# MAVEN

## 环境搭建

### 下载地址

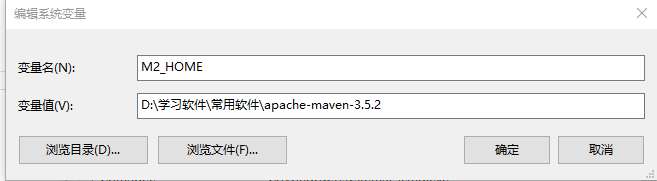
<http://maven.apache.org>

### 配置环境变量

打开环境变量，新建M2\_HOME并配置PATH

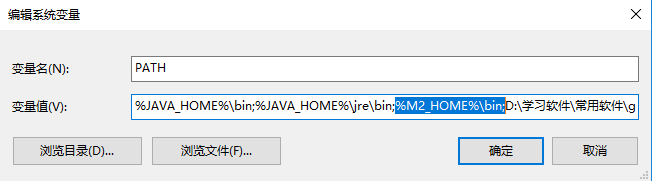
M2\_HOME (找到maven的安装目录)

例如:D:\学习软件\常用软件\apache-maven-3.5.2



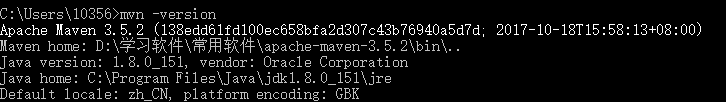
PATH

%M2\_HOME%\bin;



### 测试MAVEN安装成功

打开命令行cmd，输入命令: mvn -version



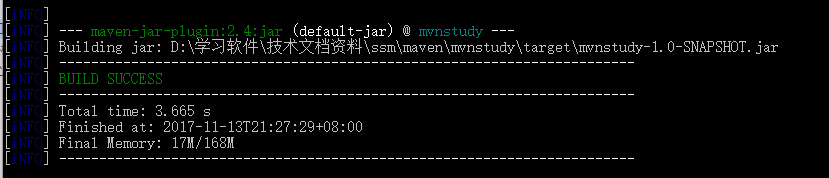
## 使用命令行工具构建Maven项目

### archetype

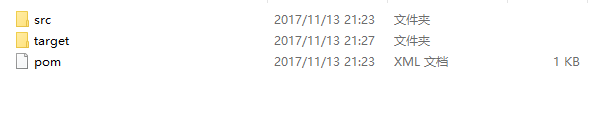
**mvn archetype:generate**

### 打包并构建项目

**mvn package**



Mvnstudy根目录下生成target目录，并生成了mvnstudy-1.0-SNAPSHOT.jar文件。



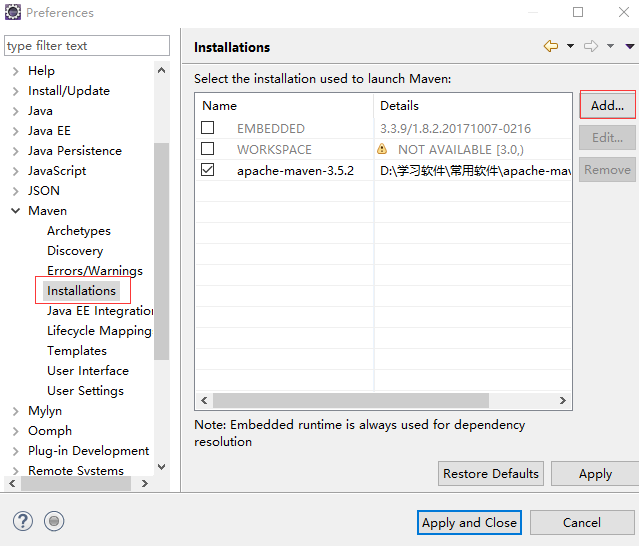


### 运行jar文件

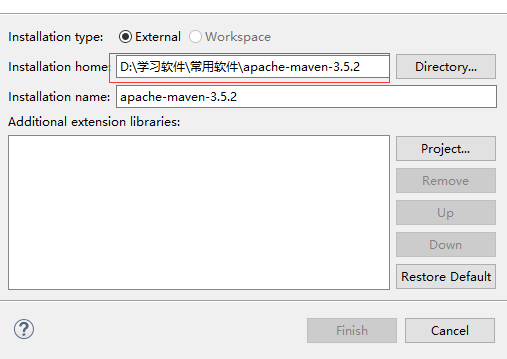
java -cp target/mvnstudy-1.0-SNAPSHOT.jar com.neusoft.App

