgregion.javg.maven.archetypes:javg-minimal-archetype (-)
614: remote -> sk.seges.sesam:sesam-annotation-archetype (-)
Choose a number or apply filter
(format: [groupId:]artifactId, case sensitive contains): 203:
```

按 Enter 键选择默认选项（203：maven-archetype-quickstart）

Maven 会要求原型的特定版本

```
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
Choose a number: 6:
```

按 Enter 键选择默认选项（6：maven-archetype-quickstart：1.1）

Maven 会要求填写项目细节信息。如果要使用默认值可直接按回车。也可以通过输入自己的值覆盖它们。

```
Define value for property 'groupId': : com.companyname.insurance
```

```
Define value for property 'artifactId': : health

Define value for property 'version': 1.0-SNAPSHOT:

Define value for property 'package': com.companyname.insurance:
```

Maven 会要求确认项目的细节信息，可按回车键或按 Y 来确认。

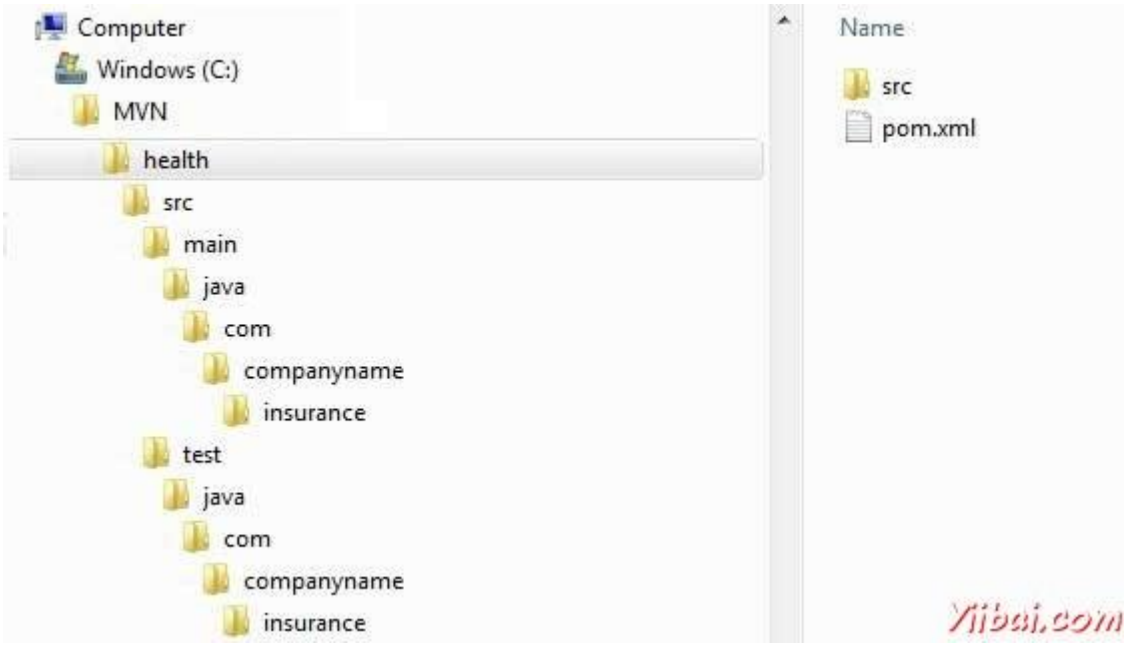
```
Confirm properties configuration:

groupId: com.companyname.insurance

artifactId: health

version: 1.0-SNAPSHOT

package: com.companyname.insurance

Y:
```

现在，Maven 将开始创建项目结构，并会显示如下内容：

```
[INFO] ------------------------------------------------------------------------

[INFO] Using following parameters for creating project

from Old (1.x) Archetype: maven-archetype-quickstart:1.1

[INFO] ------------------------------------------------------------------------

[INFO] Parameter: groupId, Value: com.companyname.insurance

[INFO] Parameter: packageName, Value: com.companyname.insurance

[INFO] Parameter: package, Value: com.companyname.insurance

[INFO] Parameter: artifactId, Value: health

[INFO] Parameter: basedir, Value: C:MVN

[INFO] Parameter: version, Value: 1.0-SNAPSHOT

[INFO] project created from Old (1.x) Archetype in dir: C:MVNhealth

[INFO] ------------------------------------------------------------------------

[INFO] BUILD SUCCESSFUL

[INFO] ------------------------------------------------------------------------

[INFO] Total time: 4 minutes 12 seconds

[INFO] Finished at: Fri Jul 13 11:10:12 IST 2012

[INFO] Final Memory: 20M/90M

[INFO] ------------------------------------------------------------------------
```

创建项目

现在进入到 C:\mvn 目录。会看到有一个 java 应用程序项目已创建了，它是在创建项目时给出 artifactId 命名：health 。 Maven 将创建一个标准的目录结构布局，如下图所示：



创建 pom.xml

Maven 项目中的生成如下所列出 pom.xml 文件，其内容如下：

```
<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>com.companyname.insurance</groupId>
   <artifactId>health</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   <name>health</name>
   <url>http://maven.apache.org</url>
   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   <dependencies>
      <dependency>
      <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>3.8.1</version>
         <scope>test</scope>
      </dependency>
```

```
  </dependencies>
</project>
```

创建 App.java

Maven 示例生成 Java 源文件，App.java 下面列出项目：

位置：C:\>MVN\health\src\main\java\com\companyname\insurance> App.java

```
package com.companyname.insurance;
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );
    }
}
```

创建 AppTest.java

Maven 实例生成 Java 源测试文件，项目中的 AppTest.java 测试文件如下面列出：

位置: C:\ > MVN > health > src > test > java > com > companyname > insurance > AppTest.java

```
package com.companyname.insurance;


import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
public class AppTest extends TestCase
{
    ...
}
```

就是这样。现在就可以看到 Maven 的功能了。可以使用 maven 单一命令来创建任何类型的项目并开始开发。

14、Maven 快照

大型应用软件一般由多个模块组成，一般它是多个团队开发同一个应用程序的不同模块，这是比较常见的场景。例如，一个团队正在对应用程序的用户界面项目(app-ui:1.0 前者是 jar 名，后者是 jar 版本) 的前端进行开发，他们使用的是数据服务工程 (data-service:1.0)。

现在，它可能会有这样的情况发生，工作在数据服务团队开发人员快速地开发 bug 修复或增强功能，他们几乎每隔一天就要释放出库到远程仓库。

现在，如果数据服务团队上传新版本后，会出现下面的问题：

- 数据服务团队应该发布更新时每次都告诉应用程序 UI 团队，他们已经发布更新了代码。
- UI 团队需要经常更新自己 pom.xml 以获得更新应用程序的版本。

为了处理这类情况，引入快照的概念，并发挥作用。

什么是快照？

快照（SNAPSHOT ）是一个特殊版本，指出目前开发拷贝。不同于常规版本，Maven 每生成一个远程存储库都会检查新的快照版本。

现在，数据服务团队将在每次发布代码后更新快照存储库为：data-service:1.0-SNAPSHOT 替换旧的 data-service。

快照与版本

在使用版本时，如果 Maven 下载所提到的版本为 data-service:1.0，那么它永远不会尝试在库中下载已经更新的版本 1.0。要下载更新的代码，data-service 的版本必须要升级到 1.1。

在使用快照（SNAPSHOT）时，Maven 会在每次应用程序 UI 团队建立自己的项目时自动获取最新的快照（data-service:1.0-SNAPSHOT）。

app-ui pom.xml

app-ui 项目使用数据服务（data-service）的 1.0-SNAPSHOT

```
<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>app-ui</groupId>
   <artifactId>app-ui</artifactId>
   <version>1.0</version>
   <packaging>jar</packaging>
   <name>health</name>
   <url>http://maven.apache.org</url>
   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   <dependencies>
      <dependency>
```

```
      <groupId>data-service</groupId>
         <artifactId>data-service</artifactId>
         <version>1.0-SNAPSHOT</version>
         <scope>test</scope>
      </dependency>
   </dependencies>
</project>
```

data-service pom.xml

数据服务（data-service）项目对于每一个微小的变化释放 1.0 快照：

```
<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>data-service</groupId>
   <artifactId>data-service</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   <name>health</name>
   <url>http://maven.apache.org</url>
   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   </project>
```

虽然，在使用快照（SNAPSHOT）时，Maven 自动获取最新的快照版本。不过我们也可以强制使用 -U 切换到任何 maven 命令来下载最新的快照版本。

