

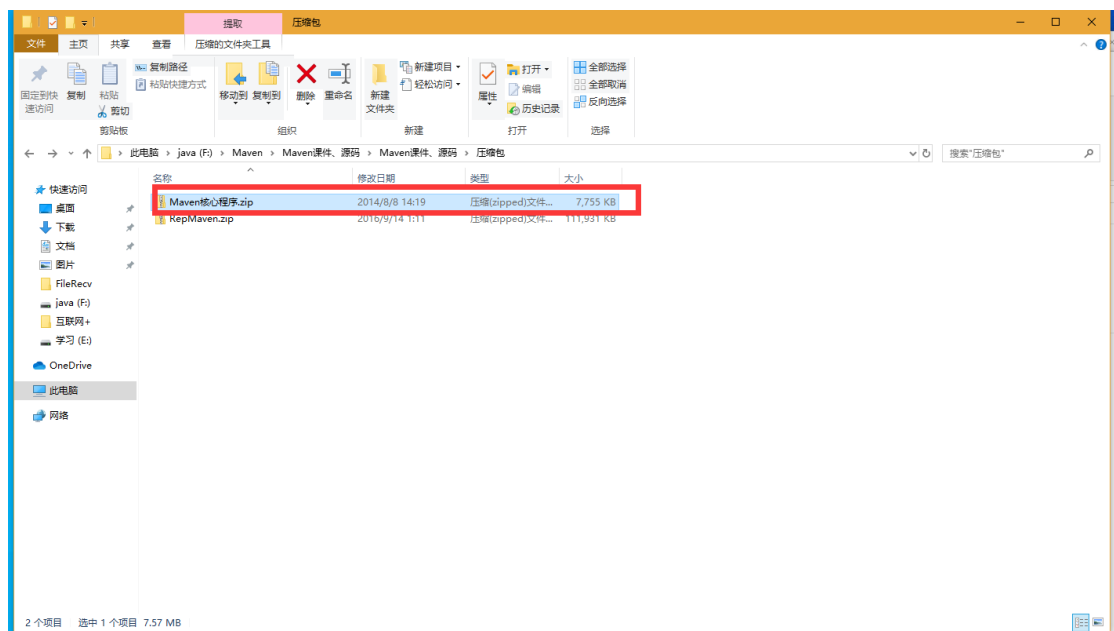
# Maven

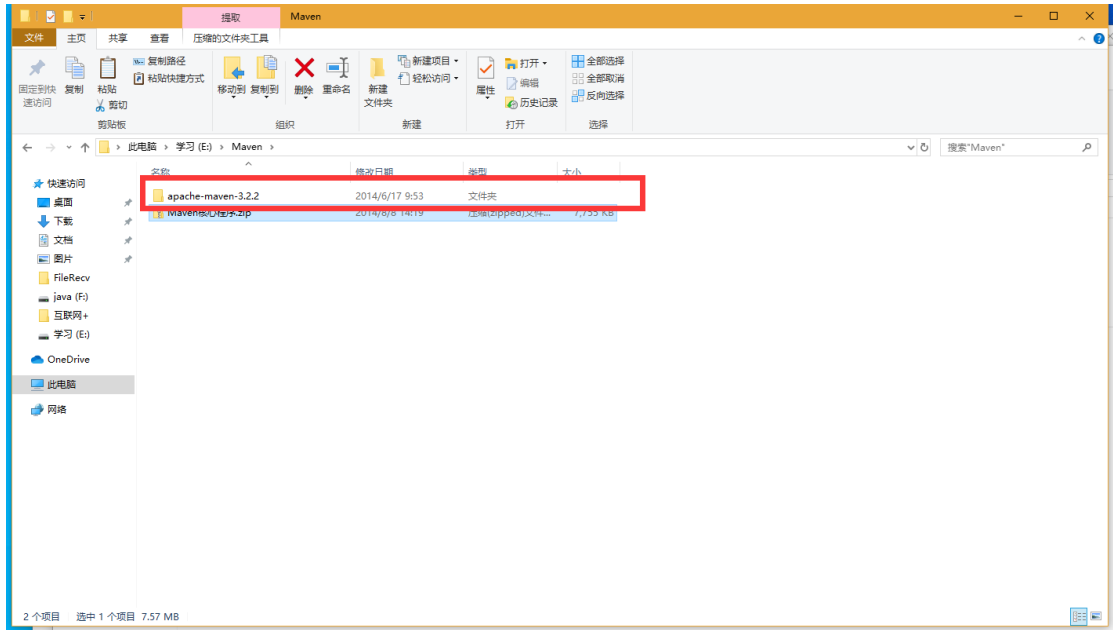
## 配置环境变量

### 1. 检查 JAVA\_HOME 环境变量

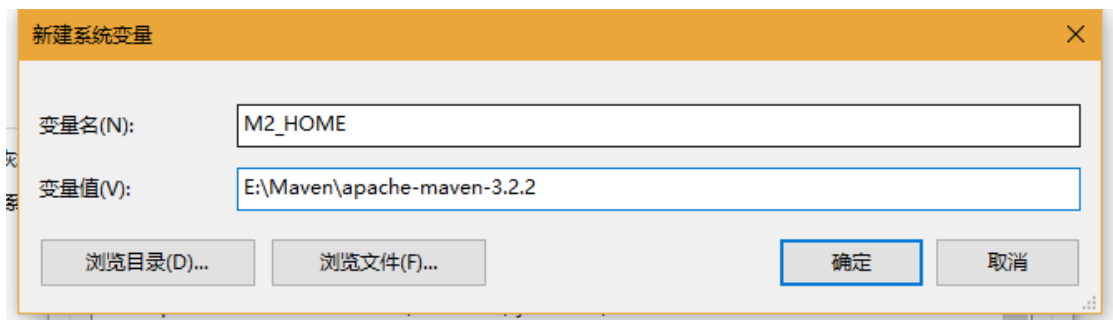
```
C:\Users\29924>echo %JAVA_HOME%  
D:\软件\JAVA\java-11-openjdk-11.0.11.9-1.windows.redhat.x86_64
```

