Maven

一、Maven简介

官网：http://maven.apache.org

1.1 软件是一个工程

我们在日常生活常能听到工程这个词，像桥梁工程、道路工程、南水北调工程等等。

工程说简单点就是各个行业的从业人员通过总结规律或者方法，以最短的时间和人力、物力来做出高效可靠的东西。我们也就能理解桥梁工程，其实就是人们通过经验的总结和各种研究得出来的、用来修建桥梁时所采用的高效的方法，当然这种方法是可复用的。我们将这种作工程的思想应用到软件上，于是就产生了一软件工程。

软件工程：**为了能够实现软件的流水线式生产，在设计和构建软件时能够有一种规范和工程化的方法，人们 便提出了软件工程概念。**

上面的内容做个日常生活的类比，做道菜就是一个工程。今天心情好，想吃红烧肉，自动动手做：

1. 想买什么猪的那个位置的肉，黑猪肉，土猪肉等，使用冰糖还是绵白糖，用什么牌子的酱油等；
2. 到菜市场购买各种原料；
3. 准备材料，洗肉，切块，花椒，大料等；
4. 开始做了，肉下锅炖煮，不同时间加入花椒，大料，冰糖，酱油；
5. 炖煮一段时间后，看肉是否熟透，尝试口味，是不是咸了等等；
6. 做好了，开始吃了；
7. 需要刷碗，倒垃圾。

这些工作从头做到尾步骤非常繁琐，每个步骤都是费时费力的。所以才出现净菜，半成品菜，拿回家直接做可以了，只做6，7步骤就可以。

软件开发需要编译代码—>开发人员自己测试代码—>把代码打包—>部署项目到测试服务器—>测试人员测试 功能—>测试测试出 bug

开发人员需要修改bug—>开发人员自己测试代码—>把代码打包—>部署项目到测试服务器—>测试人员测试功 能—>直到符合功能要求。

上述过程需要重复多次，大型项目中构建项目比较复杂，有很多的配置文件，jar 文件，多个子项目等等。都 用人力完成费时费力，效率比较低。maven 可以让我们从上面的工作中解脱出来。

maven 是自动化构建工具。

完成一个java项目，需要做哪些工作?

1. 分析项目要做什么，知道项目有哪些组成部分。
2. 设计项目，通过哪些步骤，使用哪些技术。需要多少人， 多长的时间。
3. 组建团队，招人， 购置设备，服务器， 软件， 笔记本。
4. 开发人员写代码。 开发人员需要测试自己写代码。 重复多次的工作。
5. 测试人员，测试项目功能是否符合要求。

   测试开发人员提交代码----如果测试有问题----需要开发人员修改----在提交代码给测试
   ----测试人员在测试代码----如果还有问题-----在交给开发人员----开发人员在提交----再测试，直到测试代码通过。

1.2 传统项目开发存在的问题

- 一个项目做成一个工程，造成工程比较庞大，需要使用多模块来划分项目
- 项目中需要的数量众多的 jar 包，需要手动下载并引入，并且多个项目需要的 jar 包存在重复的问题；
- 项目中需要的 jar 包有版本兼容的问题，需要手动解决；
- 项目中需要的 jar 包又依赖其它的 jar 包，需要手动解决。

传统开发项目的问题，没有使用maven【meivn】管理的项目

1. 很多模块，模块之间有关系，手工管理关系，比较繁琐。
2. 需要很多第三方功能，需要很多jar文件，需要手工从网络中获取各个jar
3. 需要管理jar的版本，你需要的是mysql.5.1.5.jar 拿你不能给一个mysql.4.0.jar
4. 管理jar文件之间的依赖，你的项目要使用a.jar 需要使用b.jar里面的类。

   必须首先获取到b.jar才可以，然后才能使用a.jar.

a.jar需要b.jar这个关系叫做依赖，或者你的项目中要使用mysql的驱动，也可以叫做项目依赖mysql驱动。

a.class使用b.class， a依赖b类

1.3 Maven 概述

Maven 是 Apache 软件基金会组织维护的一款自动化构建工具，专注服务于 Java 平台的项目构建和依赖管理。Maven 这个单词的本意是：专家，内行。

Maven 是目前最流行的自动化构建工具，对于生产环境下多框架、多模块整合开发有重要作用，Maven 是一款在大型项目开发过程中不可或缺的重要工具。

Maven 可以整合多个项目之间的引用关系，我们可以根据业务和分层需要任意拆分一个项目；

Maven 提供规范的管理各个常用 jar 包及其各个版本，并且可以自动下载和引入项目中；

Maven 可以根据指定版本自动解决 jar 包版本兼容问题；

Maven 可以把 jar 包所依赖的其它 jar 包自动下载并引入项目。

类似自动化构建工具还有：Ant, Maven, Gradle。

需要改进项目的开发和管理，需要maven

1. maven可以管理jar文件
2. 自动下载jar和他的文档，源代码
3. 管理jar直接的依赖， a.jar需要b.jar ， maven会自动下载b.jar
4. 管理你需要的jar版本
5. 帮你编译程序，把java编译为class
6. 帮你测试你的代码是否正确。
7. 帮你打包文件，形成jar文件，或者war文件
8. 帮你部署项目

1.4 构建(build)

构建（build），是面向过程的，就是一些步骤，完成项目代码的编译，测试，运行，打包，部署等等。

maven支持的构建包括有：

1. **清理**，把之前项目编译的东西删除掉，为新的编译代码做准备。
2. **编译**，把程序源代码编译为执行代码，java-class文件
   批量的，maven可以同时把成千上百的文件编译为class；javac 不一样，javac一次编译一个文件。
3. **测试**，maven可以执行测试程序代码，验证你的功能是否正确。批量的，maven同时执行多个测试代码，同时测试很多功能。
4. **报告**， 生成测试结果的文件，测试通过没有。
5. **打包**， 把你的项目中所有的class文件，配置文件等所有资源放到一个压缩文件中。这个压缩文件就是项目的结果文件， 通常java程序，压缩文件是jar扩展名的。对于web应用，压缩文件扩展名是.war
6. **安装**， 把5中生成的文件jar，war安装到本机仓库
7. **部署**， 把程序安装好可以执行。

1.5 Maven核心概念

- POM：一个文件名称是pom.xml，pom翻译过来叫做项目对象模型。maven把一个项目当做一个模型使用。控制maven构建项目的过程，管理jar依赖。
- 约定的目录结构：maven项目的目录和文件的位置都是规定的。
- 坐标：是一个唯一的字符串，用来表示资源的。
- 依赖管理：管理你的项目可以使用jar文件
- 仓库管理：你的资源存放的位置
- 生命周期：maven工具构建项目的过程，就是生命周期。
- 插件和目标：执行maven构建的时候用的工具是插件
- 继承
- 聚合

1.6 安装Maven环境

下载地址：http://maven.apache.org/download.cgi

1. 需要从maven的官网下载maven的安装包 apache-maven-3.8.1-bin.zip
2. 解压安装包，解压到一个目录，非中文目录。

   子目录 bin：执行程序，主要是mvn.cmd

   子目录 conf :maven工具本身的配置文件 settings.xml
3. 配置环境变量

   在系统的环境变量中，指定一个MAVEN_HOME的名称， 指定它的值是maven工具安装目录，bin之前的目录

   X:\newJava\Maven\apache-maven-3.8.1-bin\apache-maven-3.8.1

   再把MAVEN_HOME加入到path之中，在所有路径之前加入 %MAVEN_HOME%\bin;
4. 验证，新的命令行中，执行 mvn -v

   注意：需要配置JAVA_HOME ，指定jdk路径

   出现如下内容，maven安装，配置正确。

```
Apache Maven 3.8.1 (05c21c65bdfed0f71a2f2ada8b84da59348c4c5d)
Maven home: X:\newJava\Maven\apache-maven-3.8.1-bin\apache-maven-3.8.1\bin..
Java version: 1.8.0_301, vendor: Oracle Corporation, runtime: X:\newJava\JDK8\jre
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"
```

二、Maven的核心概念

2.1 项目约定的目录结构

maven约定的目录结构，约定是大家都遵循的一个规则。

每一个maven项目在磁盘中都是一个文件夹（项目-Hello）

```
Hello/
    ---/src

    ------/main               #放你主程序java代码和配置文件
    ----------/java           #你的程序包和包中的java文件
    ----------/resources      #你的java程序中要使用的配置文件

    ------/test               #放测试程序代码和文件的（可以没有）
    ----------/java           #测试程序包和包中的java文件
    ----------/resources      #测试java程序中要使用的配置文件

    ---/pom.xml               #maven的核心文件（maven项目必须有）
```

第一个 maven 工程

按照如下步骤，实现第一个 maven 项目，以 maven 推荐的约定方式创建目录，类文件。

1. 某个目录中创建文件夹 Hello
2. 在 Hello 中创建子目录 src
3. 拷贝 pom.xml 到 Hello 目录和 src 是同级放置的。
4. 进入 src 目录，创建 main，test目录
5. 进入 main 目录，创建 java，resources 目录。 进入 java 目录，创建目录 com/bjpowernode/
6. 在 com/bjpowernode/目录下创建 HelloMaven.java 文件

```java
package com.bjpowernode;
public class HelloMaven {
  public int add(int n1,int n2){
    return n1+n2;
  }

  public static void main(String args[]){
      HelloMaven hello = new HelloMaven();
      int res = hello.add(10,20);
      System.out.println("10+20="+res);
  }
}
```

7. 进入到 Hello 所在目录，执行 mvn compile
8. 进入到 target/classes 目录执行 java com.bjpowernode.HelloMaven