```
mvn clean package -U
```

打开命令控制台，进入到 C:\MVN\app-ui 目录，然后执行以下命令 mvn 命令。

```
C:\MVN\app-ui>mvn clean package -U
```

Maven 会下载数据服务的最新快照后并开始构建该项目。

15、Maven 依赖管理

其中一个 Maven 的核心特征是依赖管理。管理依赖关系变得困难的任务一旦我们处理多模块项目（包含数百个模块/子项目）。 Maven 提供了一个高程度的控制来管理这样的场景。

传递依赖发现

这是很通常情况下，当一个库 A 依赖于其他库 B 的情况下，另一个项目 Ç 想用 A，则该项目需要使用库中 B。

在 Maven 帮助下以避免这样的要求来发现所有需要的库。 Maven 通过读取依赖项项目文件（pom.xml 中），找出它们的依赖等。

我们只需要在每个项目 POM 定义直接依赖关系。 Maven 自动处理其余部分。

传递依赖，包括库的图形可能会快速增长在很大程度上。可能出现情况下，当有重复的库。 Maven 提供一些功能来控制传递依赖程度

Feature	描述
Dependency mediation	Determines what version of a dependency is to be used when multiple versions of an artifact are encountered. If two dependency versions are at the same depth in the dependency tree, the first declared dependency will be used.
Dependency management	Directly specify the versions of artifacts to be used when they are encountered in transitive dependencies. For an example project C can include B as a dependency in its dependencyManagement section and directly control which version of B is to be used when it is ever referenced.
Dependency scope	Includes dependencies as per the current stage of the build
Excluded dependencies	Any transitive dependency can be excluede using "exclusion" element. As example, A depends upon B and B depends upon C then A can mark C as excluded.
Optional dependencies	Any transitive dependency can be marked as optional using "optional" element. As example, A depends upon B and B depends upon C. Now B marked C as optional. Then A will not use C.

依赖范围

传递依赖发现可以使用各种依赖范围如下文所述受到限制

Scope	描述
compile	This scope indicates that dependency is available in classpath of project. It is default scope.
provided	This scope indicates that dependency is to be provided by JDK or web-Server/Container at runtime.