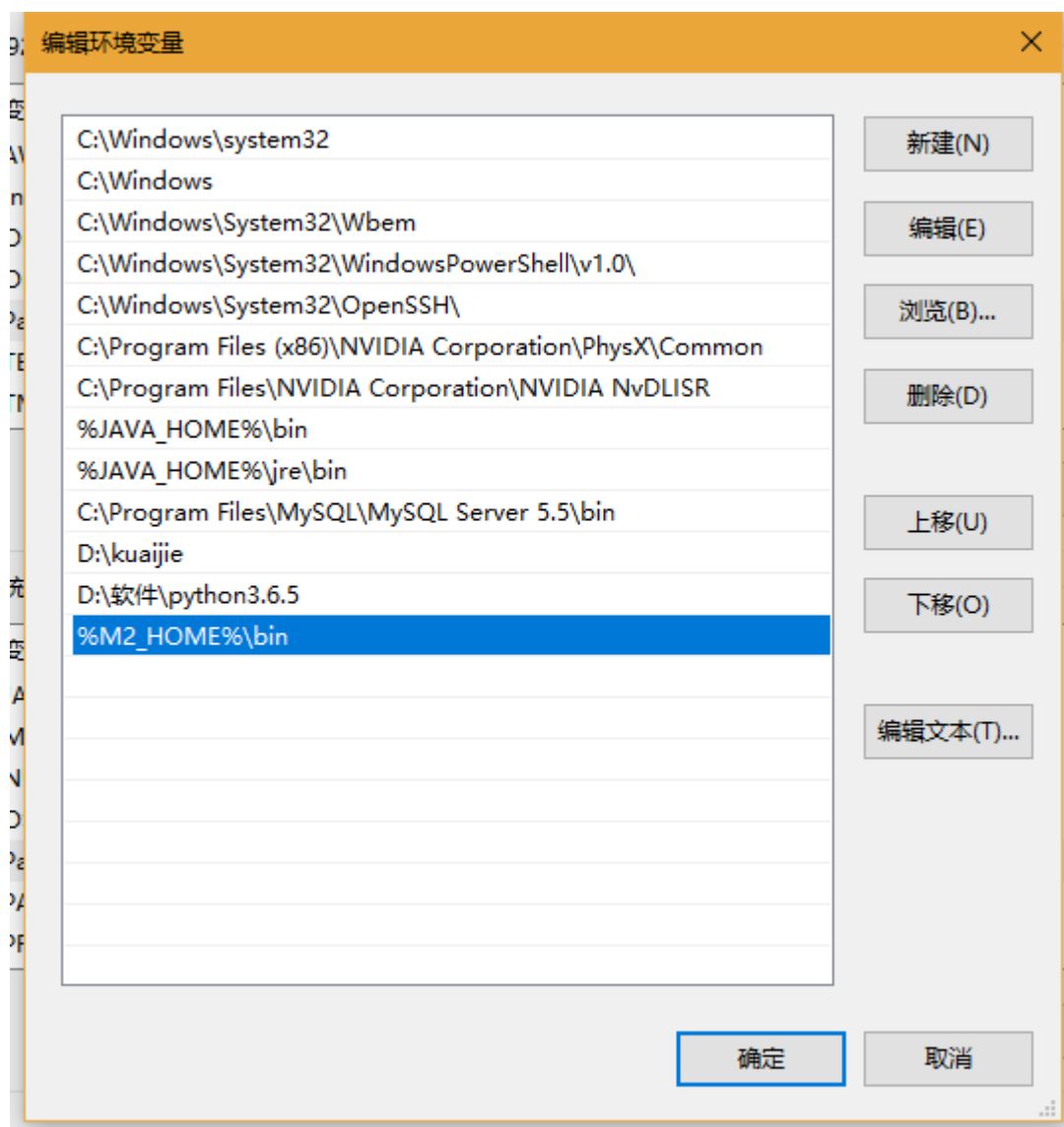
### 2. 解压 zip 包,解压到一个无中文和空格的文件夹下





### 3. 配置 maven 的环境变量





#### 4. 使用 `mvn -v` 查看版本

```
C:\Users\29924>mvn -v
Apache Maven 3.2.2 (45f7c06d68e745d05611f7fd14efb6594181933e; 2014-06-17T21:51:42+08:00)
Maven home: E:\Maven\apache-maven-3.2.2
Java version: 11.0.11, vendor: Red Hat, Inc.
Java home: D:\软件\JAVA\java-11-openjdk-11.0.11.9-1.windows.redhat.x86_64
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "dos"
C:\Users\29924>
```