疑问：mvn compile 编译 src/main 目录下的所有java文件的。

1. 为什么要下载

   maven工具执行的操作需要很多插件（java类--jar文件）完成的
2. 下载什么东西了

   jar文件--叫做插件--插件是完成某些功能
3. 下载的东西存放到哪里了。

   默认仓库（本机仓库）：C:\Users\（登录操作系统的用户名）Administrator\ .m2\repository

执行mvn compile，结果是在项目的根目录下生成target目录（结果目录）

maven编译的java程序，最后的class文件都放在target目录中

设置本机存放资源的目录位置(设置本机仓库)：

1. 修改maven的配置文件，maven安装目录/conf/settings.xml；需先备份 settings.xml
2. 修改 指定你的目录（不要使用中文目录）

   X:\newJava\Maven\maven_repository

2.2 仓库

1. 仓库的概念

   现在我们对maven工程有一个大概的认识了，那现在思考一个问题，maven怎么就这么神奇，我们写完的工程交给他之后，他就能够自动帮我们管理，我们依赖的jar包它从哪儿获取呢？有同学说已经安装了，在它的安装包里啊，大家可以看一下maven下载下来才8M，我们需要的jar包有时候都几百兆甚至几个G，它从哪儿弄去呢？其实，maven有仓库的概念。在Maven中，任何一个依赖、插件或者项目构建的输出，都可以称之为构建。Maven核心程序仅仅定义了自动化构建项目的生命周期，但具体的构建工作是由特定的构件完成的。而且为了提高构建 的效率和构件复用，maven把所有的构件统一存储在某一个位置，这个位置就叫做仓库。

2. 仓库存什么

   仓库是存放东西的，Maven 仓库的是：

   - Maven 的插件，插件也是一些 jar，这些 jar 可以完成一定的功能。
   - 我们自己开发项目的模块
   - 第三方框架或工具的 jar 包

3. 仓库的分类

   - 本地仓库：本机当前电脑上的资源存储位置，为本机上所有 Maven工程提供服务
   - 远程仓库：不在本机上， 通过网络才能使用。多电脑共享使用的。
     1. 中央仓库：通过Internet访问，为全世界所有 Maven工程服务。最权威的。
     2. 中央仓库的镜像：就是中央仓库的备份，架设在不同位置，欧洲，美洲，亚洲等每个洲都有若干的服务器，为中央仓库分担流量。减轻中央仓库的访问，下载的压力。所在洲的用户首先访问的是本洲的镜像服务器。
     3. 私服：在局域网环境中部署的服务器，为当前局域网范围内的所有 Maven工程服务。公司中常常使用这种方式。

4. 仓库的使用

   maven仓库的使用不需要人为参与。

   开发人员需要使用mysql驱动 ---> maven首先查本地仓库 ---> 私服 ---> 镜像 ---> 中央仓库

2.3 POM文件

即 Project Object Model 项目对象模型。Maven 把一个项目的结构和内容抽象成一个模型，在 xml 文件中进行声明，以方便进行构建和描述，pom.xml 是 Maven 的灵魂。所以，maven 环境搭建好之后，所有的学习和操作都是关于 pom.xml 的。

1. 坐标

   groupId：组织名，通常是公司或组织域名倒序+项目名

   artifactId：模块名，通常是工程名

   version：版本号

   唯一值， 在互联网中唯一标识一个项目的

   ```
   <groupId>公司域名的倒写</groupId>
   <artifactId>自定义项目名称</artifactId>
   <version>自定版本号</version>
   ```

   https://mvnrepository.com/ 搜索使用的中央仓库，使用groupId 或者 artifactId作为搜索条件

2. packaging：打包后压缩文件的扩展名，默认是jar，web应用是war

   packaging 可以不写，默认是jar

3. 依赖

   dependencies 和 dependency，相当于是 java代码中import

   你的项目中要使用的各种资源说明， 比我的项目要使用mysql驱动

   ```
   <dependencies>
       <!--依赖  java代码中 import -->
       <dependency>
           <groupId>mysql</groupId>
           <artifactId>mysql-connector-java</artifactId>
           <version>5.1.9</version>
       </dependency>

       <dependency>
           <groupId>junit</groupId>
           <artifactId>junit</artifactId>
           <version>4.11</version>
       </dependency>
   </dependencies>
   ```

4. properties：设置属性

```xml
<properties>
    <!--maven构建项目使用的是utf-8 ， 避免中文的乱码-->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!--编译java代码使用的jdk版本-->
    <maven.compiler.source>1.8</maven.compiler.source>
    <!--你的java项目应该运行在什么样的jdk版本-->
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

5. build：maven在进行项目的构建时，配置信息，例如指定编译java代码使用的jdk的版本等

2.4 Maven生命周期、命令、插件

1. maven的生命周期：就是maven构建项目的过程；清理，编译，测试，报告，打包，安装，部署
2. maven的命令：maven独立使用，通过命令，完成maven的生命周期的执行。 maven可以使用命令，完成项目的清理，编译，测试等等
3. maven的插件：maven命令执行时，真正完成功能的是插件，插件就是一些jar文件、一些类。

- 单元测试（测试方法）：用的是junit， junit是一个专门测试的框架（工具）。

  junit测试的内容： 测试的是类中的方法， 每一个方法都是独立测试的。方法是测试的基本单位（单元）。

  maven借助单元测试，批量的测试你类中的大量方法是否符合预期的。

- 使用步骤

1. 加入依赖，在pom.xml加入单元测试依赖

```xml
<!-- 单元测试 -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
</dependency>
```

2. 在maven项目中的src/test/java目录下，创建测试程序。

   推荐的创建类和方法的提示：

   1.测试类的名称 是Test + 你要测试的类名

   2.测试的方法名称 是：Test + 方法名称

   例如你要测试HelloMaven，创建测试类 TestHelloMaven

```java
@Test
public void testAdd(){
    //测试HelloMaven的add方法是否正确
    int res  = 20 + 10;
    //期望值 实际值
    Assert.assertEquals(30, res);
}

/*
其中testAdd叫做测试方法，它的定义规则
    1.方法是public的，必须的
    2.方法没有返回值， 必须的
    3.方法名称是自定义的，推荐是Test + 方法名称
    4.在方法的上面加入 @Test
*/
```

3. mvn compile

   编译main/java/目录下的 java 为 class 文件，同时把 class 拷贝到 target/classes 目录下面

   把 main/resources 目录下的所有文件都拷贝到 target/classes 目录下

- Maven的常用命令

1. mvn clean 清理 (会删除原来编译和测试的目录，即 target 目录，但是已经 install 到仓库里的包不会删除)
2. mvn compile 编译主程序 (会在当前目录下生成一个 target，里边存放编译主程序之后生成的字节码文件)
3. mvn test-compile 编译测试程序(会在当前目录下生成一个 target,里边存放编译测试程序之后生成的字节码文件)
4. mvn test 测试 (会生成一个目录surefire-reports，保存测试结果)
5. mvn package 打包主程序(会编译、编译测试、测试、并且按照 pom.xml 配置把主程序打包生成 jar 包或者 war 包)
6. mvn install 安装主程序(会把本工程打包，并且按照本工程的坐标保存到本地仓库中)
7. mvn deploy 部署主程序(会把本工程打包，按照本工程的坐标保存到本地库中，并且还会保存到私服仓库中。 还会自动把项目部署到 web 容器中)。

- 插件

  官网插件说明：http://maven.apache.org/plugins

  在项目根目录下执：mvn clean install

  1. clean 插件 maven-clean-plugin:2.5

     clean 阶段是独立的一个阶段，功能就是清除工程目前下的 target 目录

  2. resources 插件 maven-resources-plugin:2.6

     resource 插件的功能就是把项目需要的配置文件拷贝到指定的目当，默认是拷贝 src\main\resources 目录下的件到 classes 目录下

  3. compile 插件 maven-compiler-plugin

     compile 插件执行时先调用 resouces 插件，功能就是把 src\mainjava 源码编译成字节码生成 class 文件，并把编译好 的 class 文件输出到 target\classes 目录下

  4. test 测试插件

     单元测试所用的 compile 和 resources 插件和主代码是相同的，但执行的目标不行，目标 testCompile 和 testResources 是把 src\test\java 下的代码编译成字节码输出到 target\test-classes，同时把 src\test\resources 下的配置文件拷贝到 target\test-classes

  5. package 打包插件 maven-jar-plugin

     这个插件是把 class 文件、配置文件打成一个 jar(war 或其它格式)包

  6. deploy 发布插件 maven-install-plugin

     发布插件的功能就是把构建好的 artifact 部署到本地仓库，还有一个 deploy 插件是将构建好的 artifact 部署到远程 仓库

  7. 配置插件

```
<build>
    <!--配置插件-->
    <plugins>
         <!--配置具体的插件-->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
             <!--插件的名称-->
            <artifactId>maven-compiler-plugin</artifactId>
             <!--插件的版本-->
            <version>3.8.1</version>
            <!--配置插件的信息-->
            <configuration>
                 <!--告诉maven我们写的代码是在jdk1.8上编译的-->
                <source>1.8</source>
                 <!--我们的程序应该运行在jdk1.8环境上-->
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