runtime	This scope indicates that dependency is not required for compilation, but is required during execution.
test	This scope indicates that the dependency is only available for the test compilation and execution phases.
system	This scope indicates that you have to provide the system path.
import	This scope is only used when dependency is of type pom. This scopes indicates that the specified POM should be replaced with the

	dependencies in that POM's <dependencyManagement> section.

16、Eclispe IDE 集成 Maven

(目前最新版本的 Eclipse 已经集成了 Maven，且 Eclispe IDE 集成 Maven 与 Maven 创建 java/web 项目并提供 Eclipse 的支持是不一样的)

Eclipse 提供了一个很好的插件 m2eclipse 无缝将 Maven 和 Eclipse 集成在一起。

m2eclipse 一些特点如下

- 您可以从 Eclipse 运行 Maven 目标。
- 可以使用其自己的控制台查看 Maven 命令的输出在 Eclipse 里面。
- 你可以更新 maven 的依赖关系使用 IDE。
- 您可以启动 Maven 在 Eclipse 中建立。
- 它的依赖管理基于 Maven 的 pom.xml 在 Eclipse 构建路径。
- 它解决了从 Eclipse 工作区 Maven 的依赖关系，而不需要安装到本地 Maven 仓库（需要依赖项目在同一个工作区）。
- 它自动下载需要的依赖和源从远程 Maven 仓库。
- 它提供了向导，用于创建新的 Maven 项目，pom.xml 和现有项目可让 Maven 支持
- 它提供了快速搜索远程 Maven 仓库的依赖

安装 m2eclipse 插件

请使用以下链接之一安装 m2eclipse：

Eclipse	URL
Eclipse 3.5 (Gallileo)	Installing m2eclipse in Eclipse 3.5 (Gallileo)
Eclipse 3.6 (Helios)	Installing m2eclipse in Eclipse 3.6 (Helios)

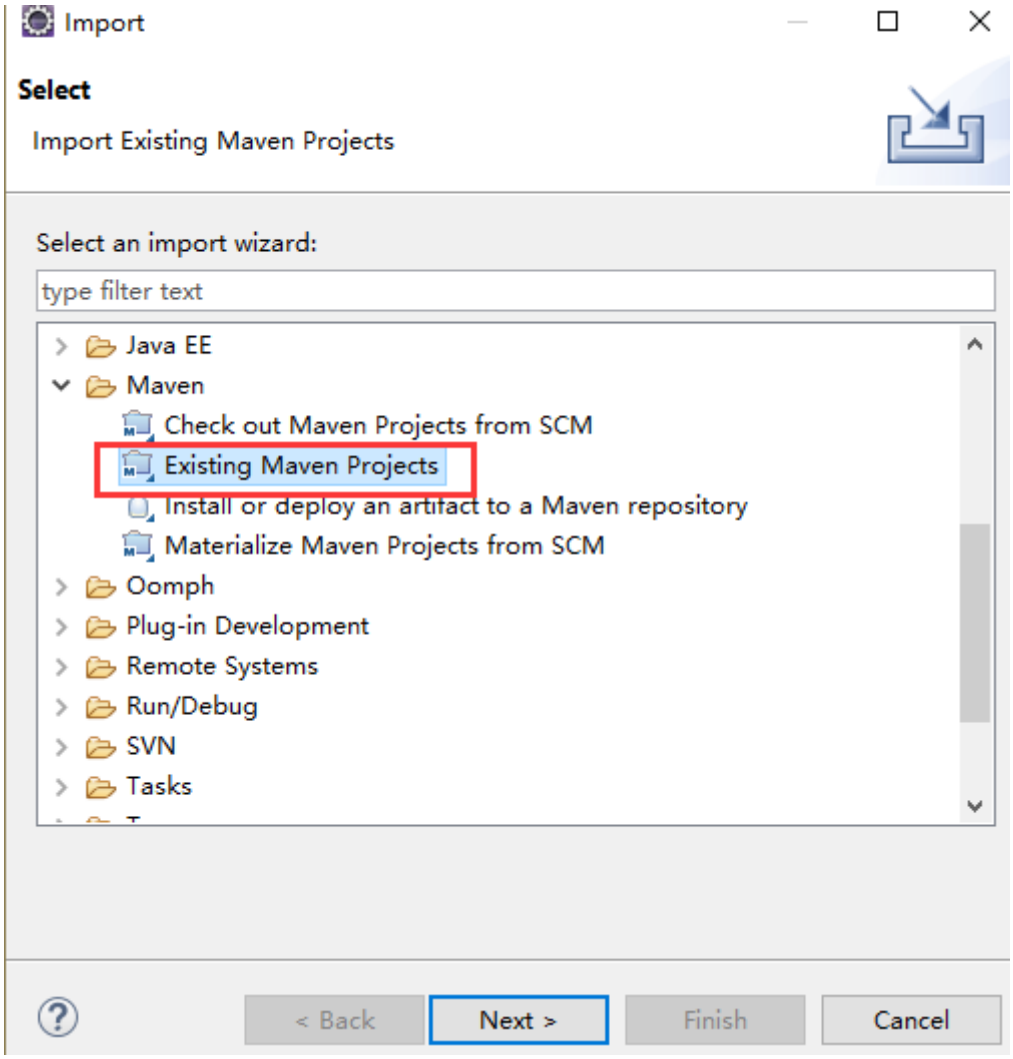
下面的例子将帮助您利用集成 Eclipse 和 Maven。

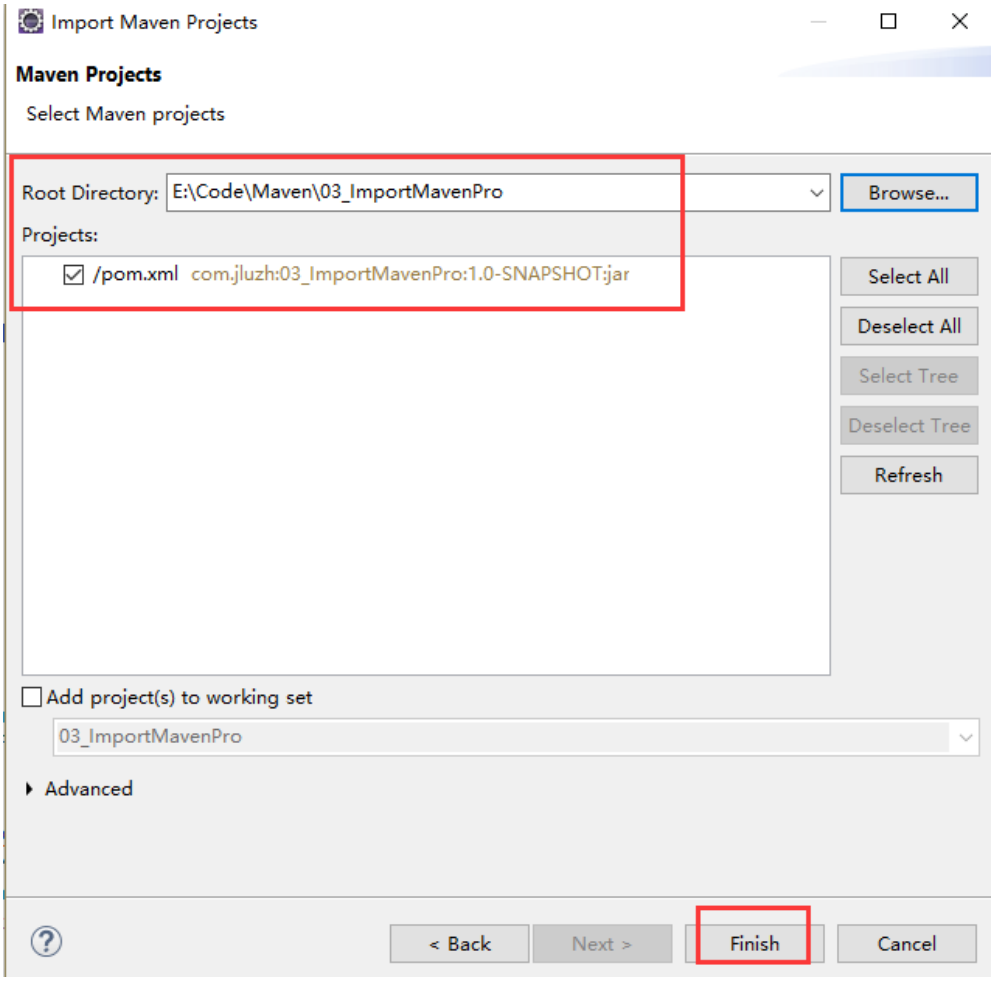
Eclipse 导入 Maven 项目

区别于导入用 Maven 创建的 Java/Web 项目，这不支持在 Eclipse 中使用 Maven 命令；而 Eclipse 导入 Maven 项目，既是 Java/Web 项目，同时也是 Maven 项目，支持在 Eclipse 中使用 Maven 命令。

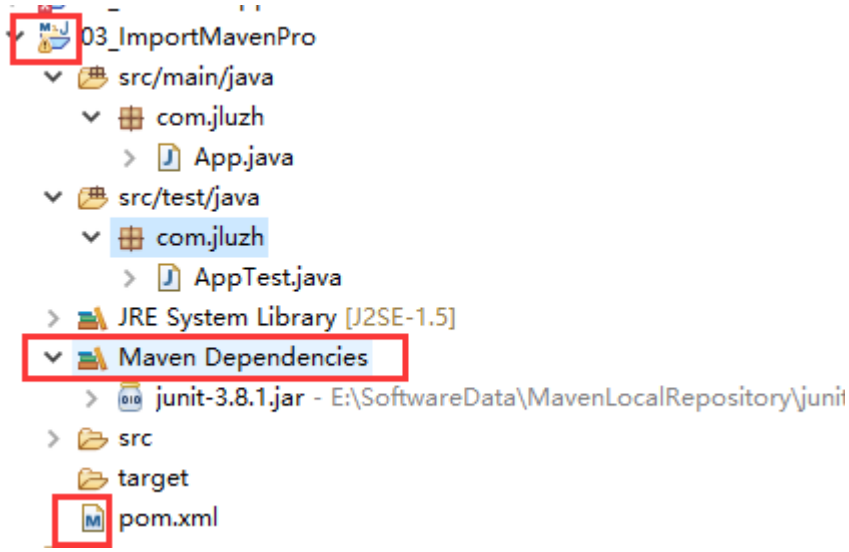
```
E:\Code\Maven>mvn archetype:generate -DgroupId=com.jluzh -DartifactId=03_ImportMavenPro -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:3.0.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< maven-archetype-plugin:3.0.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO]