### Maven 的核心概念

#### 1. 约定的目录结构

#### 2. POM

3. 坐标
4. 依赖
5. 仓库
6. 生命周期
7. 继承
8. 聚合

## 常用的 maven 命令

### 1. 注意

执行构建过程的 maven 命令,必须进入 pom.xml 所在的目录

### 2. 常用命令:

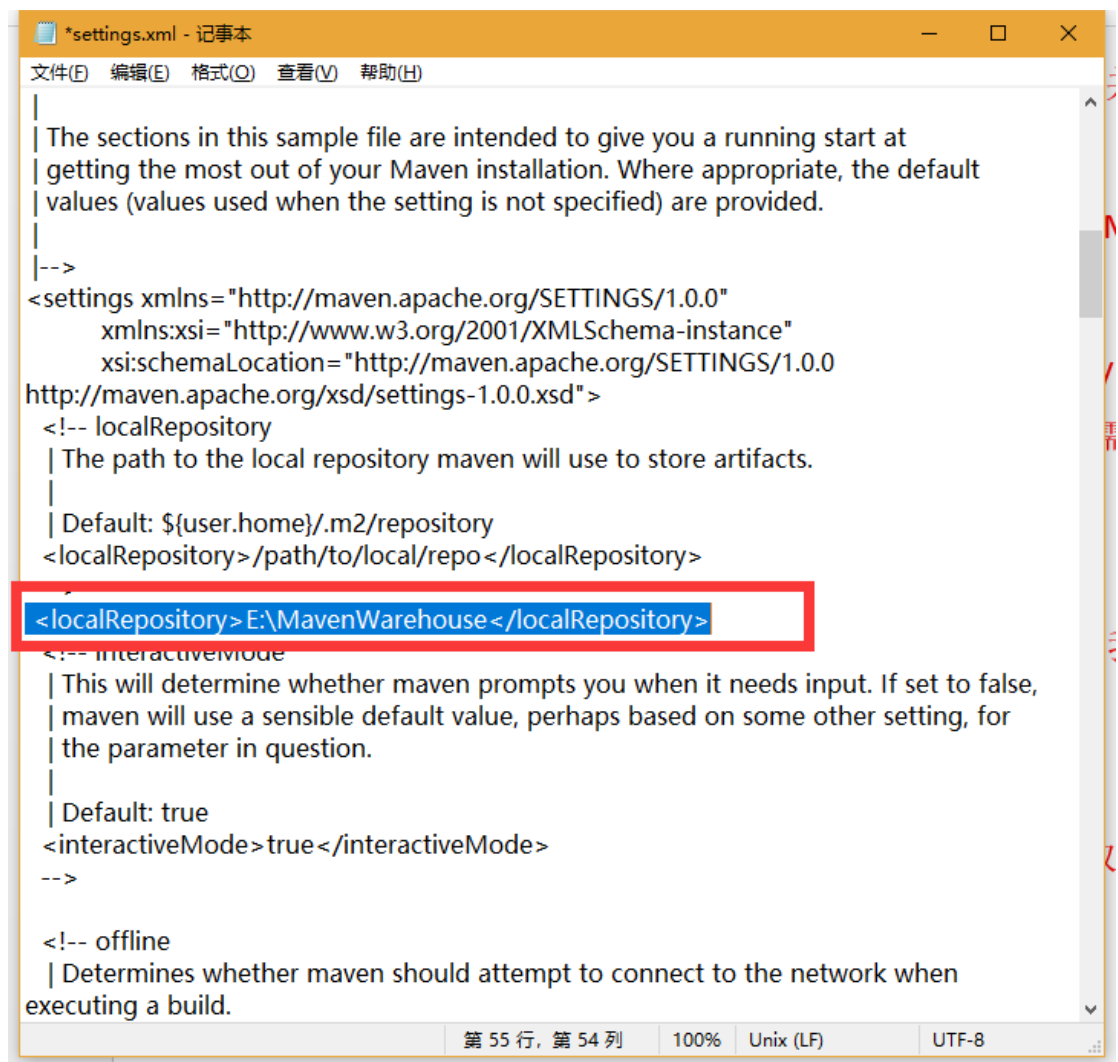
- a) mvn clear:清理
- b) mvn compile:编译
- c) mvn test-compile:编译测试程序
- d) mvn test:执行测试
- e) mvn package:打包
- f) mvn install 安装
- g) mvn site:生成站点