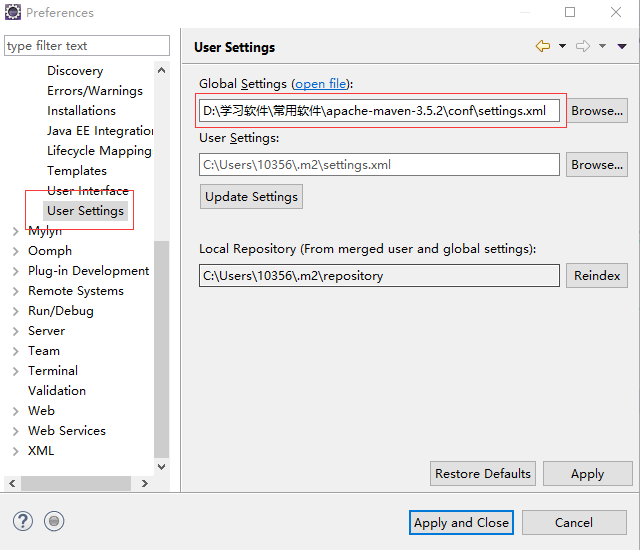
## 使用Eclipse构建Maven项目

### Eclipse中配置Maven

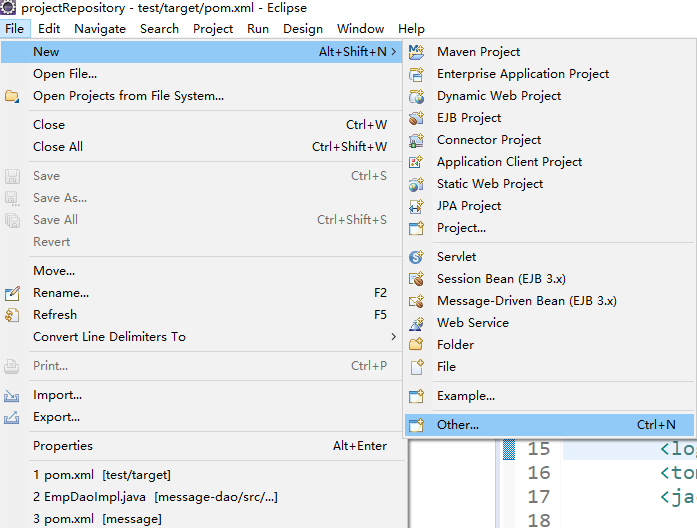


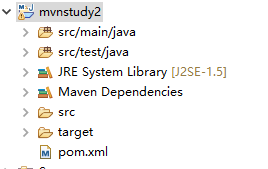