[INFO] --- maven-archetype-plugin:3.0.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: E:\Code\Maven
[INFO] Parameter: package, Value: com.jluzh
[INFO] Parameter: groupId, Value: com.jluzh
[INFO] Parameter: artifactId, Value: 03_ImportMavenPro
[INFO] Parameter: packageName, Value: com.jluzh
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: E:\Code\Maven\03_ImportMavenPro
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
```

- 打开 Eclipse.
- 选择 File -- Import -- 选项.
- 选择 Maven 项目选项。单击 Next 按钮。



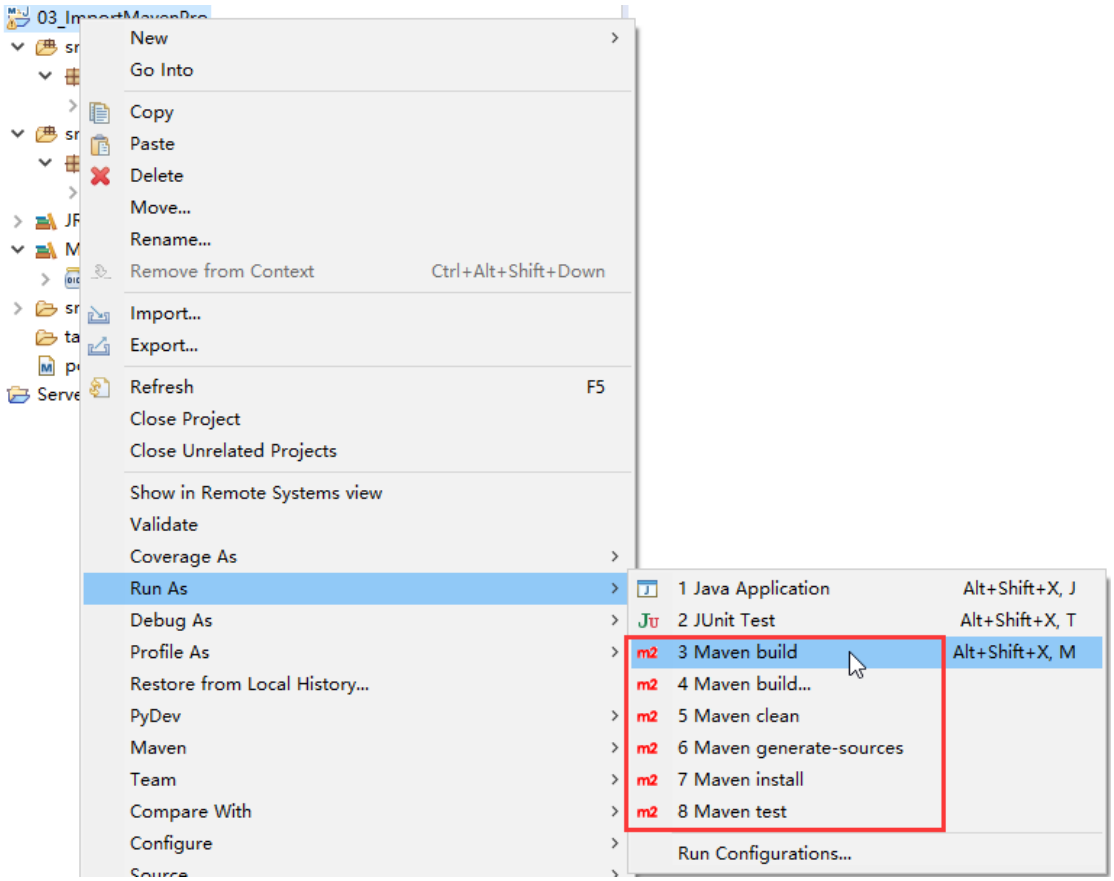


现在，你可以看到 Maven 项目在 eclipse。



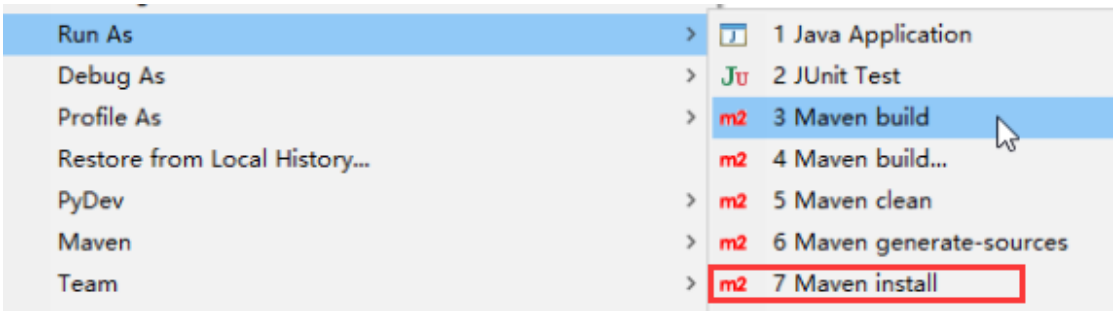
可以看到，Eclipse 已经添加 Maven 的依赖关系(Maven Dependencies)，而且项目、pom.xml 的图标都是有个 M 字，表示是 Maven 项目，这就是与之前的区别。

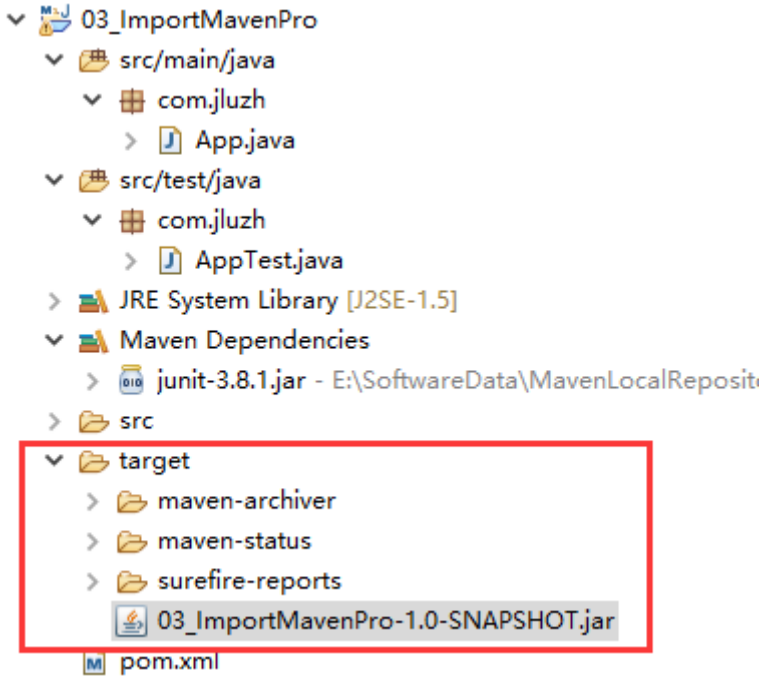
右键项目，然后选择 Run As ，可以看到 Maven 的一些命令。



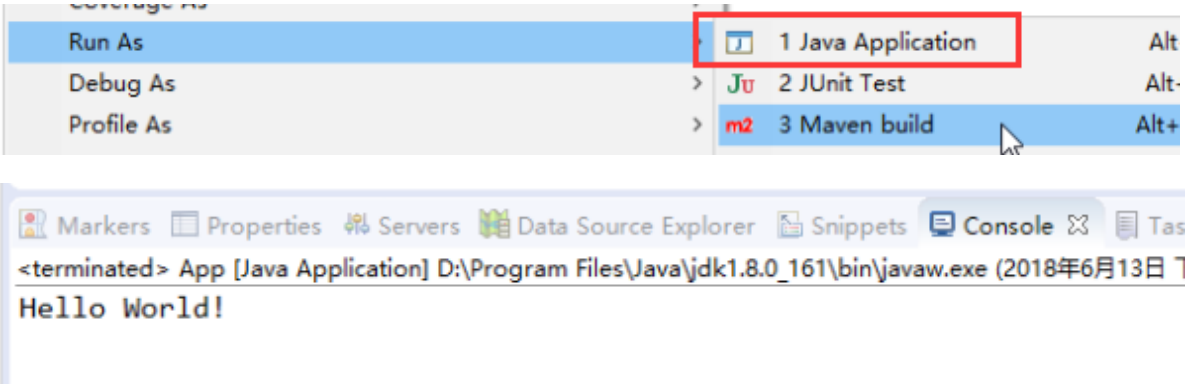
使用 Eclipse 的 Maven 来构建项目：

执行 run as -- maven install（注意哈，maven install 包括了 run as -- maven package，如果有 maven package 则执行 maven package 即可）





现在，右键点击 App.java，选择 Run As 选项，选择 Java 应用程序，会看到结果。

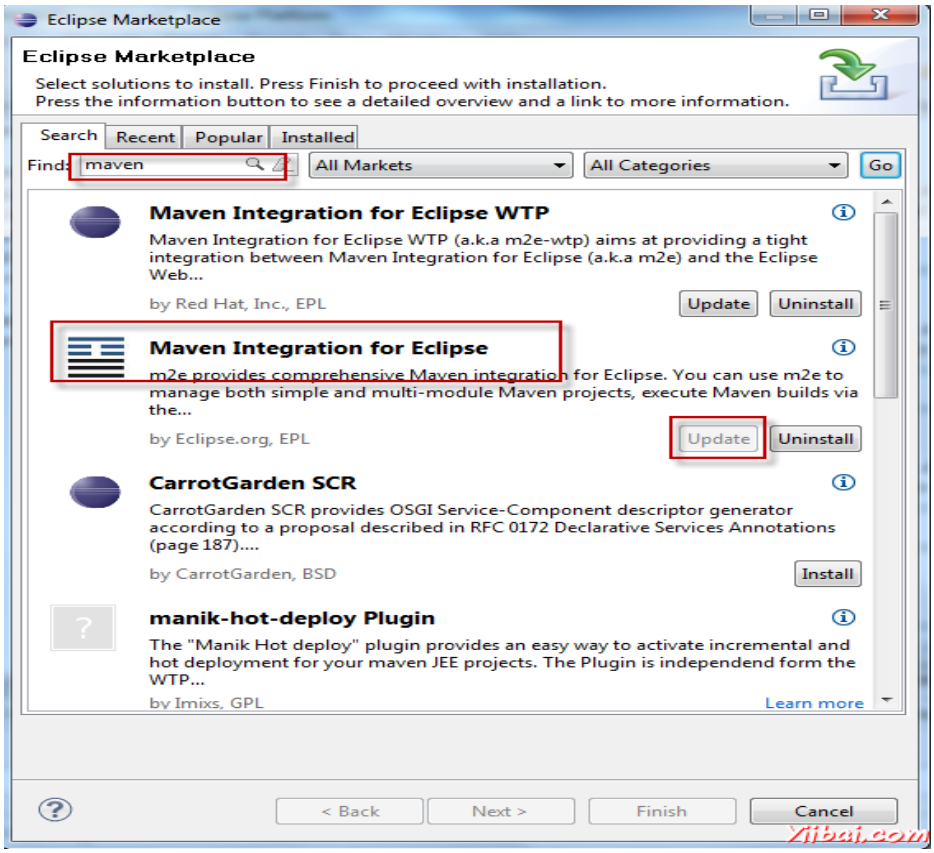


17、Eclipse 构建 Maven 项目

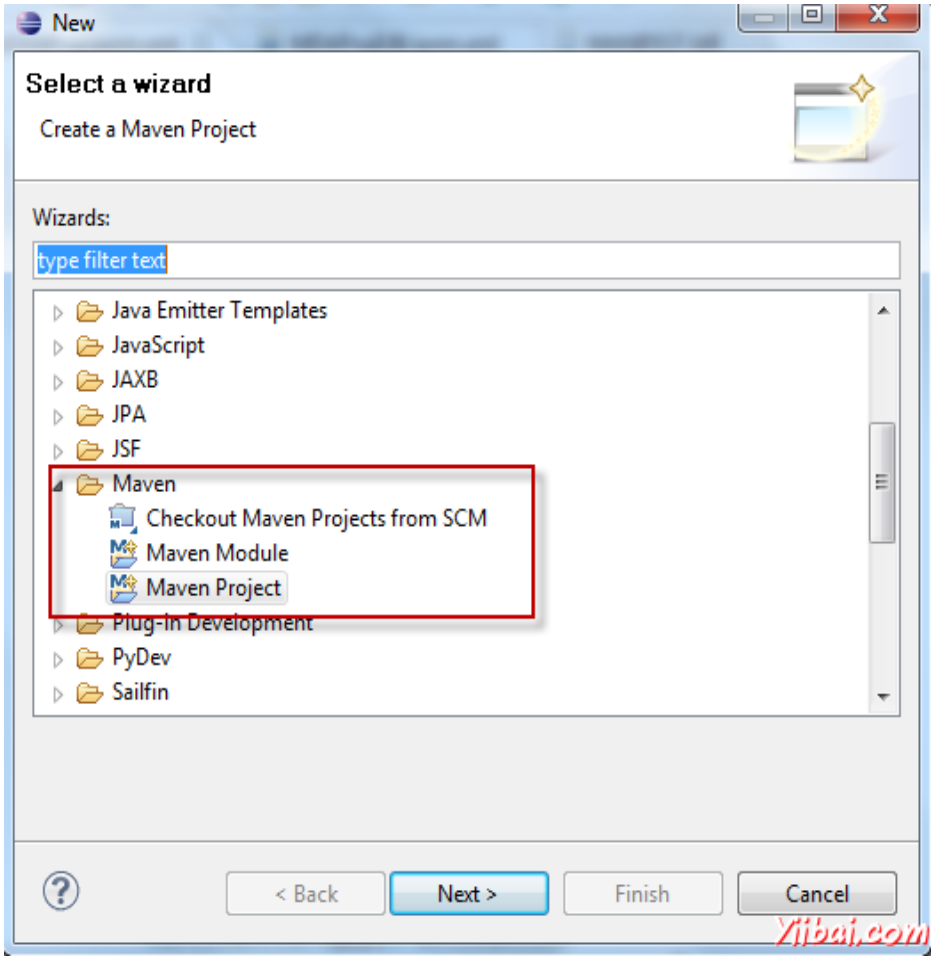
直接使用 Eclipse 带插件的图形化界面构建，比使用命令行构建方便快捷一些。

1. 安装 m2eclipse 插件(或最新版 Eclipse)

点击 eclipse 菜单栏 Help->Eclipse Marketplace 搜索到插件 Maven Integration for Eclipse 并点击安装即可(不同版本安装的方式不同)，如下图：



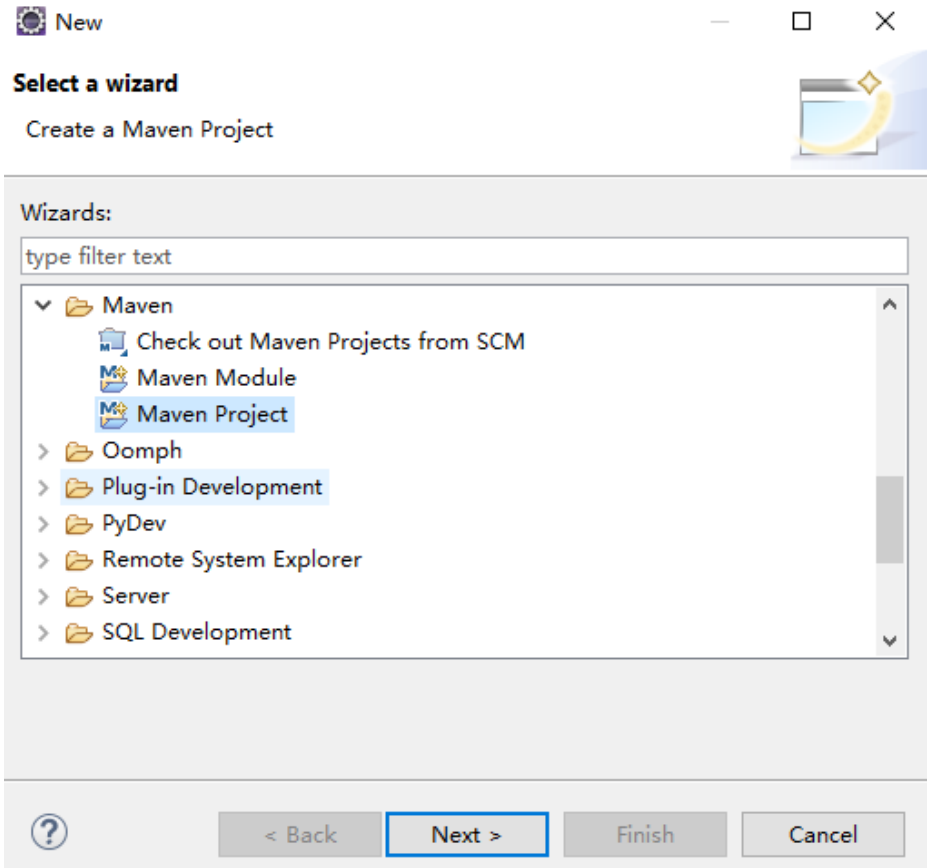
安装成成之后我们在 Eclipse 菜单栏中点击 File->New->Other,在弹出的对话框中会看到如下图所示：

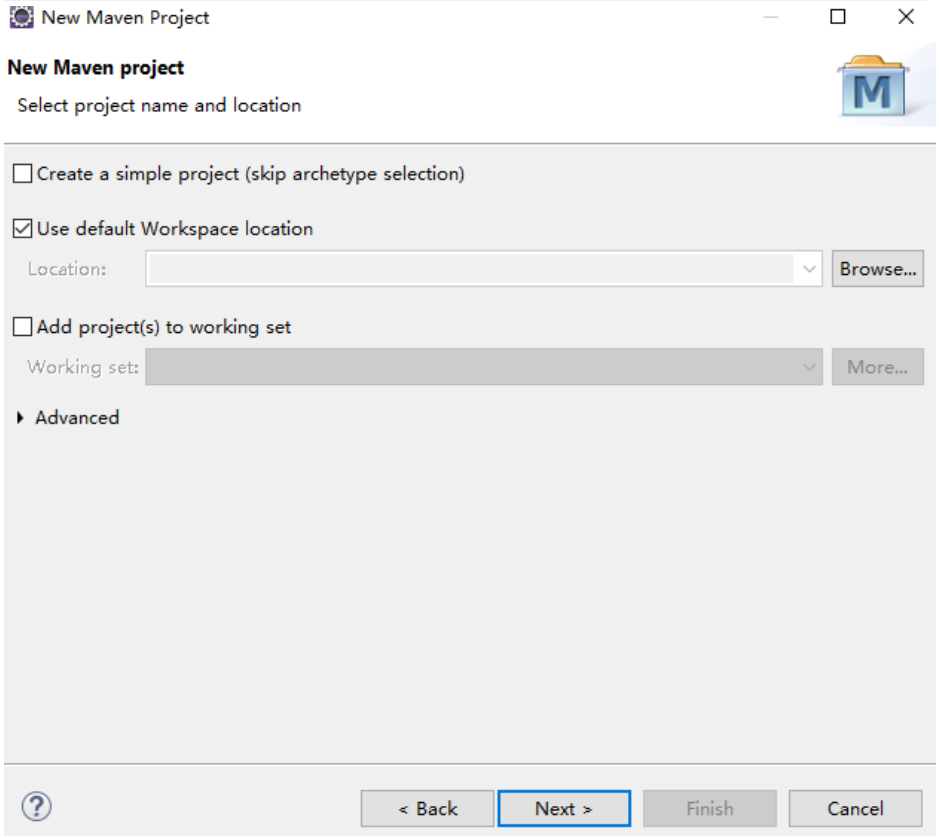


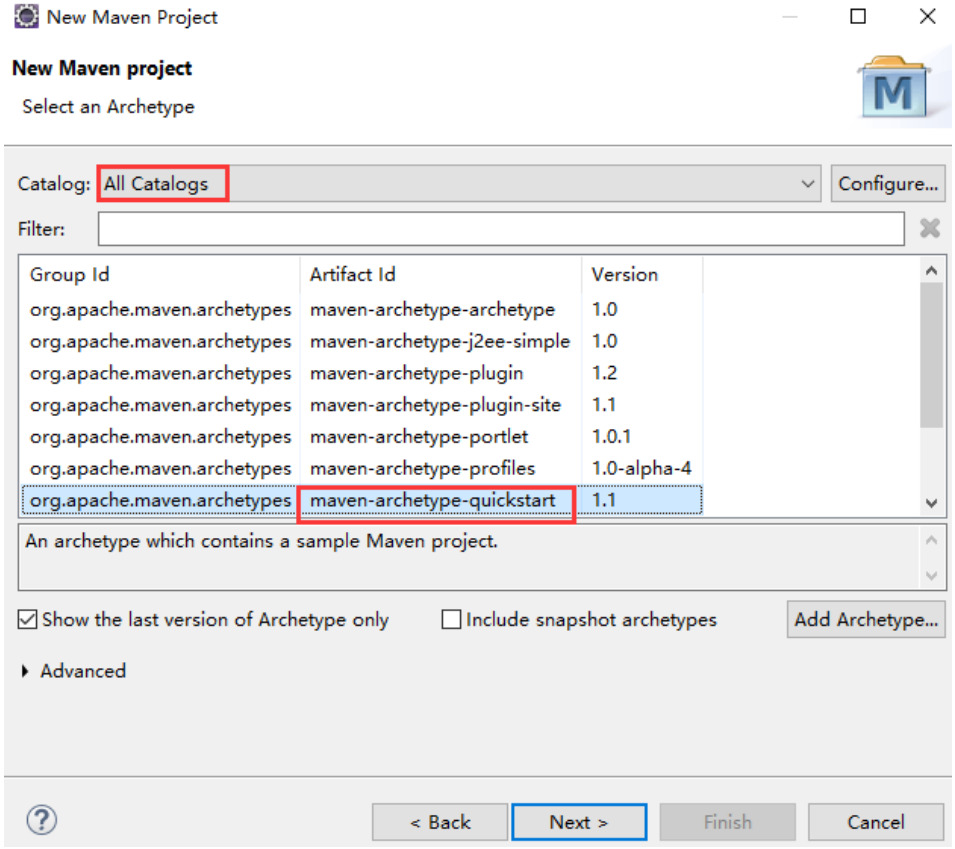
2. 构建 Maven 项目

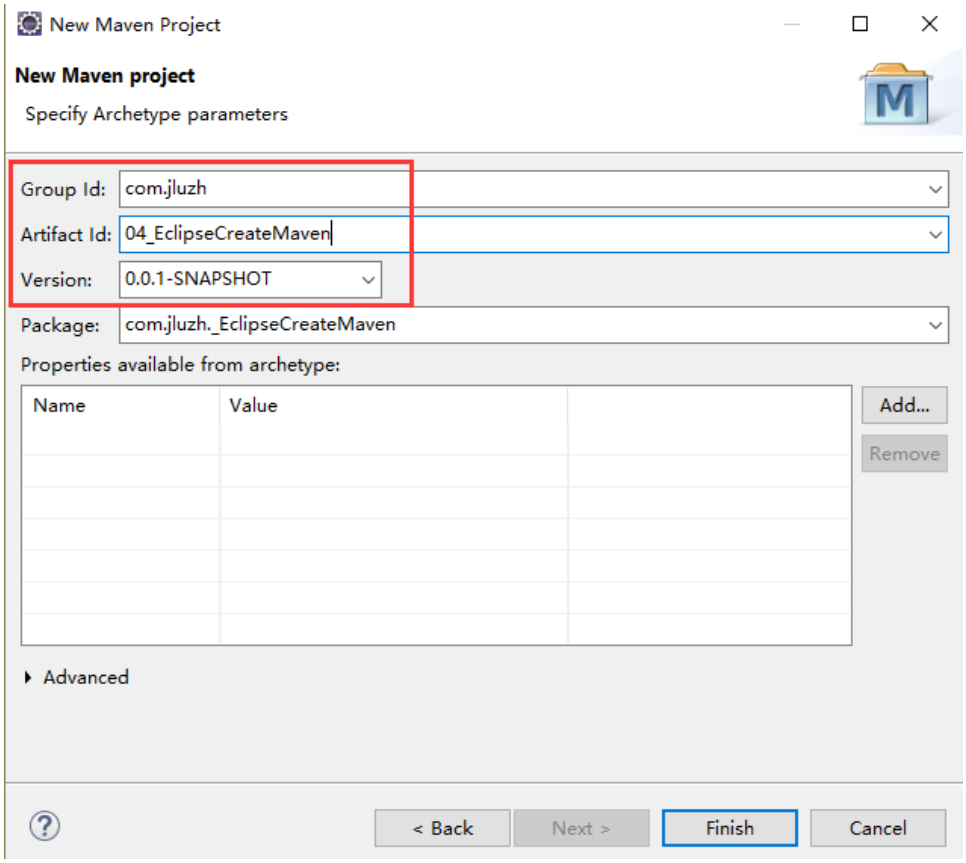