## 关于联网问题+构建自己仓库

### 1. Maven 的核心程序中仅仅定义了抽象的生命周期,但是具

体的工作必须由特定的插件来完成.而插件本身并不包含在 Maven 的核心程序中

2. 当我们执行的 Maven 命令需要用到某些插件时,Maven 核心程序会先从本地仓库查找
3. 本地仓库的默认位置:系统家目录\m2\repository
4. Maven 核心程序如果在本地仓库中找不到需要的插件,它会自动连接外网,到中央仓库下载
5. 如果此时无法连接外网,则构建失败
6. 修改默认本地仓库的位置可以让 Maven 程序到我们事先准备好的仓库去查找插件
  - a) 找到 Maven 解压目录\conf\settings.xml
  - b) 在 settings.xml 找到<localRepository>标签提取出来
  - c) 修改路径



```
*settings.xml - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

|
| The sections in this sample file are intended to give you a running start at
| getting the most out of your Maven installation. Where appropriate, the default
| values (values used when the setting is not specified) are provided.
|
|-->
<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:\MavenWarehouse</localRepository>
  <!-- interactiveMode
  | This will determine whether maven prompts you when it needs input. If set to false,
  | maven will use a sensible default value, perhaps based on some other setting, for
  | the parameter in question.
  |
  | Default: true
  <interactiveMode>true</interactiveMode>
  -->

  <!-- offline
  | Determines whether maven should attempt to connect to the network when
  | executing a build.
```

## 创建约定的目录结构

- a) 根目录:工程名
- b) src 目录:源码
  - c) main 目录:存放主程序
    - i. java 目录:存放 java 源文件
    - ii. resources 目录:存放框架和其他工具的配置文件
- d) test 目录:存放测试程序
  - i. java 目录:存放 java 源文件

- ii. resources 目录:存放框架和其他工具的配置文件
- e) pom.xml:Maven 配置文件

## POM(project object model)

1.含义:项目对象模型

2.pom.xml 对于 Maven 工程是核心配置文件,构造过程相关的一切设置都在这个文件中进行配置重要程度相当于 web.xml 对于动态 web 工程

## 坐标

1. 数学中的坐标:

- a) 在平面上,使用  $x,y$  两个向量可以唯一的定位平面中的任何一个点
- b) 在空间上,使用  $x,y,z$  三个向量可以唯一的定位空间中的任何一个点

2. Maven 坐标

使用三个向量在长裤中唯一定位一个 Maven 工程

- 1) groupid:域名+项目名
- 2) artifactid:模块名
- 3) version:版本

## 仓库

1. 仓库分类:

a) 本地仓库:当前电脑上部署的仓库目录,为当前电脑上所有 **Maven** 工程服务

b) 远程仓库:

1. 私服:局域网之内的服务器

2. 中央仓库:架设在外网上,为所有的 **Maven** 工程服务器

3. 中央仓库镜像:负载均衡,减轻中央仓库压力,提高用户访问速度

2. 仓库保存的内容:

a) **Maven** 自身的插件

b) 第三方框架或工具的 jar 包

c) 我们自己开发的工程

## 依赖

1.**Maven** 解析依赖信息时会到本地仓库中查找被依赖的 jar 包

2.依赖的范围:

1)compile

a)对主程序是否有效 有

b)对测试程序是否有效 有

c)是否参与打包 参与

2)test



- a)对主程序无效
- b)对测试程序有效
- c)不参与打包

### 3)provided

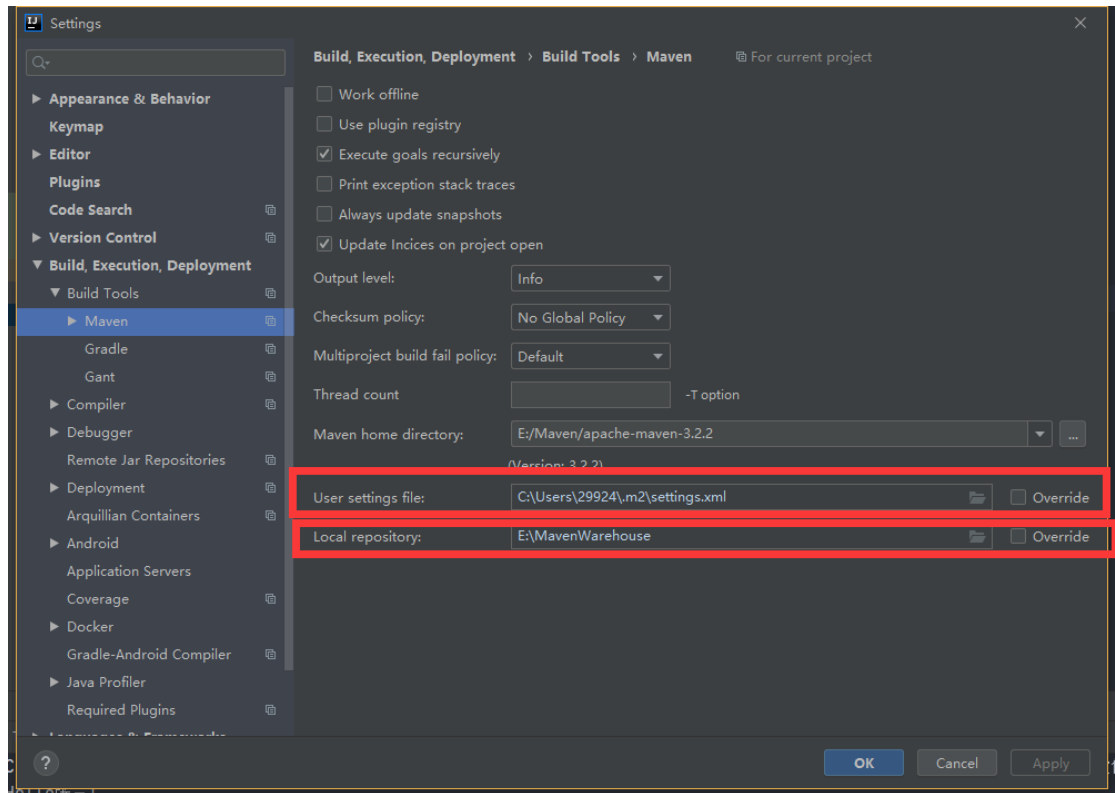
- a)对主程序有效
- b)测试有效
- c)不参与打包
- b)不参与部署