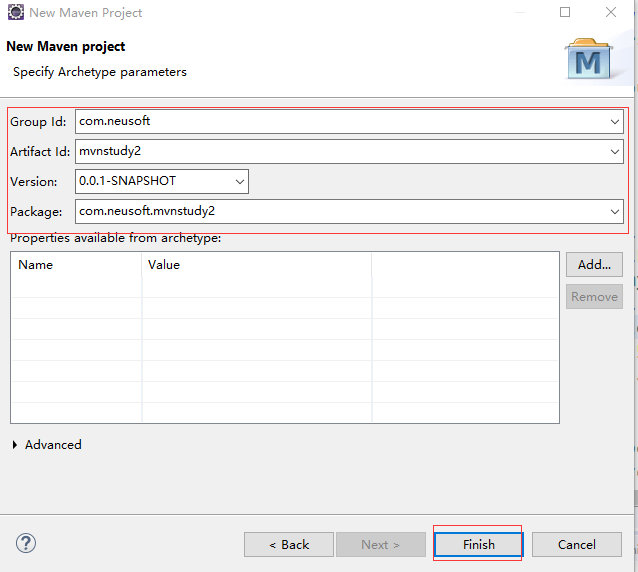
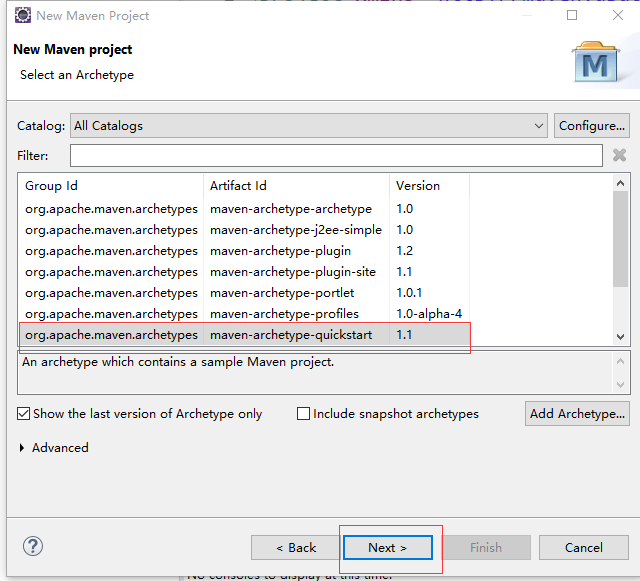
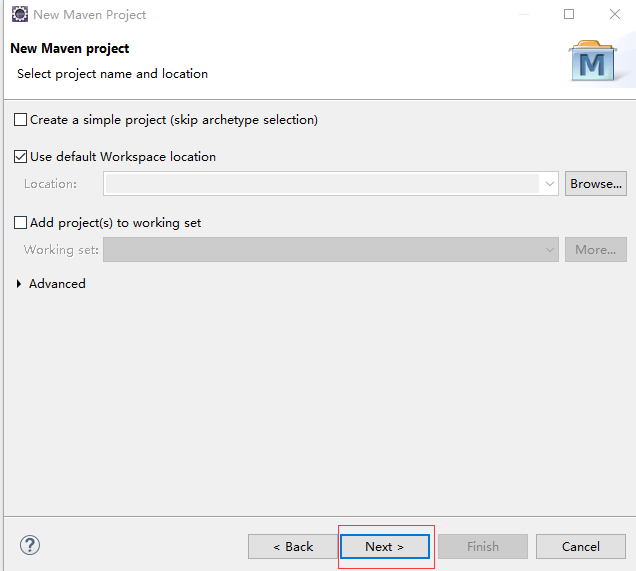
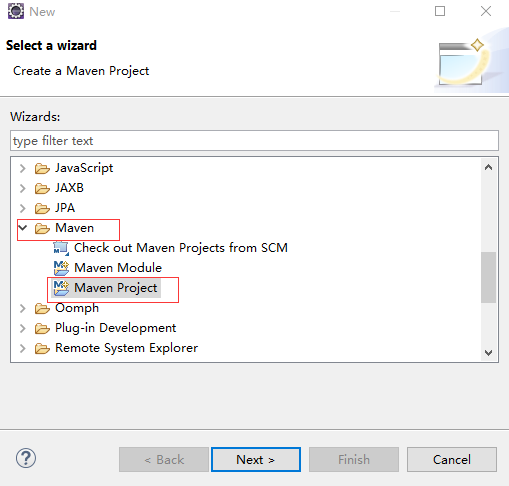
Installatin home:maven安装根目录