1）创建简单 Maven 项目(即 jar)

点击 Eclipse 菜单栏 File->New->Other->Maven 得到如下图所示对话框：

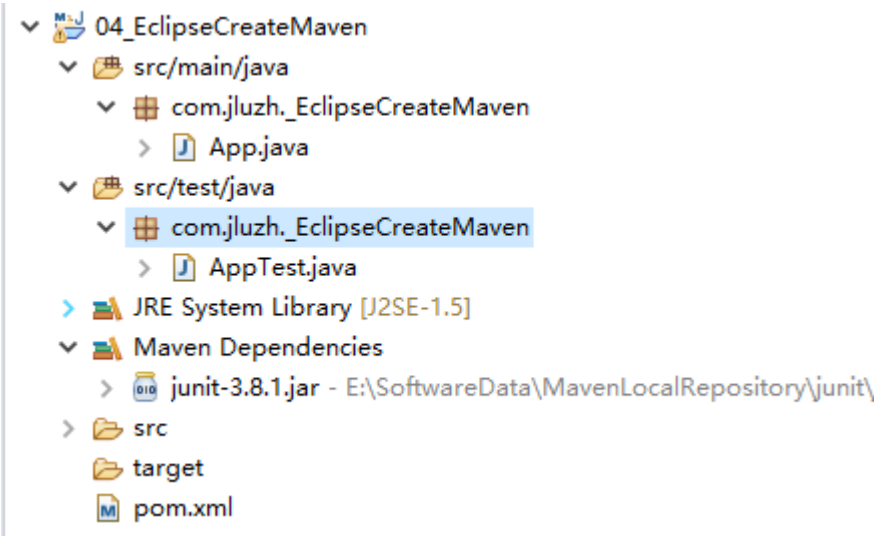





New Maven Project
New Maven project
Select an Archetype
Catalog: All Catalogs
Configure...
Filter:
Group Id
Artifact Id
Version
org.apache.maven.archetypes
maven-archetype-archetype
1.0
org.apache.maven.archetypes
maven-archetype-j2ee-simple
1.0
org.apache.maven.archetypes
maven-archetype-plugin
1.2
org.apache.maven.archetypes
maven-archetype-plugin-site
1.1
org.apache.maven.archetypes
maven-archetype-portlet
1.0.1
org.apache.maven.archetypes
maven-archetype-profiles
1.0-alpha-4
org.apache.maven.archetypes
maven-archetype-quickstart
1.1
An archetype which contains a sample Maven project.
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced
< Back
Next >
Finish
Cancel


New Maven Project
New Maven project
Specify Archetype parameters
Group Id: com.jluzh
Artifact Id: 04_EclipseCreateMaven
Version: 0.0.1-SNAPSHOT
Package: com.jluzh._EclipseCreateMaven
Properties available from archetype:
Name
Value
Add...
Remove
Advanced
< Back
Next >
Finish
Cancel

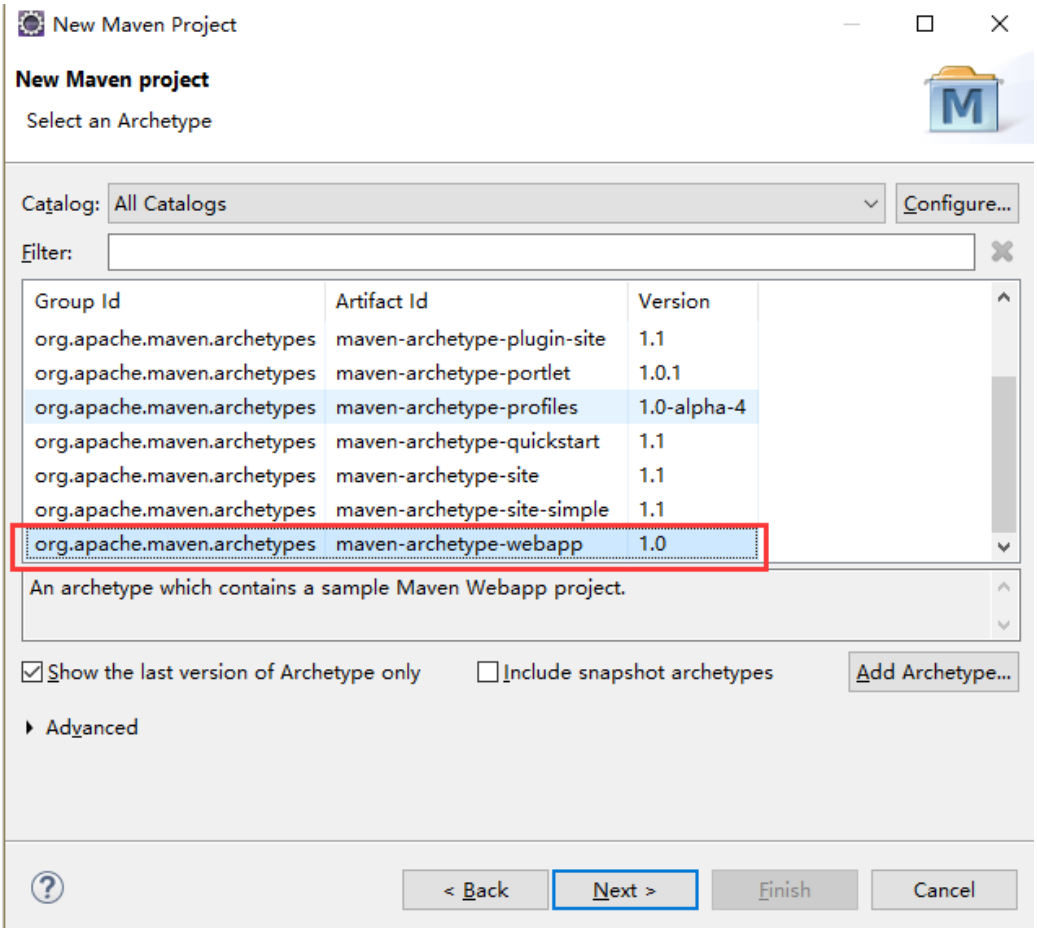
由此我们成功创建了一个简单的 Maven 项目，项目结构如图所示

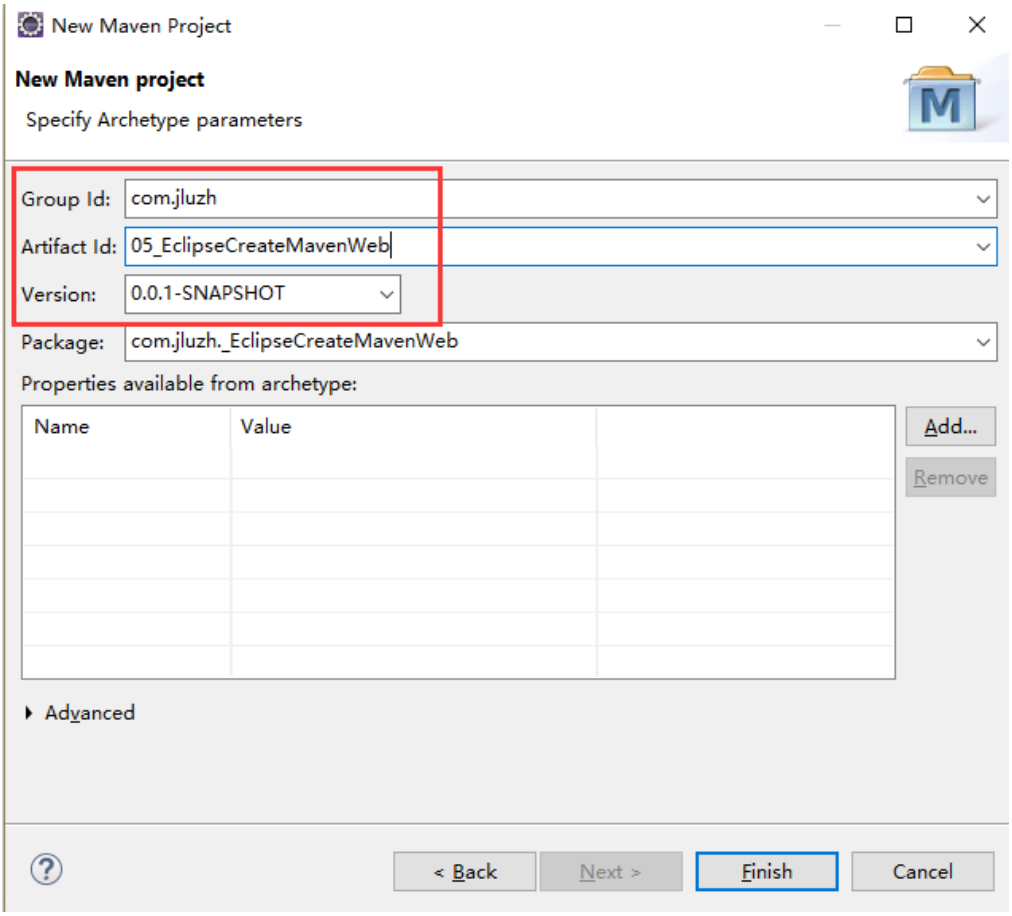


2）创建 Maven web 项目(即 war)

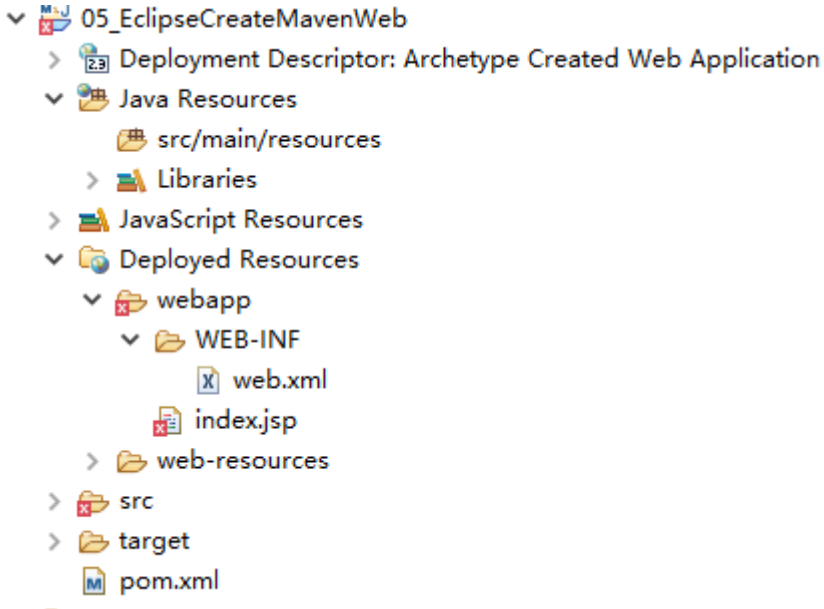
操作跟创建简单 Maven 项目类似，点击 Eclipse 菜单 File->New->Other->Maven->Maven Project。

注意选择 webapp：

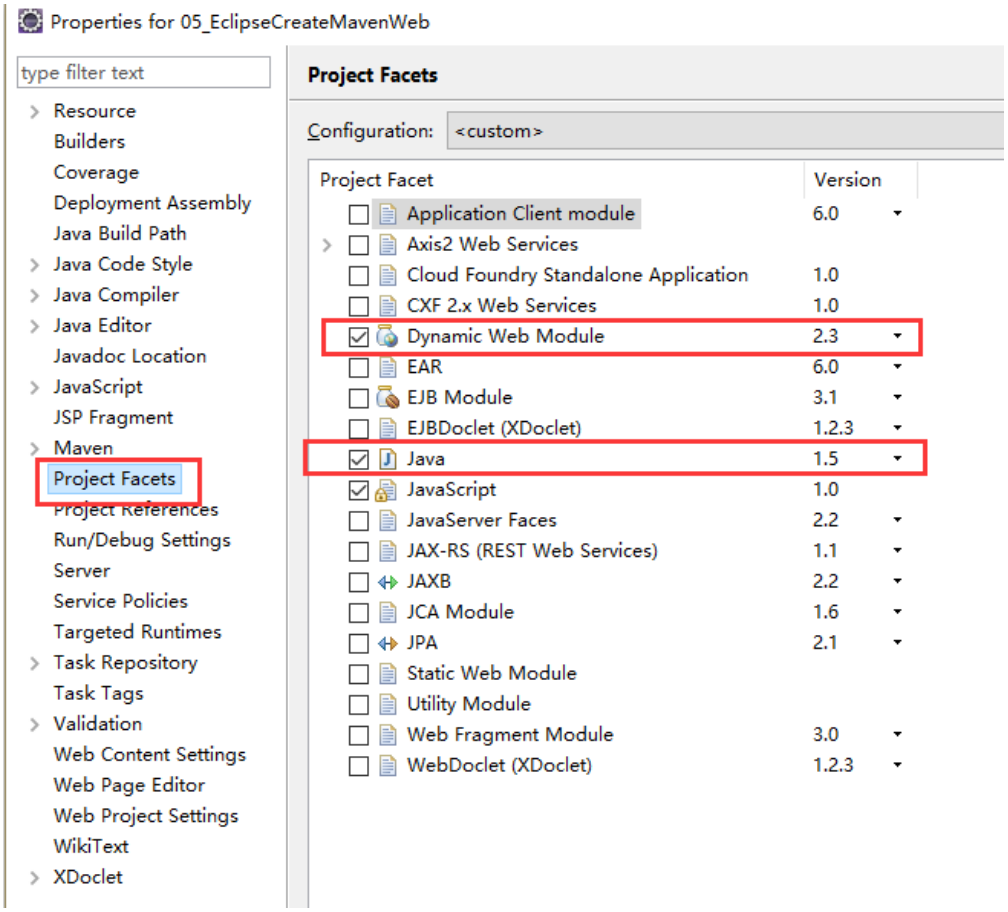




得到的 Maven web 项目结构如下图所示(Project Explorer 视图)，如果不习惯 Project Explorer 视图，可以使用 Package Explorer：



右击项目，点击 Properties->Project Facets



如上图可以看到项目为 web2.3、java1.5 当然我们也可以改成我们所需要的版本。

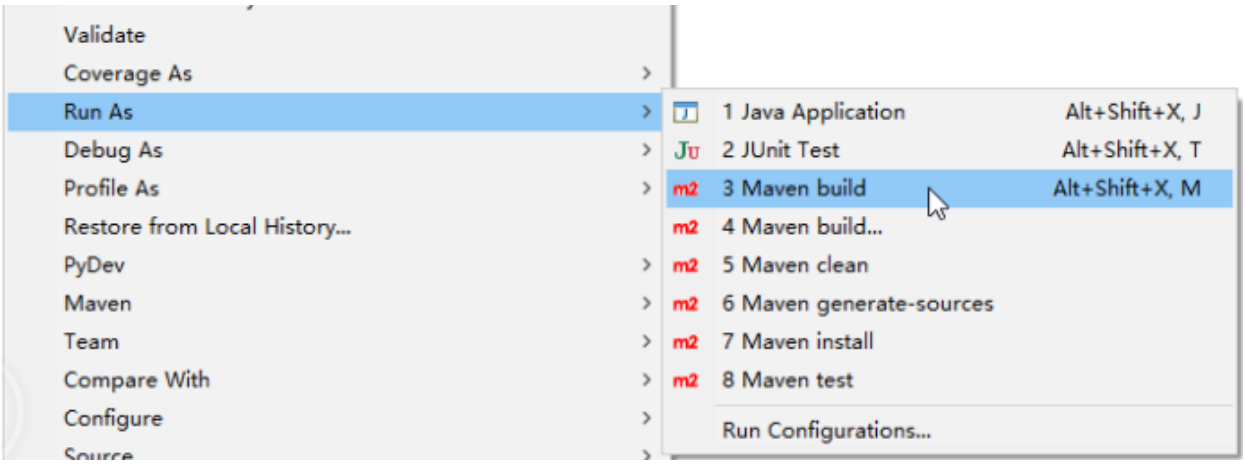
或者 打开 05_EclipseCreateMavenWeb/.settings/org.eclipse.wst.common.project.facet.core.xml，进行修改：

Xml 代码