## 生命周期

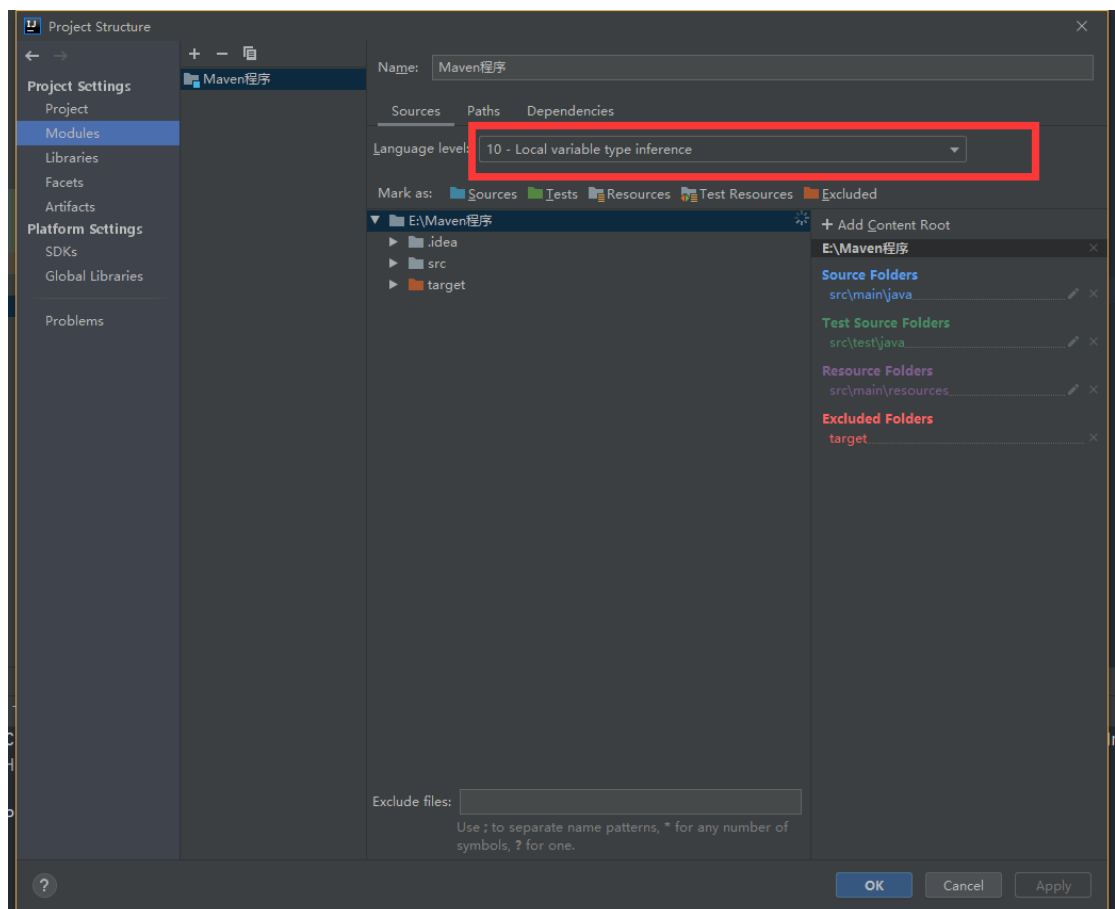
1. 各个构建环节的执行顺序:不能打乱顺序,必须按照既定的正确顺序执行
2. **Maven** 的核心程序中定义了抽象的生命周期,声明周期中各个阶段具体任务由插件完成
3. **Maven** 核心程序为了更好实现自动化构建,按照这一特点执行生命周期中各个阶段:不论现在要执行生命周期哪个阶段,都是从这个声明周期最初的位置开始执行