### 创建Maven项目

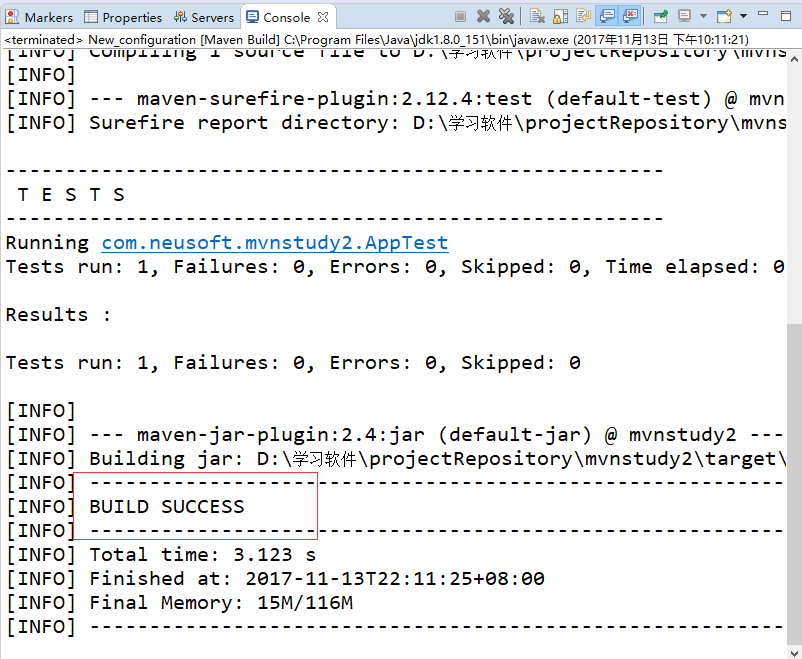
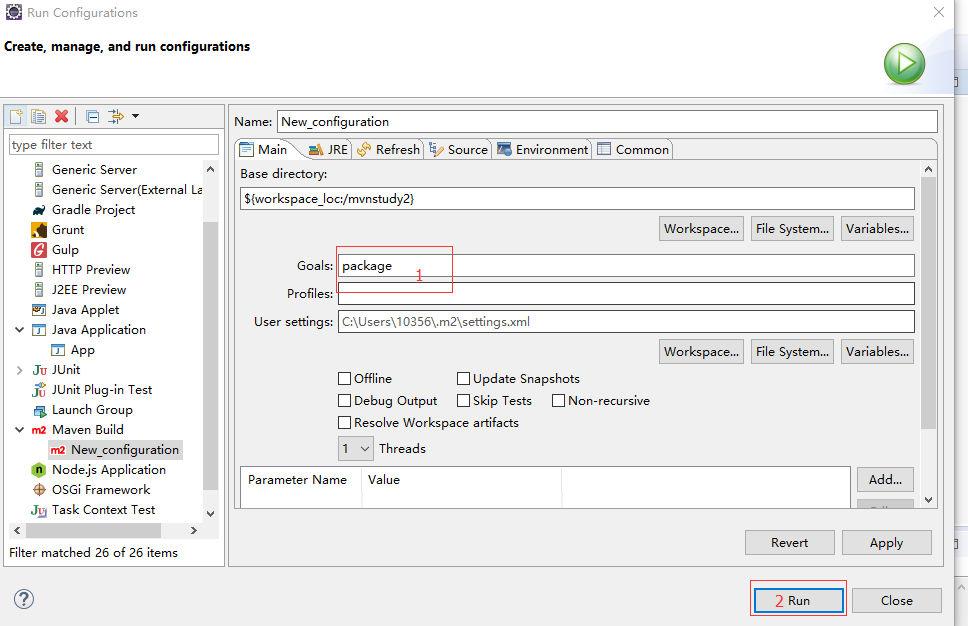
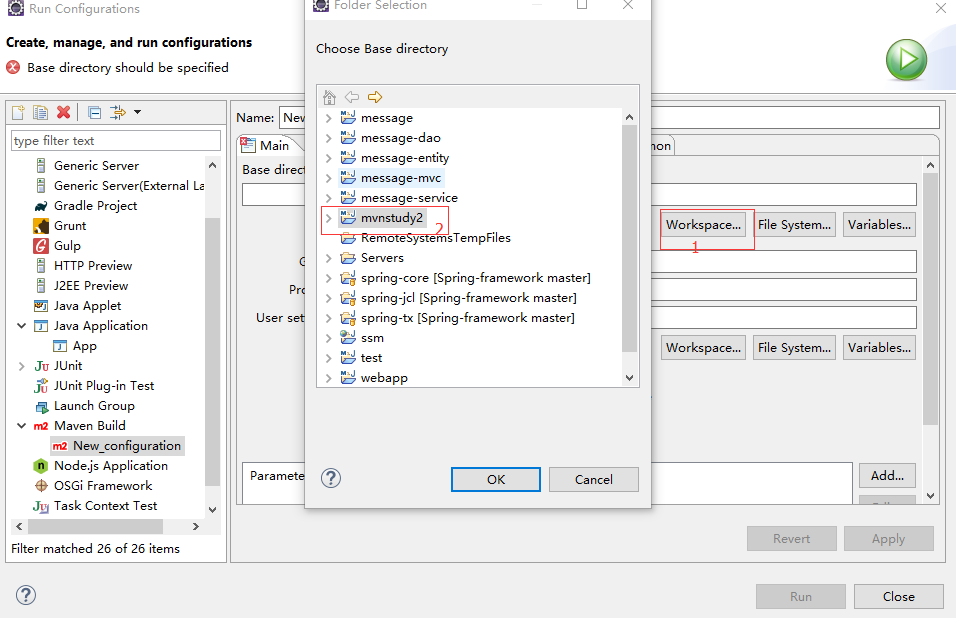