```xml
<?xml version="1.0" encoding="UTF-8"?>
<faceted-project>
  <fixed facet="wst.jsdt.web"/>
  <installed facet="java" version="1.5"/>
  <installed facet="jst.web" version="2.3"/>
  <installed facet="wst.jsdt.web" version="1.0"/>
</faceted-project>
```

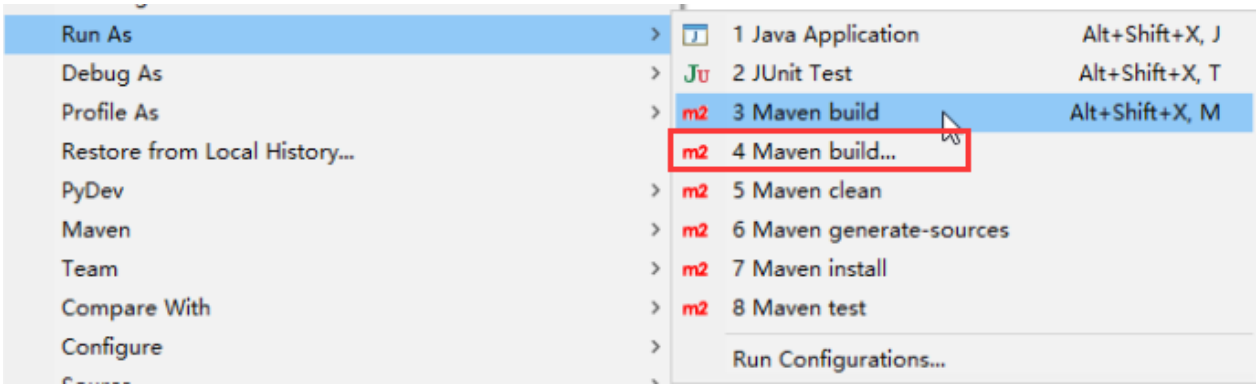
3. 运行 Maven 命令

右击项目，点击 Run as，如下图：

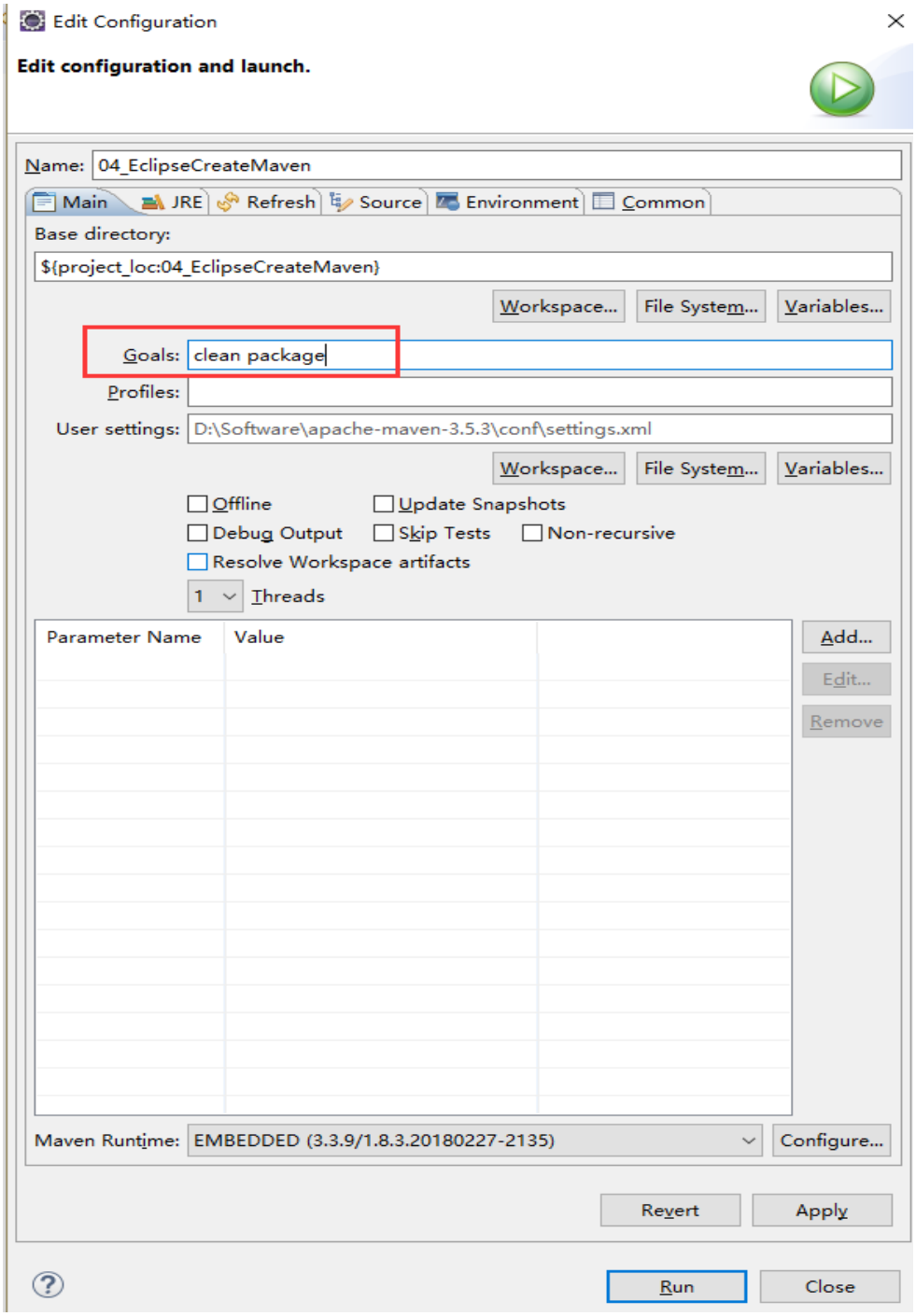


即可看到有很多现有的 maven 命令，点击即可运行，并在控制台可以看到运行信息

如果你想运行的 maven 命令在这里没有找到，点击 Maven build...创建新的命令，操作如下图所示：



如下图填入 Maven 命令，比如 eclipse 打 war 包，使用 clean package(其实就是 mvn clean package 清理并打包)，点击 Run 即可：



18、使用 Maven 模板创建项目

使用 mvn archetype:generate 从现有的 Maven 模板列表中生成项目。在 Maven 3.3.3，有超过 1000+个模板，Maven 团队已经过滤掉一些无用的模板。

通常情况下，我们只需要使用下面的两个模板：

1. maven-archetype-webapp – Java Web Project (WAR)
2. maven-archetype-quickstart – Java Project (JAR)

19、mvn package 详情

“mvn package”

当你运行“mvn package”命令，它会编译源代码，运行单元测试和包装这取决于在 pom.xml 文件的“packaging”标签。 例如,

1. If “packaging” = jar，将您的项目打包成一个“jar”文件，并把它变成你的目标文件夹。

File : pom.xml