三、Maven 在 IDEA 中的应用

3.1 IDEA 集成 Maven

在idea中设置maven，让idea和maven结合使用。

idea中内置了maven，一般不使用内置的， 因为用内置修改maven的设置不方便。

使用自己安装的maven，需要覆盖idea中的默认的设置。让idea指定maven安装位置等信息

配置的入口：

1. 配置当前工程的设置：

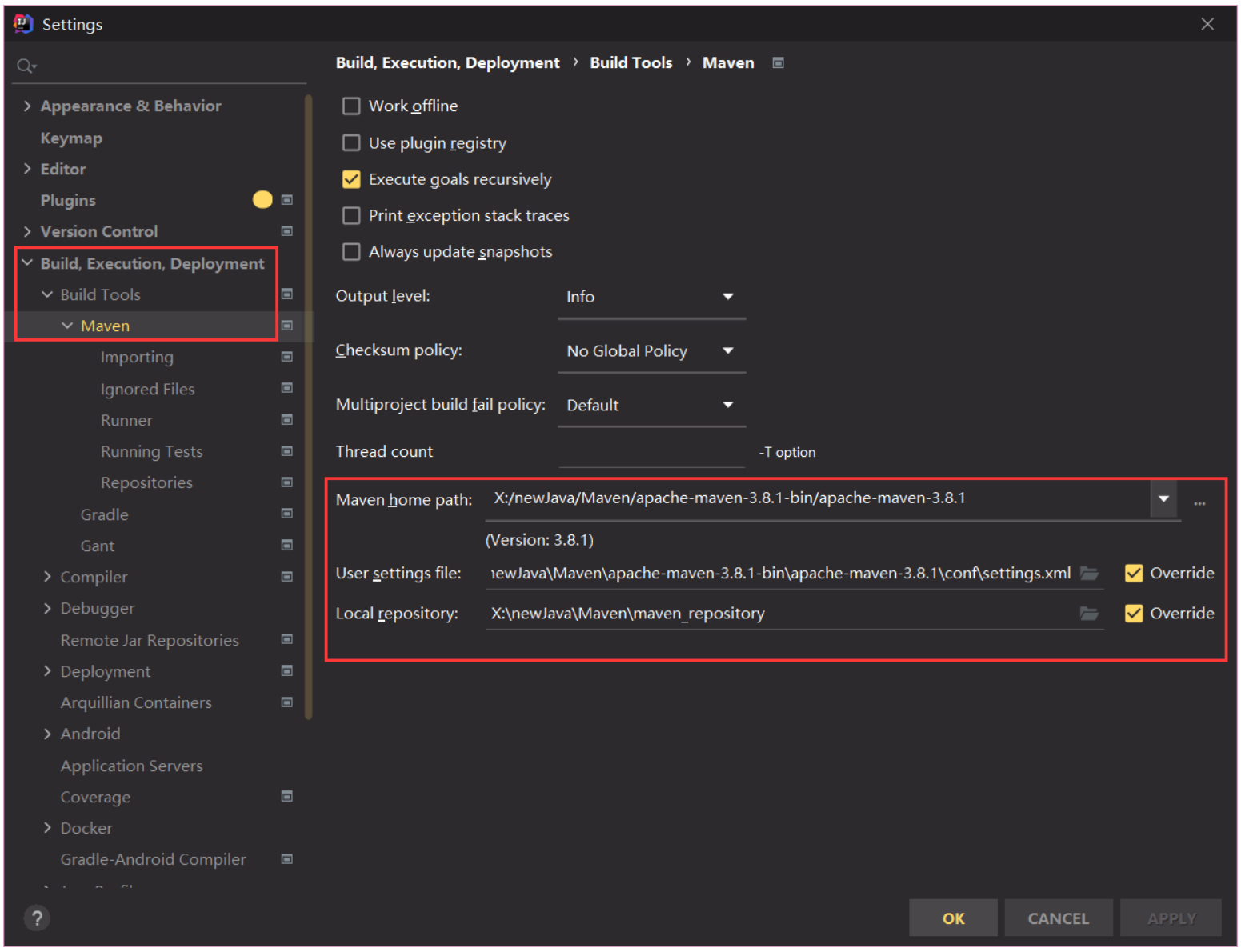
   file---> settings ---> Build, Excution,Deployment --->Build Tools
   --->Maven