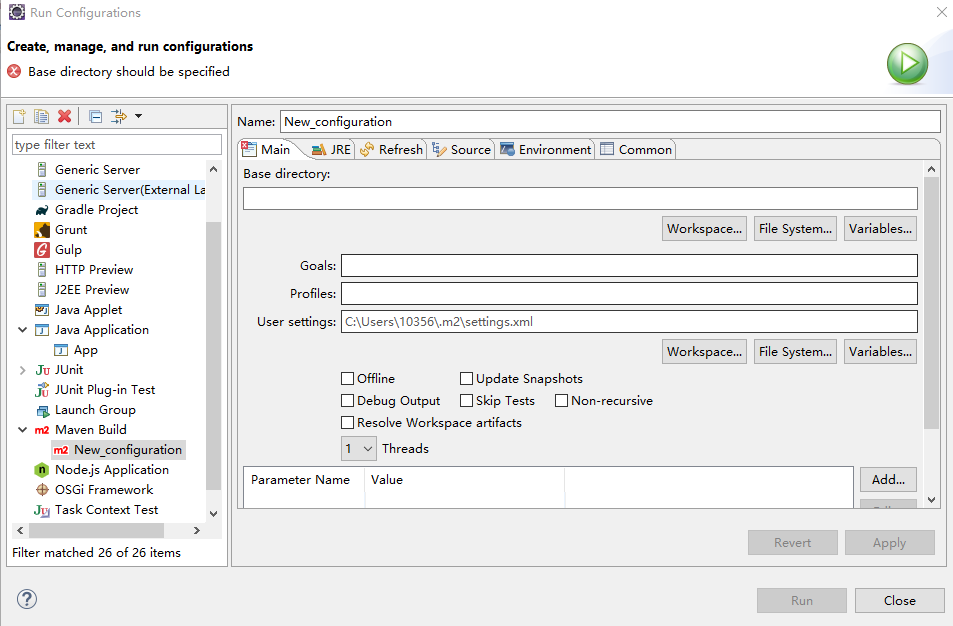
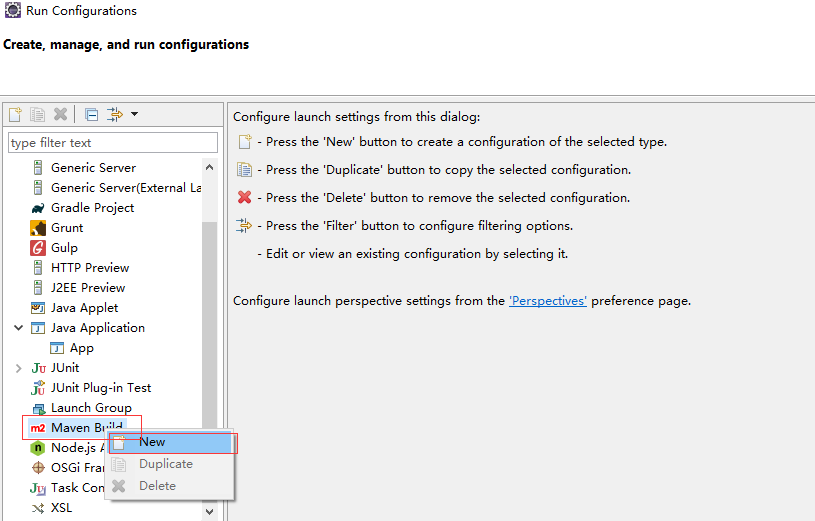
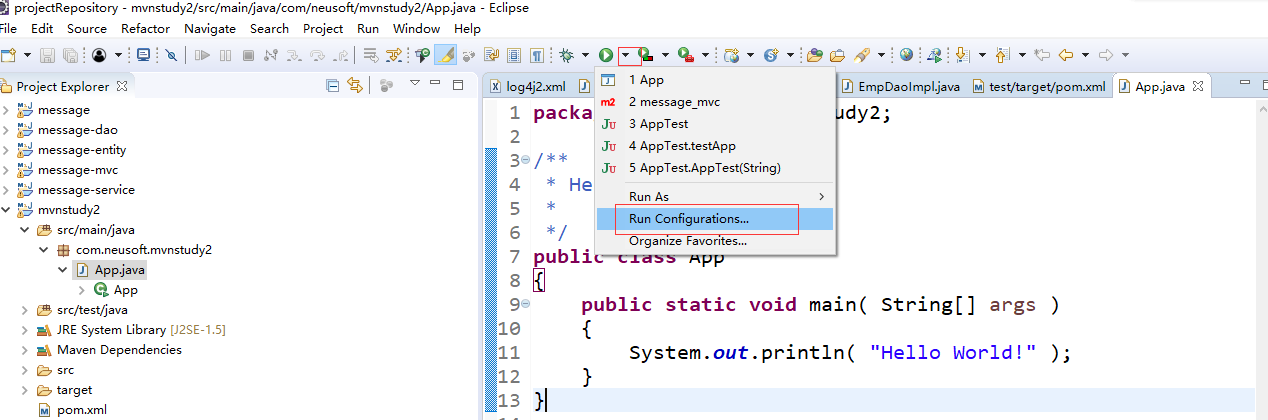