```
...<packaging>jar</packaging> ...
```

2. 如果 “packaging” = war,将您的项目打包成“war”文件，并把它变成目标文件夹。

File : pom.xml

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

20、使用 Maven 清理项目

在基于 Maven 的项目中，很多缓存输出在“target”文件夹中。如果想建立项目部署，必须确保清理所有缓存的输出，从而能够随时获得最新的部署。

要清理项目缓存的输出，发出以下命令：

```
mvn clean
```

可以查看到输出结果...

```
E:\Code\Maven> mvn clean
```

当“mvn clean”执行，在“target”文件夹中的一切都将被删除。

部署进行生产

要部署您的项目进行生产，它总是建议使用 “mvn clean package“，以确保始终获得最新的部署。

21、使用 Maven 运行单元测试

要通过 Maven 运行单元测试，发出此命令：

```
mvn test
```

这会在你的项目中运行整个单元测试。

案例学习

创建两个单元测试，并通过 Maven 的运行它。参见一个简单的 Java 测试类：

```
package com.yiibai.core;
```

```
public class App {
	public static void main(String[] args) {
		System.out.println(getHelloWorld());
	}
	public static String getHelloWorld() {
		return "Hello World";
	}
	public static String getHelloWorld2() {
		return "Hello World 2";
	}
}
```

Unit Test 1

单元测试为 getHelloWorld()方法。

```
package com.yiibai.core;
import junit.framework.Assert;
import org.junit.Test;
public class TestApp1 {
	@Test
	public void testPrintHelloWorld() {
		Assert.assertEquals(App.getHelloWorld(), "Hello World");
	}
}
```

Unit Test 2

单元测试为 getHelloWorld2()方法。

```
package com.yiibai.core;
import junit.framework.Assert;
import org.junit.Test;
public class TestApp2 {
	@Test
	public void testPrintHelloWorld2() {
		Assert.assertEquals(App.getHelloWorld2(), "Hello World 2");
	}
}
```

运行单元测试

使用 Maven 运行单元测试看见下面的例子。

示例 1
运行整个单元测试(TestApp1 和 TestApp2)，发出以下命令：

```
mvn test
```

示例 2
为了运行单个测试(TestApp1)，发出此命令：

```
mvn -Dtest=TestApp1 test
```

示例 3
为了运行单个测试(TestApp2)，发出此命令：

```
mvn -Dtest=TestApp2 test
```

22、将项目安装到 Maven 本地资源库

在 Maven 中，可以使用“mvn install”打包项目，并自动部署到本地资源库，让其他开发人员使用它。

```
mvn install
```

注意，当“install”在执行阶段，上述所有阶段 “validate“, “compile“, “test“, “package“, “integration-test“, “verify” , 包括目前的“install”阶段将被有序执行。

mvn install 示例

一个 Java 项目，具有以下 pom.xml 文件。基于 pom.xml 文件，在“mvn install”被执行，它会打包项目为“xxxxx.jar”文件，并复制到本地存储库。

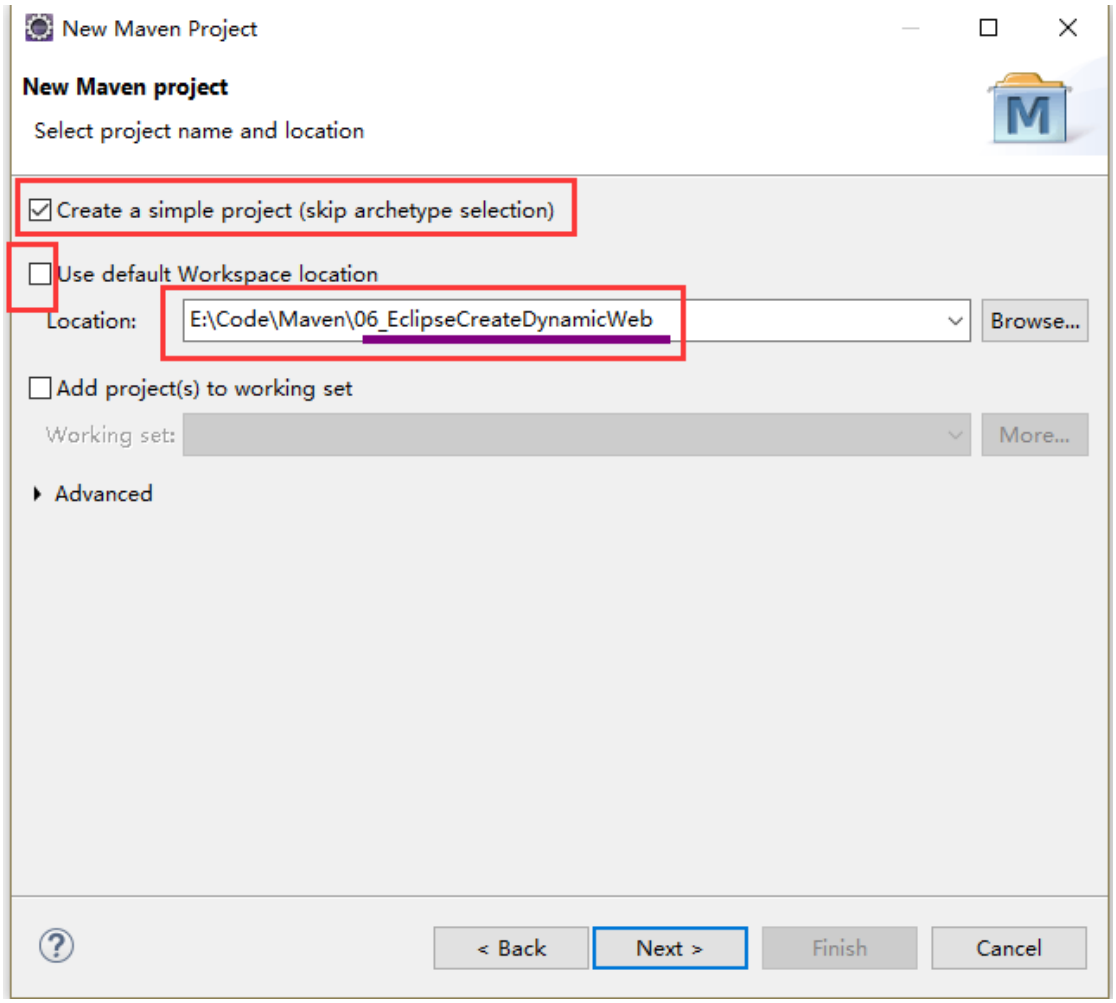
警告:
建议运行“clean”和“install”在一起使用，让您能始终部署最新的项目到本地存储库。