   ---Maven

   Maven Home directory: maven的安装目录

   User Settings File： 就是maven安装目录conf/setting.xml配置文件

   Local Repository： 本机仓库的目录位置



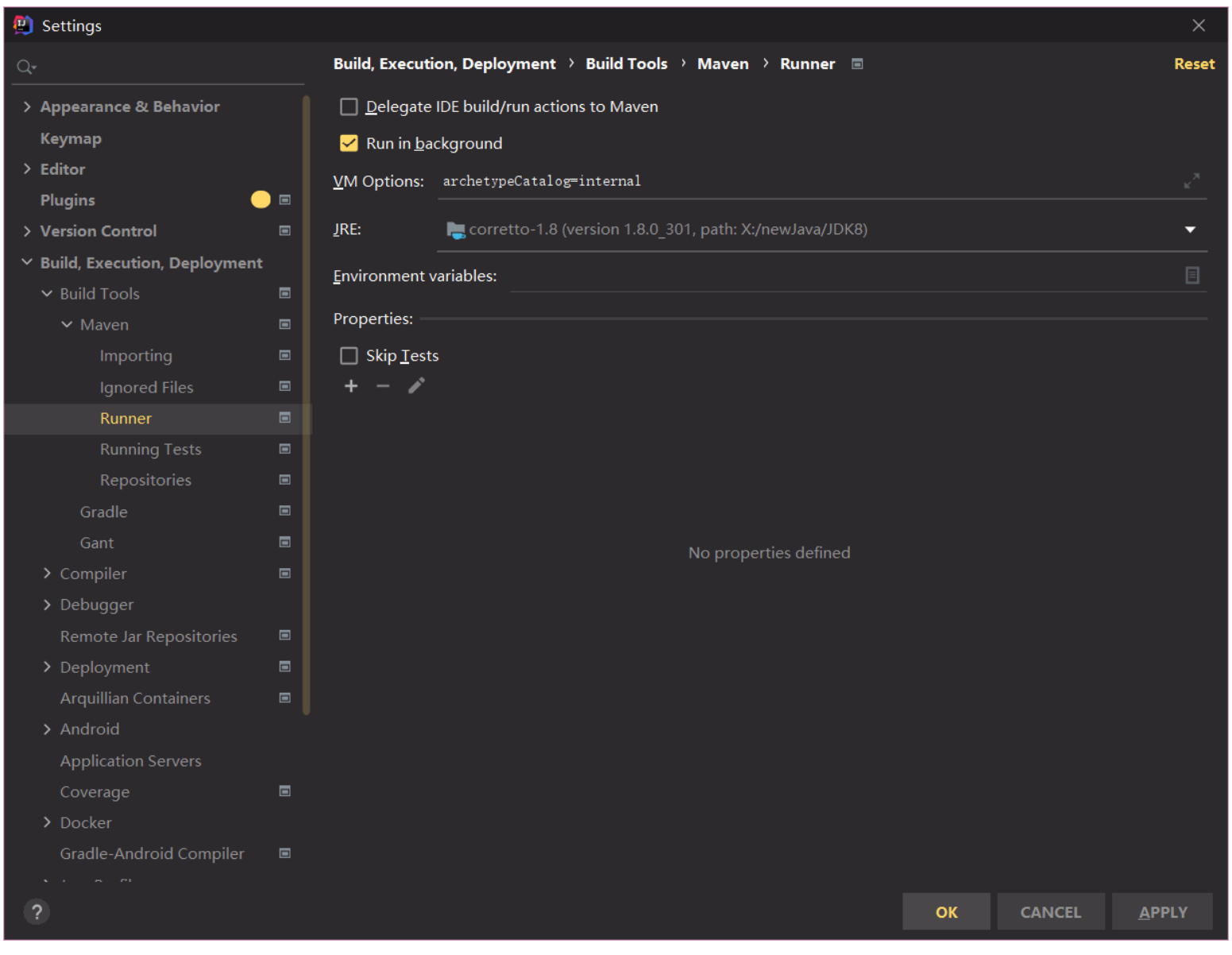
--->Build Tools--->Maven--->Runner

VM Options : archetypeCatalog=internal

JRE: 你项目的jdk

archetypeCatalog=internal , maven项目创建时，会联网下载模版文件，

比较大， 使用 archetypeCatalog=internal，不用下载， 创建maven项目速度快。



2. 配置以后新建工程的设置

   file---> other settings---> Settings for New Project

使用模版创建项目

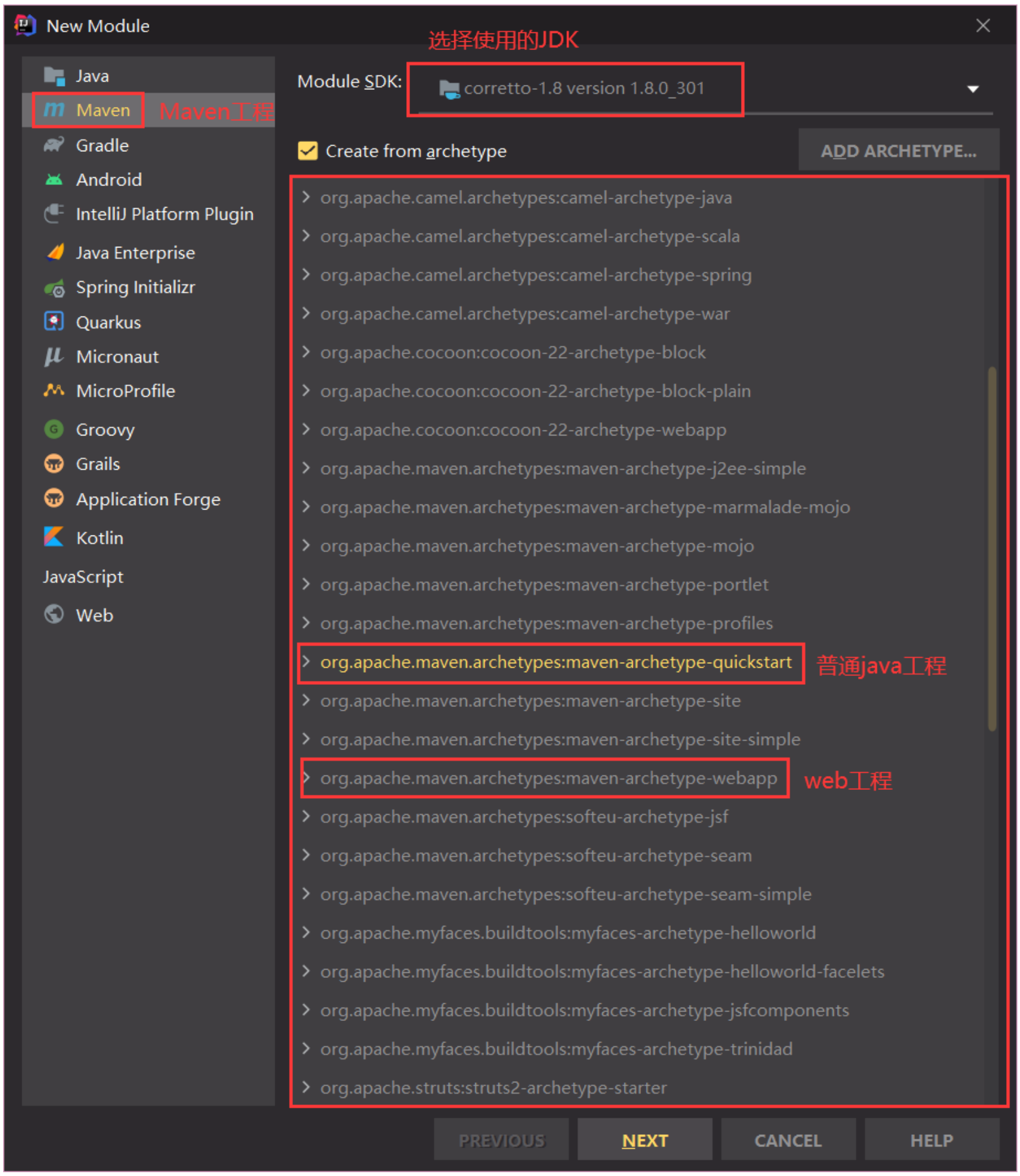
1. maven-archetype-quickstart : 普通的java项目

2. maven-archetype-webapp : web工程

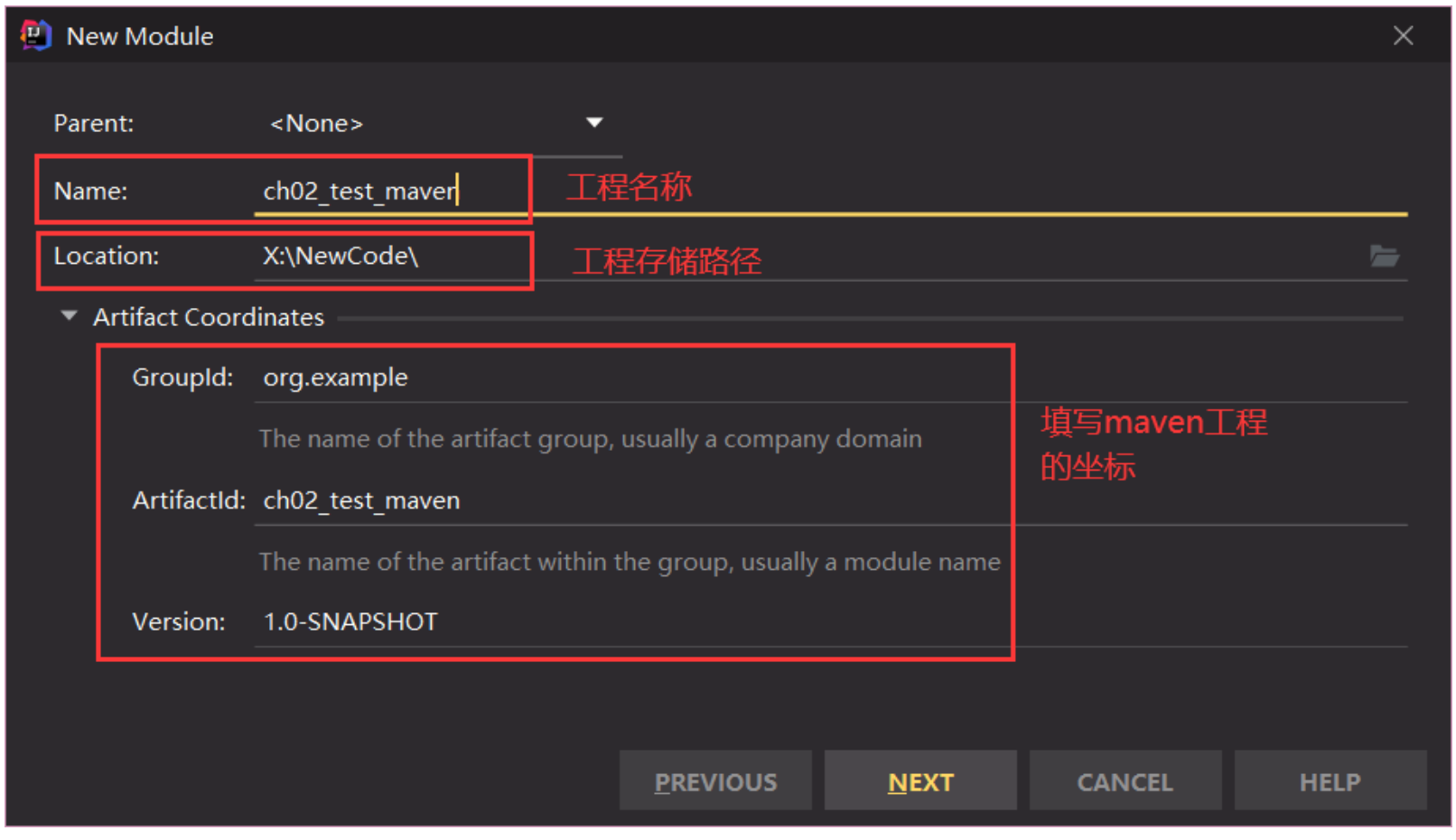
3.2 IDEA创建Maven版Java工程

1. 创建maven版java工程

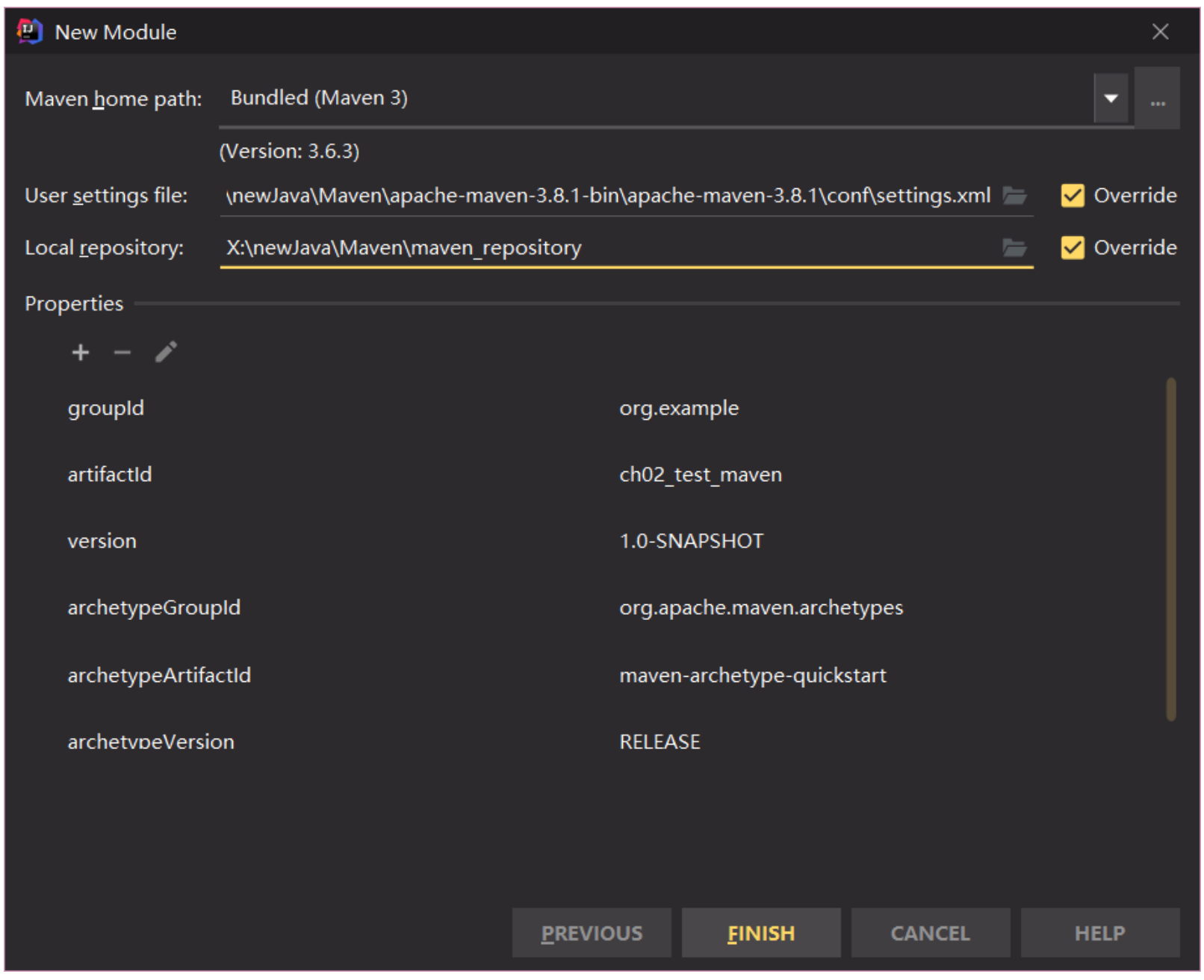
   File-->New-->Module... :