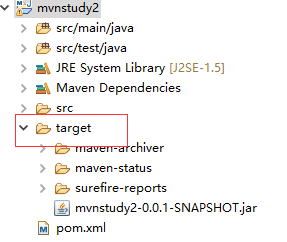
App.java

右键 run as🡪 javaApplication

### Eclipse中使用Maven打包项目



**刷新项目，target目录下生成了jar包。**



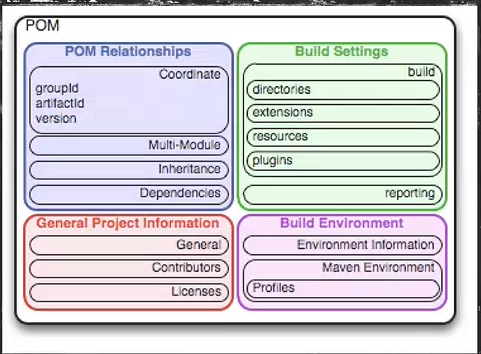
### Eclipse导入Maven项目

Import🡪maven

## MAVEN核心概念

### POM概述

POM是Project Object Model的缩写。项目的属性、依赖、构建配置这些信息都被抽象到对象模型里边。



### POM组成

#### 1. 基本信息

|  |  |
| --- | --- |
| modelVersion | Maven模块版本，目前我们一般都取值4.0.0 |
| groupId | 整个系统的名称。 |
| artifactId | 子模块名称。 |
| packaging | 打包类型，可取值：jar,war、pom、ear |

#### 2. dependencies

依赖关系。实际上pom之间存在好三种关系：继承、依赖、聚合。

|  |  |
| --- | --- |
| groupId | 依赖项的groupId |
| artifactId | 依赖项的artifactId |
| version | 依赖项的版本 |
| scope | 依赖项的适用范围：   * compile，缺省值，适用于所有阶段，会随着项目一起发布。 * provided，类似compile，期望JDK、容器或使用者会提供这个依赖。如servlet.jar。 * runtime，只在运行时使用，如JDBC驱动，适用运行和测试阶段。 * test，只在测试时使用，用于编译和运行测试代码。不会随项目发布。 * system，类似provided，需要显式提供包含依赖的jar，Maven不会在Repository中查找它。   之前例子里的junit就只用在了test中。 |
| exclusions | 排除项目中的依赖冲突时使用。 |

##### 2.1 关于排除依赖冲突

我们可能经常会遇到这样一个问题：我们的项目有两个依赖项：A & B，而且A和B同时依赖了C，但不是同一个版本。那么我们怎么办呢？

###### 2.1.1 添加检查插件

1 <reporting>

2 <plugins>

3 <plugin>

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

5 <artifactId>maven-project-info-reports-plugin</artifactId>

6 </plugin>

7 </plugins>

8 </reporting>

然后运行：mvn project-info-reports:dependencies，来查看依赖项报告。

##### 2.1.2 去除依赖项

如果我们需要在某个dependency中去除某个依赖项，直接这样即可：

1 <!-- Struts2 -->

2 <dependency>

3 <groupId>org.apache.struts</groupId>

4 <artifactId>struts2-core</artifactId>

5 <version>${struts.version}</version>

6 <exclusions>

7 <exclusion>

8 <groupId>org.freemarker</groupId>

9 <artifactId>freemarker</artifactId>

10 </exclusion>

11 </exclusions>

12 </dependency>

#### 3. 继承

<parent>

3 <groupId>com.thoughtworks.xstream</groupId>

4 <artifactId>xstream-parent</artifactId>

5 <version>1.4.3</version>

6 </parent>

#### 4. 聚合

我们可以通过一个大的项目来整合各个小的模块：

<modules>

2 <module>my-app</module>

3 </modules>

#### 5. 属性

<!-- mysql -->

2 <dependency>

3 <groupId>mysql</groupId>

4 <artifactId>mysql-connector-java</artifactId>

5 <version>${mysql.version}</version>

6 </dependency>

#### 6. 构建

##### 6.1 plugin

Plugin的配置如下:

1 <pluginManagement>

2 <plugins>

3 <plugin>

4 <groupId>org.apache.tomcat.maven</groupId>

5 <artifactId>tomcat7-maven-plugin</artifactId>

6 <version>2.1</version>

7 <configuration>

8 <tomcat-url>http://localhost:8080/manager/html</tomcat-url>

9 <server>tomcat\_localtest</server>

10 </configuration>

11 </plugin>

12 </plugins>

13 </pluginManagement>

##### 6.2 resource

指定你在Build时需要的资源文件：

<resources>

2 <resource>

3 <targetPath>WEB-INF/resource</targetPath>

4 <!-- 不对文件中的表达式进行处理 -->

5 <filtering>false</filtering>

6 <directory>${basedir}/src/test/resources</directory>

7 <includes>

8 <include>include.xml</include>

9 </includes>

10 <excludes>

11 <exclude>exclude.xml</exclude>

12 </excludes>

13 </resource>

14 </resources>

### 插件与目标

**Mvn archetype:generate**

执行archetype的generate目标。生成项目结构

调用插件格式：

Pluginid:goalid

调用插件目标的两种方式：

1. 将插件目标与生命周期绑定，执行生命周期
2. 直接执行插件目标

Maven常用插件

Maven-archetype-plugins 快速生成项目骨架插件

Maven-dependency-plugin 分析项目依赖

Maven-help-plugin

Maven-resources-plugin 提供资源管理

Maven-surefire-plugin

Jetty-maven-plugin 内置jetty容器，并将项目部署到jetty容器中，检查源文件是否变更，并及时部署。

Maven-enforcer-plugin 强制遵守某些规则，当规则被破坏就报错。还可以扩展规则，对团队开发有用。

Maven官方提供的插件：

<http://maven.apache.org/plugins/index.html>

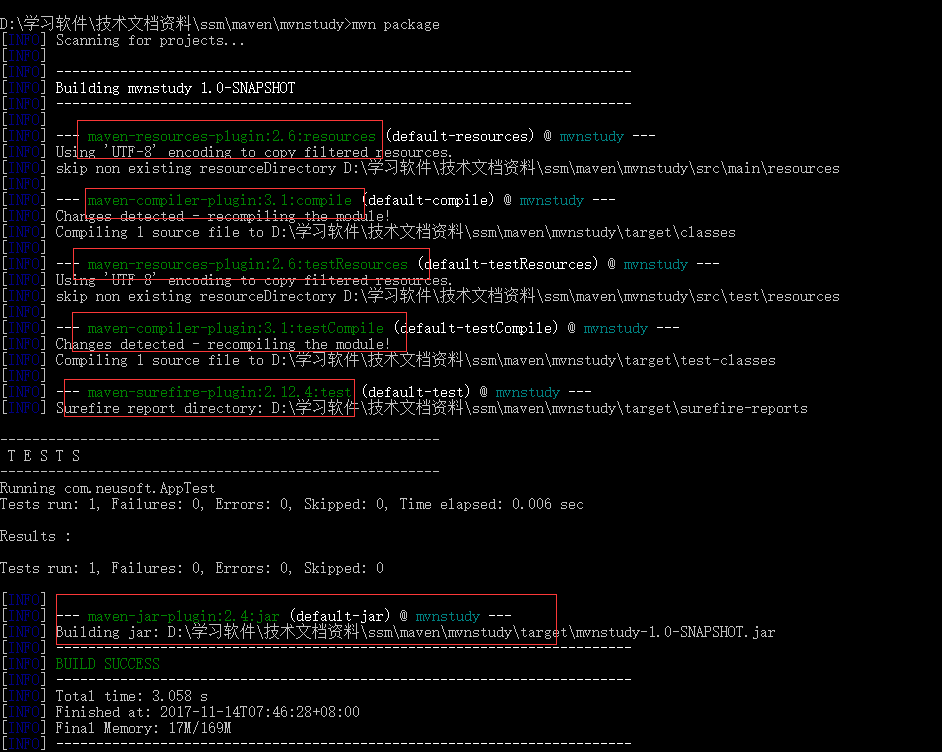
<http://mojo.codehaus.org/plugins.html>

### 项目生命周期阶段

插件目标可以绑定到生命周期上，一个生命周期可以绑定多个插件目标。



Mvn package



### Maven依赖管理

项目中用到的第三方类库，称为依赖。MAVEN通过坐标管理依赖。

在dependencies中定义依赖。

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

在实际项目当中，<dependencies/>部分是最复杂的。

项目依赖第三方jar包，第三方jar包可能依赖其他jar文件，maven中需要配置这种间接依赖到pom文件中吗，答案是否定的，Maven提供了**传递性依赖。**