```
mvn clean install
```

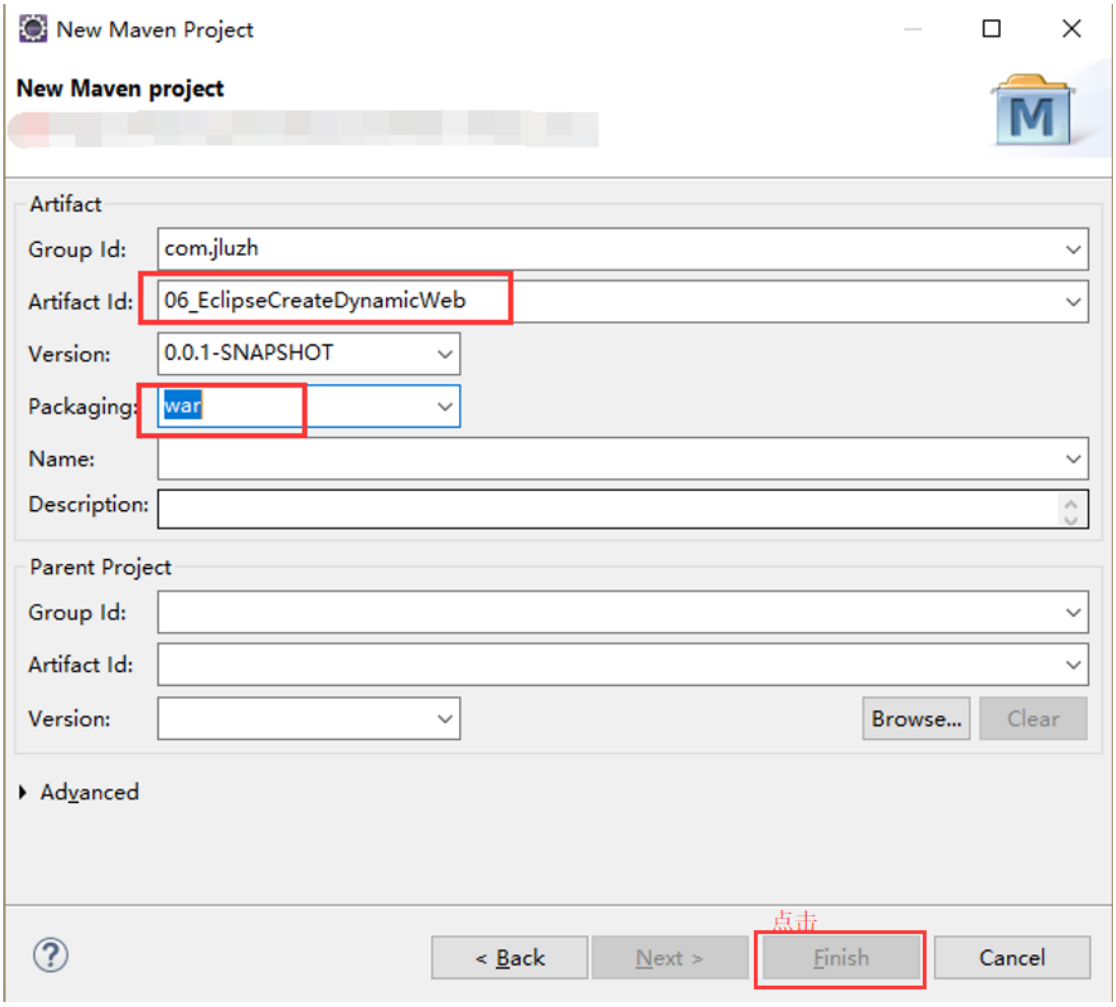
23、Eclipse 手动创建一个 Maven 动态 Web 项目

在 Eclipse IDE 中使用 maven 创建一个动态 Web 项目，不通过模板来创建，其实跟使用模板也是差不多的。

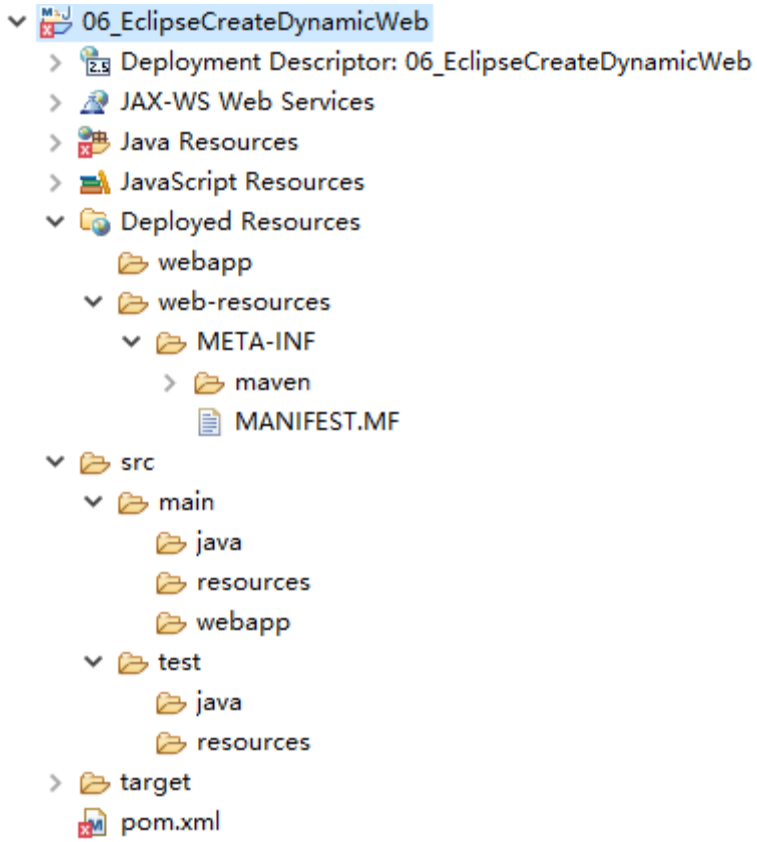
转到 *New* 菜单 *Other.. -> Maven -> Maven Project* ，跳过模板选择。如下图所示



指定 Packaging 为 war



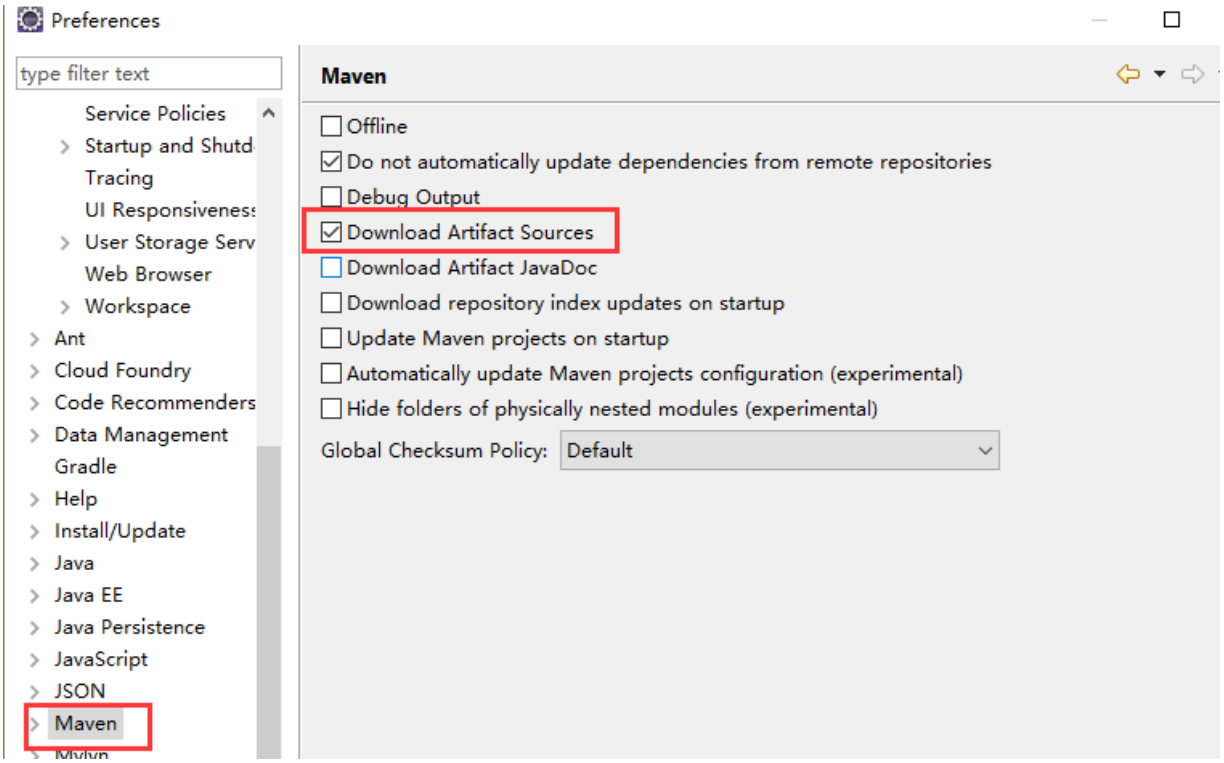
创建完的项目结构其实差不多，如下：



pom.xml:

```
<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>com.jluzh</groupId>
  <artifactId>06_EclipseCreateDynamicWeb</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <!-- 不知道为什么，不设置的话，会报错：web.xml is missing and <failOnMissingWebXml> is set to true -->
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <failOnMissingWebXml>false</failOnMissingWebXml>
  </properties>

</project>
```

勾选 Download Artifact Sources，就会在下载依赖的时候，把源码也一起下载了。



其他的就跟以前差不多了。

结束日期：2018-6-14