2. 填写工程名和存储路径以及 maven 工程的坐标



3. 确定maven路径，点击finish

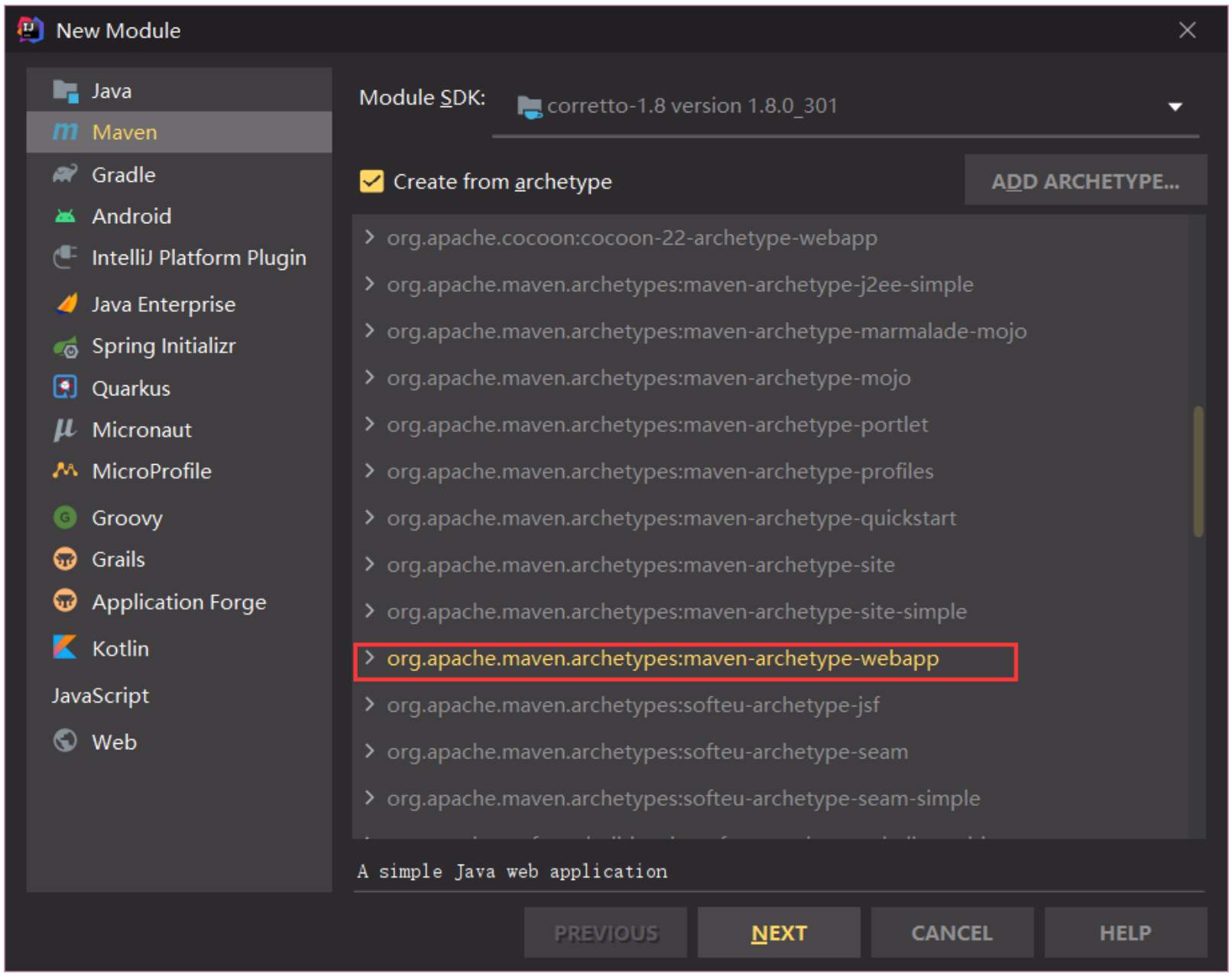


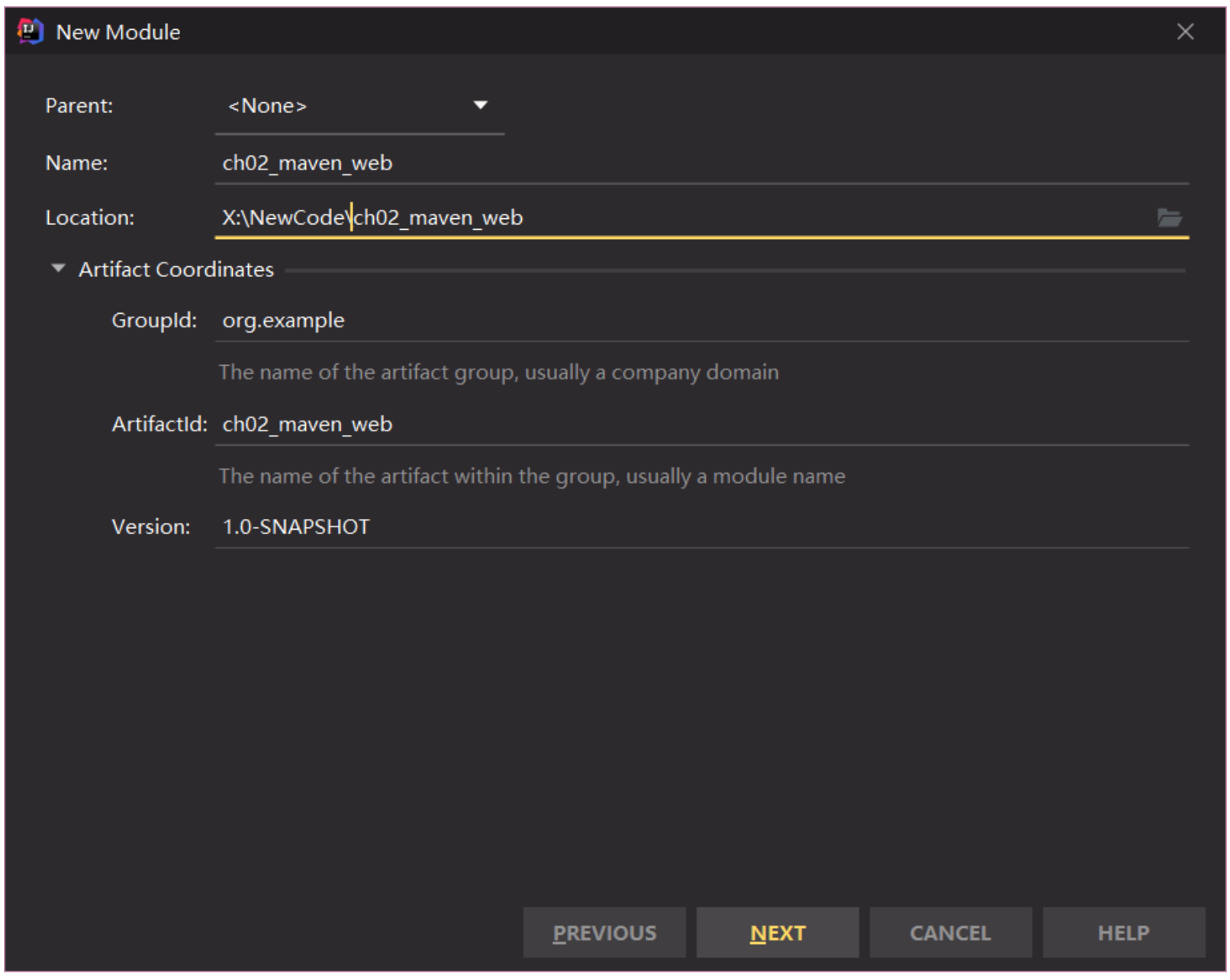
4. pom.xml

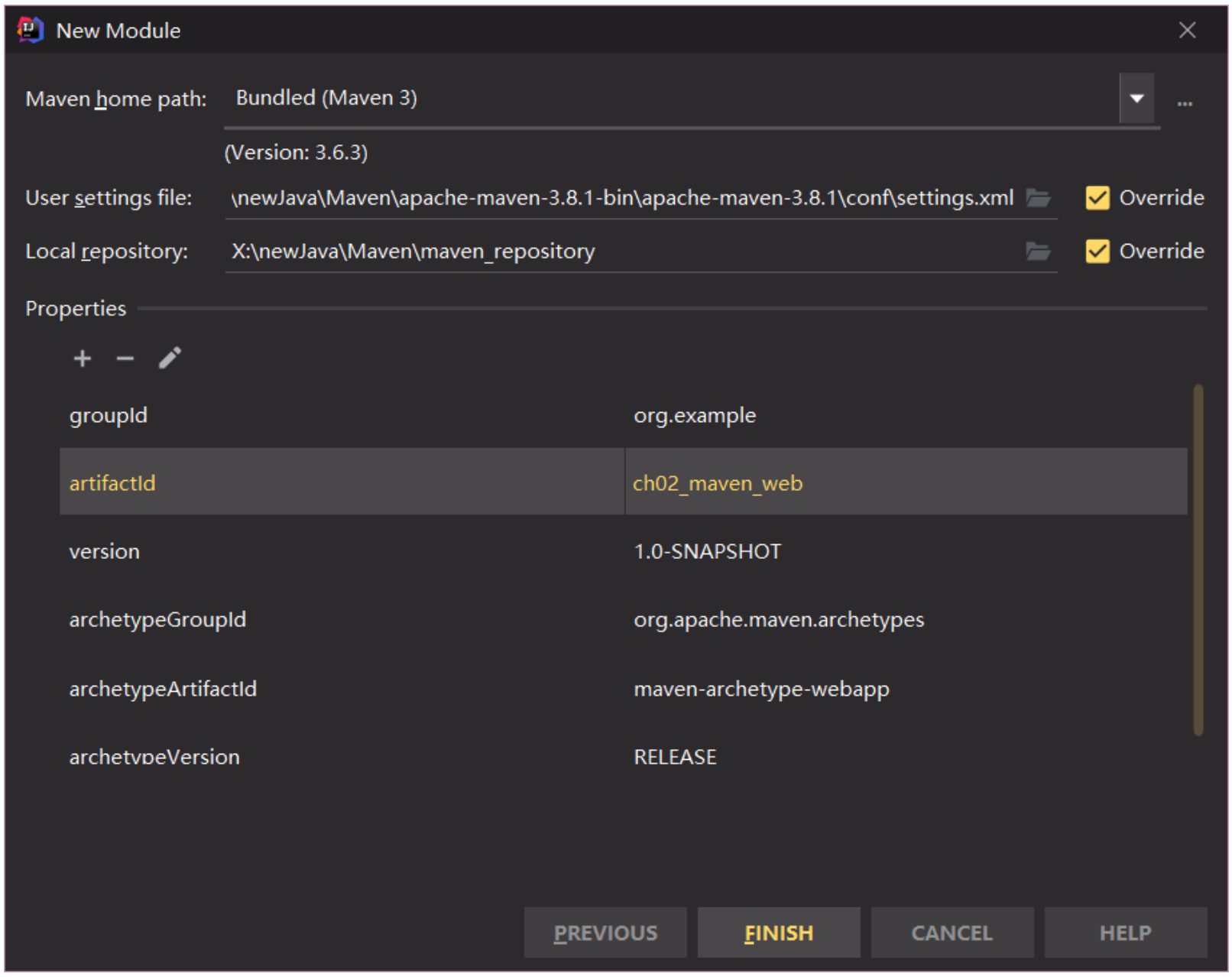
```xml
<?xml version="1.0" encoding="UTF-8"?>

<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>

  <!-- 自己项目的坐标（gav)-->
  <groupId>org.example</groupId>
  <artifactId>ch01_maven_java</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <!--maven构建项目使用的是utf-8 ， 避免中文的乱码-->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!--编译java代码使用的jdk版本-->
```

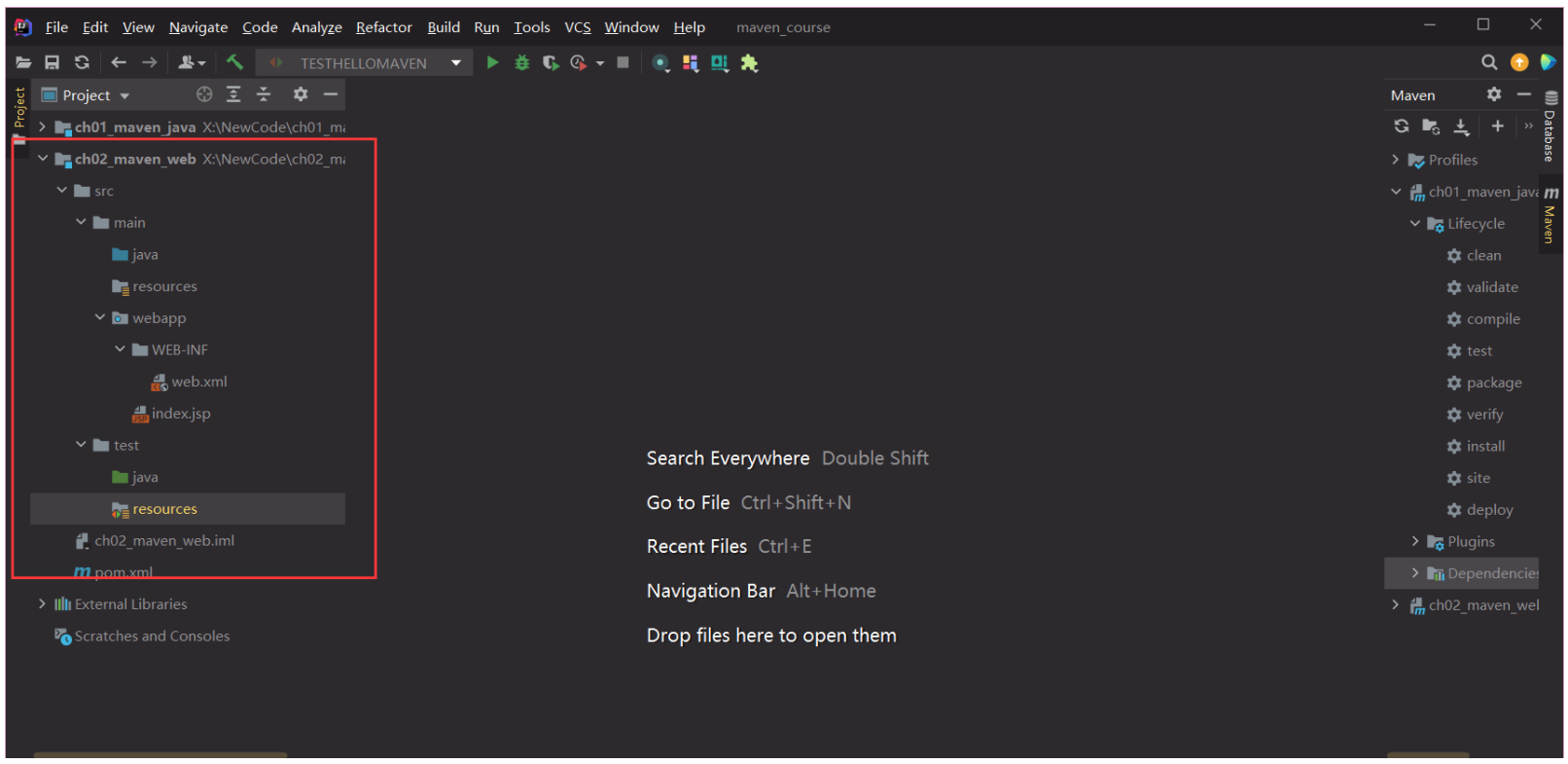
```
    <maven.compiler.source>1.8</maven.compiler.source>
    <!--你的java项目应该运行在什么样的jdk版本-->
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <!--依赖-->
  <dependencies>
    <dependency>
      <!--单元测试-->
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->
      <plugins>
        <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean_Lifecycle -->
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.1.0</version>
        </plugin>
        <!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin_bindings_for_jar_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.22.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-jar-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>2.5.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>2.8.2</version>
        </plugin>
        <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site_Lifecycle -->
        <plugin>
          <artifactId>maven-site-plugin</artifactId>