#### 传递性依赖

一个复杂的项目将会包含很多依赖，也有可能包含依赖于其它构件的依赖。你不必找出所有这些依赖然后把它们写到POM文件中，你只需要加上你直接依赖的那些库，Maven会隐式的把这些库间接依赖的库也加入到你的项目中。

#### 依赖范围

Compile(编译范围，默认)

Provided（已提供范围，不会打包到war文件，由容器提供）

Runtime（运行时范围，编译时不需要运行时需要）

Test（测试范围，测试编译、测试运行时需要）

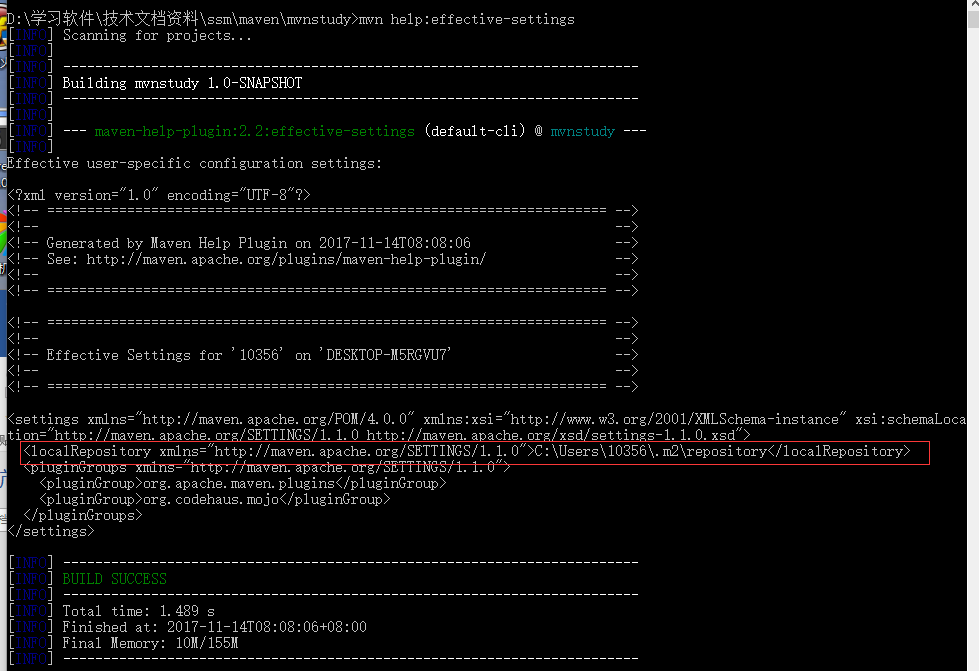
System（系统范围）

### Maven仓库

Maven仓库就是一个存放了所有依赖的仓库，这个仓库通过依赖的坐标对其进行管理。

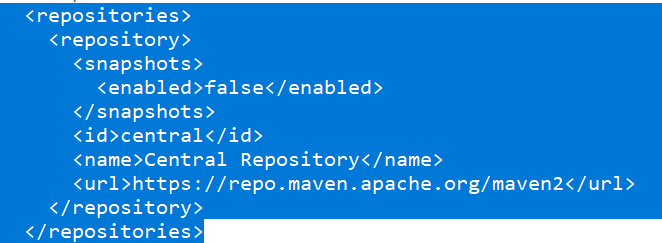
#### 本地仓库

查看本地仓库：mvn help:effective-settings



#### 远程仓库

Mvnstudy项目为例，打开effective pom文件，找到远程仓库配置，如下图：



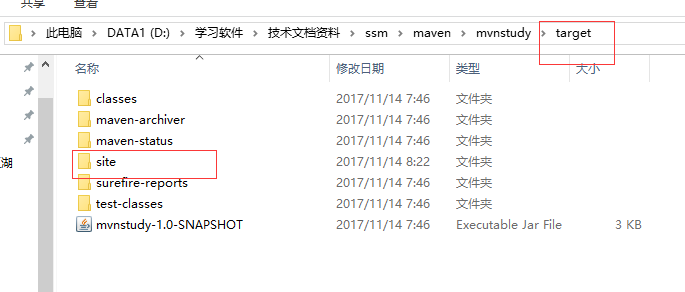
配置新的远程仓库：

在pom.xml中添加repositories标签，下图配置了开源中国的远程仓库。



### 项目站点报告

命令：mvn site



添加项目信息：