          <version>3.7.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-project-info-reports-plugin</artifactId>
          <version>3.0.0</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

3.3 IDEA创建Maven版Web工程


New Module
Java
Maven
Gradle
Android
IntelliJ Platform Plugin
Java Enterprise
Spring Initializr
Quarkus
Micronaut
MicroProfile
Groovy
Grails
Application Forge
Kotlin
JavaScript
Web
Module SDK:
corretto-1.8 version 1.8.0_301
Create from archetype
ADD ARCHETYPE...
org.apache.cocoon:cocoon-22-archetype-webapp
org.apache.maven.archetypes:maven-archetype-j2ee-simple
org.apache.maven.archetypes:maven-archetype-marmalade-mojo
org.apache.maven.archetypes:maven-archetype-mojo
org.apache.maven.archetypes:maven-archetype-portlet
org.apache.maven.archetypes:maven-archetype-profiles
org.apache.maven.archetypes:maven-archetype-quickstart
org.apache.maven.archetypes:maven-archetype-site
org.apache.maven.archetypes:maven-archetype-site-simple
org.apache.maven.archetypes:maven-archetype-webapp
org.apache.maven.archetypes:softeu-archetype-jsf
org.apache.maven.archetypes:softeu-archetype-seam
org.apache.maven.archetypes:softeu-archetype-seam-simple
A simple Java web application
PREVIOUS
NEXT
CANCEL
HELP


New Module
Parent:
<None>
Name:
ch02_maven_web
Location:
X:\NewCode\ch02_maven_web
Artifact Coordinates
GroupId:
org.example
The name of the artifact group, usually a company domain
ArtifactId:
ch02_maven_web
The name of the artifact within the group, usually a module name
Version:
1.0-SNAPSHOT
PREVIOUS
NEXT
CANCEL
HELP



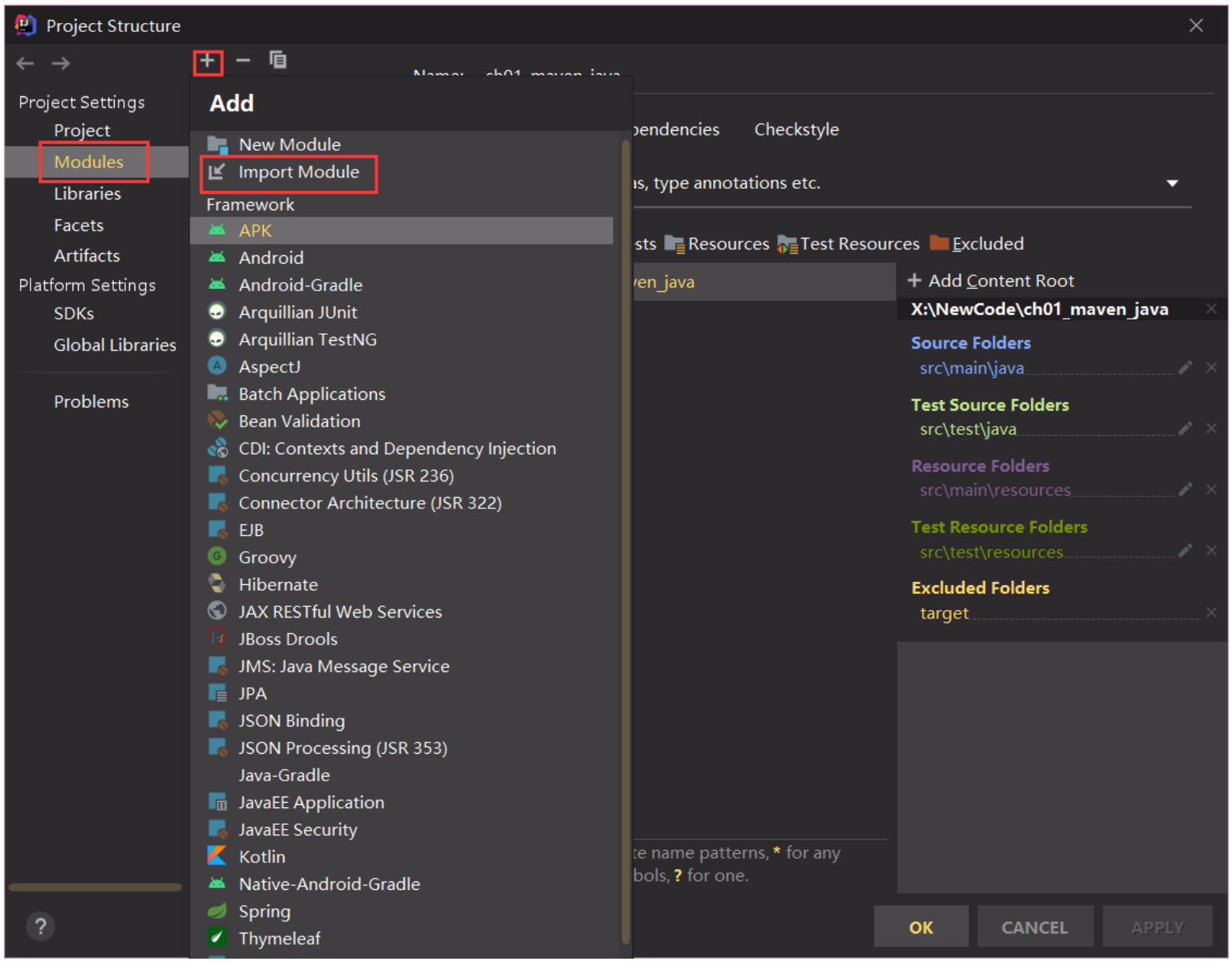



```xml
<!--加入依赖-->
<dependencies>
    <!--单元测试-->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <!--加入servlet依赖(servlet的jar)-->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
    </dependency>
    <!--jsp依赖(jsp相关的jar加入进来)-->
    <dependency>
      <groupId>javax.servlet.jsp</groupId>
      <artifactId>jsp-api</artifactId>
      <version>2.1</version>
      <scope>provided</scope>
    </dependency>
```

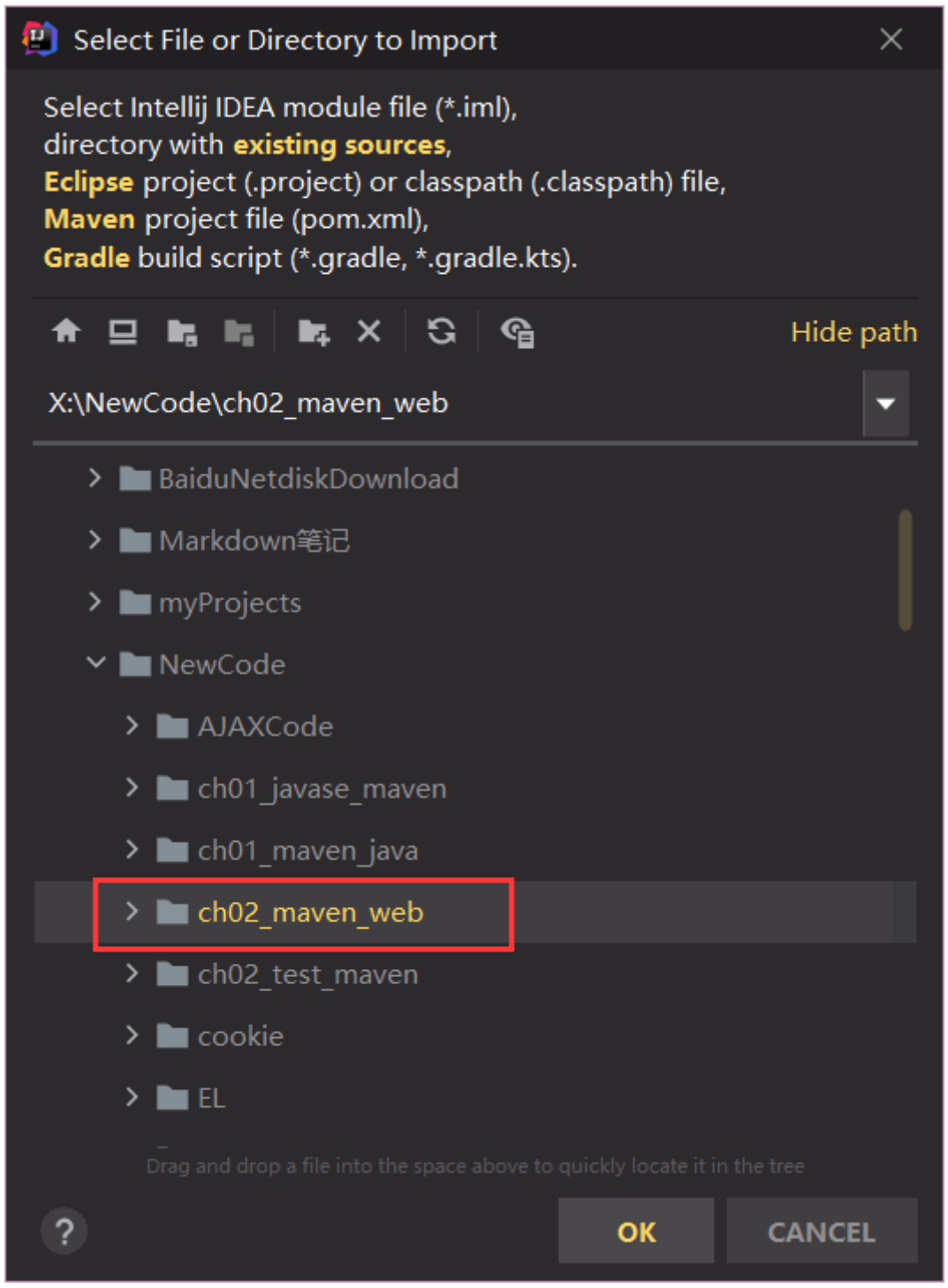
```
24  </dependencies>
```

3.4 IDEA 中导入 Maven 工程(module)

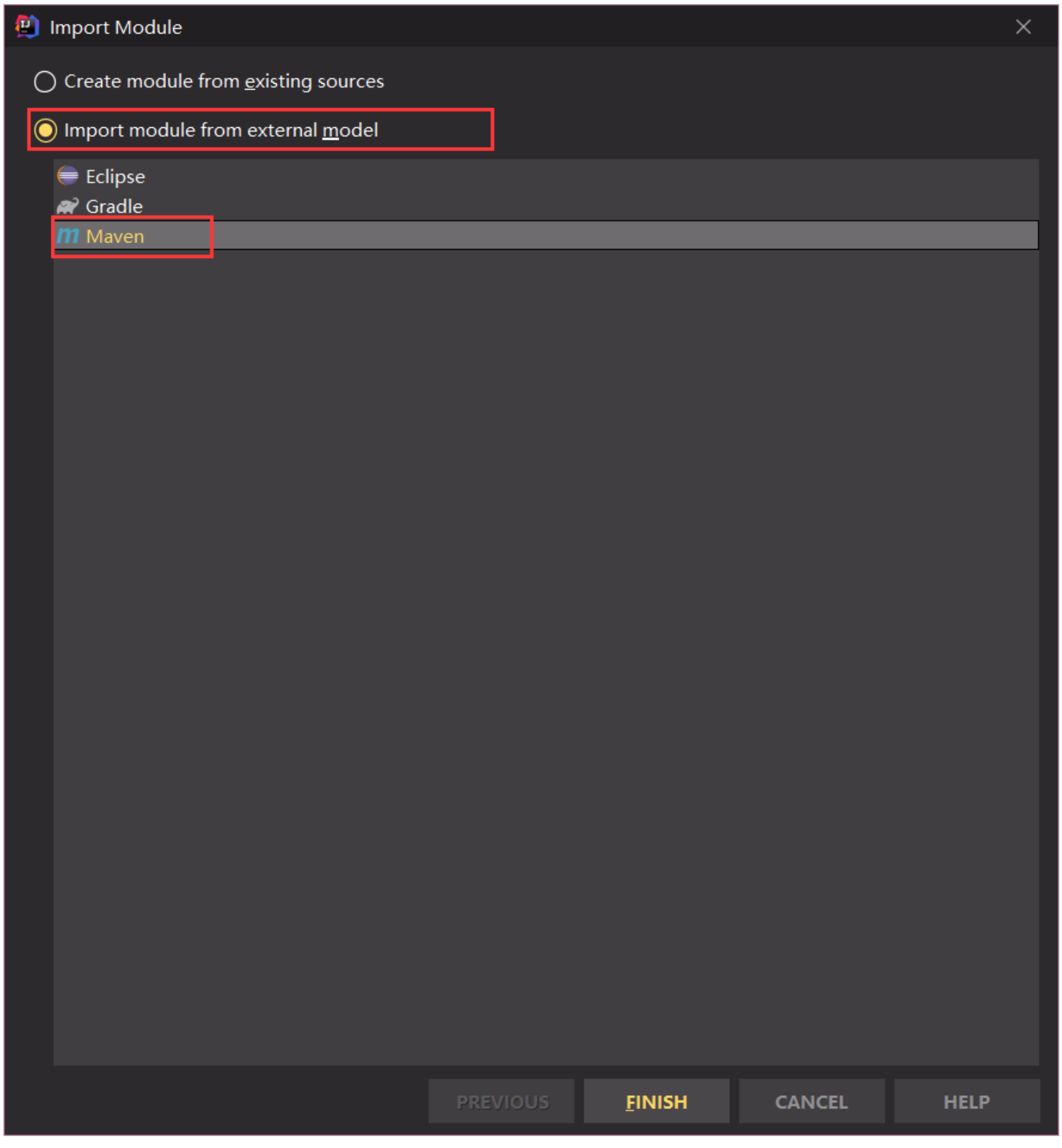
项目结构---> modules ---> + ---> import module



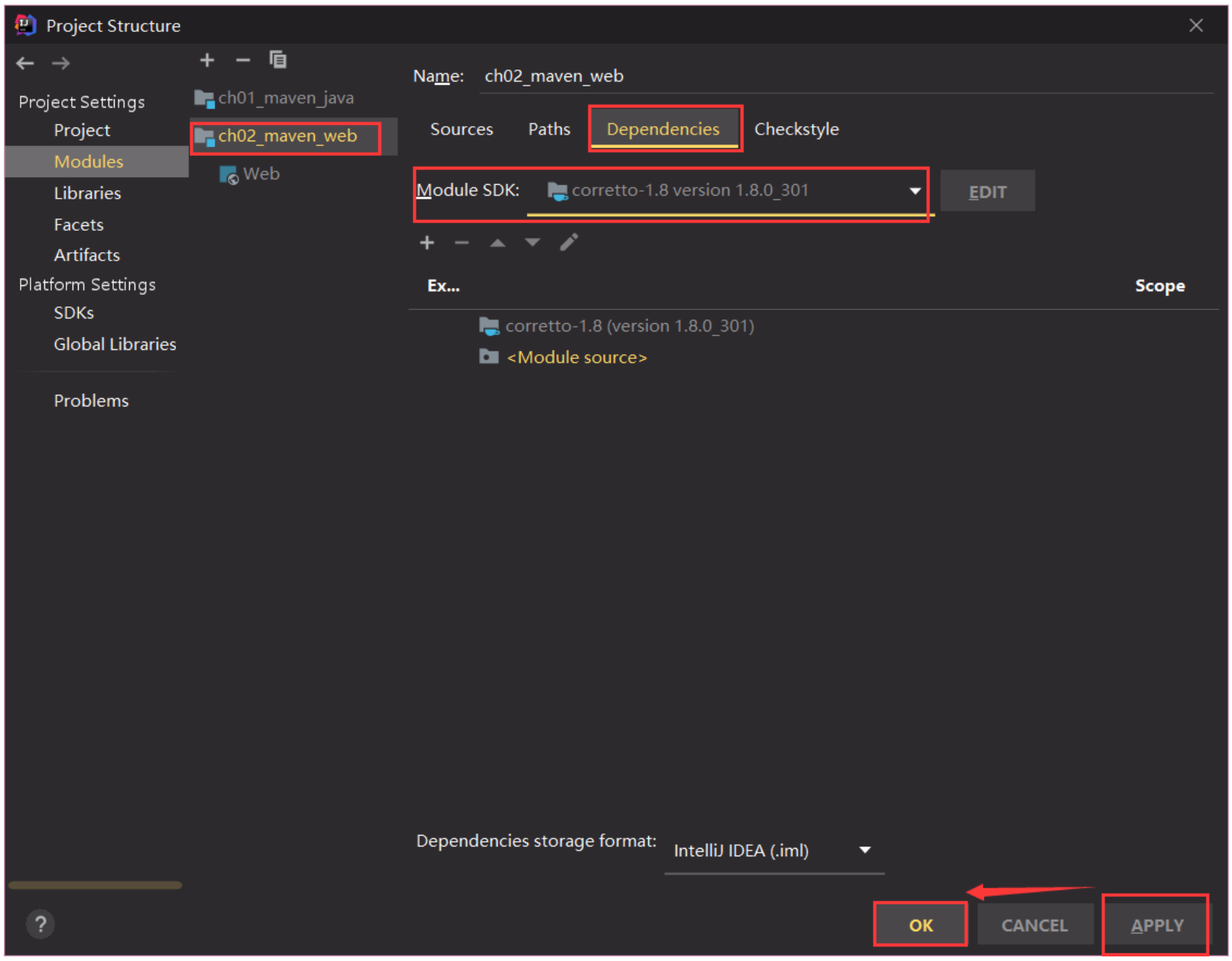
选择要导入的 Module


Select File or Directory to Import
Select Intellij IDEA module file (*.iml),
directory with existing sources,
Eclipse project (.project) or classpath (.classpath) file,
Maven project file (pom.xml),
Gradle build script (*.gradle, *.gradle.kts).
Hide path
X:\NewCode\ch02_maven_web
BaiduNetdiskDownload
Markdown笔记
myProjects
NewCode
AJAXCode
ch01_javase_maven
ch01_maven_java
ch02_maven_web
ch02_test_maven
cookie
EL
Drag and drop a file into the space above to quickly locate it in the tree
OK
CANCEL

选择导入方式


Import Module
Create module from existing sources
Import module from external model
Eclipse
Gradle
Maven
PREVIOUS
FINISH
CANCEL
HELP

设置依赖管理



3.5 依赖管理

依赖范围，使用scope表示的

scope 的值有 compile、test、provided，默认是compile；

- compile：默认，在构建的编译，测试，打包，部署阶段需要
- test：只在测试阶段需要
- provided：在部署时，有服务器提供，根本就不需要自带

scope：表示依赖使用的范围，也就是在maven构建项目的那些阶段中起作用；

maven构建项目 编译，测试，打包，安装，部署 过程（阶段）

	compile	test	provided
对主程序是否有效	是	否	是
对测试程序是否有效	是	是	是
是否参与打包	是	否	否
是否参与部署	是	否	否

```
    junit的依赖范围是 test
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>a</groupId>
      <artifactId>b</artifactId>   b.jar
      <version>4.11</version>
    </dependency>

     <dependency>
      <groupId>javax.servlet</groupId>
```

```
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>          servlet.jar
      <scope>provided</scope>  提供者
    </dependency>
```

你在写项目的中的用到的所有依赖（jar ），必须在本地仓库中有；没有必须通过maven下载，包括provided的都必须下载。

你在servlet需要继承HttpServlet(provided)，你使用的HttpServlet是maven仓库中的。

当你的写好的程序，放到 tomat 服务器中运行时，此时你的程序中不包含servlet的jar
因为 tomcat 提供了 servlet 的.jar

3.6 maven常用设置

1. maven的属性设置

   设置maven的常用属性

```
<properties>
    <!--maven构建项目使用的是utf-8 ， 避免中文的乱码-->
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!--编译java代码使用的jdk版本-->
    <maven.compiler.source>1.8</maven.compiler.source>
    <!--你的java项目应该运行在什么样的jdk版本-->
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

2. maven的全局变量

   自定义的属性：

   1. 在 通过自定义标签声明变量（标签名就是变量名）
   2. 在pom.xml文件中的其它位置，使用 ${标签名} 使用变量的值

自定义全局变量一般是定义依赖的版本号，当你的项目中要使用多个相同的版本号，先使用全局变量定义，在使用${变量名}

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
    <!--自定义变量，表示版本号-->
    <spring.version>5.2.5.RELEASE</spring.version>
</properties>

<dependencies>
    <!--spring依赖-->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
      <version>${spring.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
</dependencies>
```

3. 指定资源位置插件

   src/main/java 和 src/test/java 这两个目录中的所有 *.java 文件会分别在 comile 和 test-comiple 阶段被编译，编译结果分别放到了 target/classes 和 targe/test-classes 目录中，但是这两个目录中的其他文件都会被忽略掉，如果需要把 src 目录下的文件包放到 target/classes 目录，作为输出的 jar 一部分。需要指定资源文件位置。以下内容放到标签中。

```
  <build>
    <resources>
        <resource>
             <!--所在的目录-->
            <directory>src/main/java</directory>
            <includes>
                  <!--包括目录下的 .properties, .xml文件都会扫描到-->
                <include>**/*.properties</include>
                <include>**/*.xml</include>
```

```
            </includes>
            <!-- fltering 选项 false 不启用过滤器， *.property 已经起到过滤的作用了 -->
            <filtering>false</filtering>
        </resource>
    </resources>
</build>
```

作用：mybatis课程中会用到这个作用

默认没有使用resources的时候，maven执行编译代码时，会把src/main/resource目录中的文件拷贝到target/classes目录中。

我们的程序有需要把一些文件放在 src/main/java 目录中，当我们在执行java程序时，需要用到 src/main/java 目录中的文件；就需要告诉 maven 在 mvn compile src/main/java 目录下的程序时，需要把文件一同拷贝到 target/classes 目录中；此时就需要在 中加入