## IDEA 中配置 Maven 环境

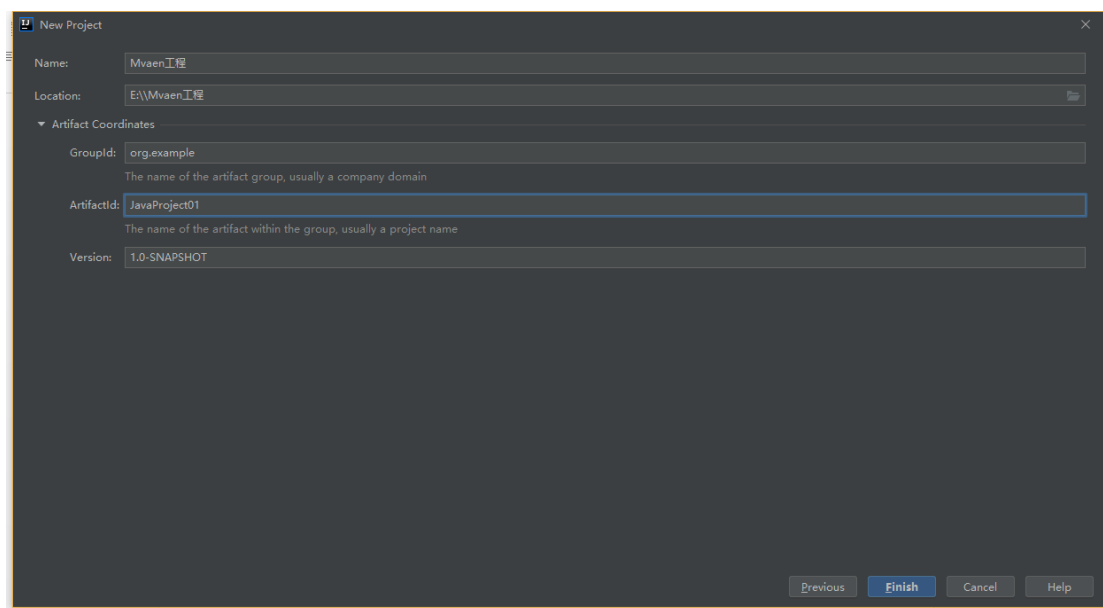
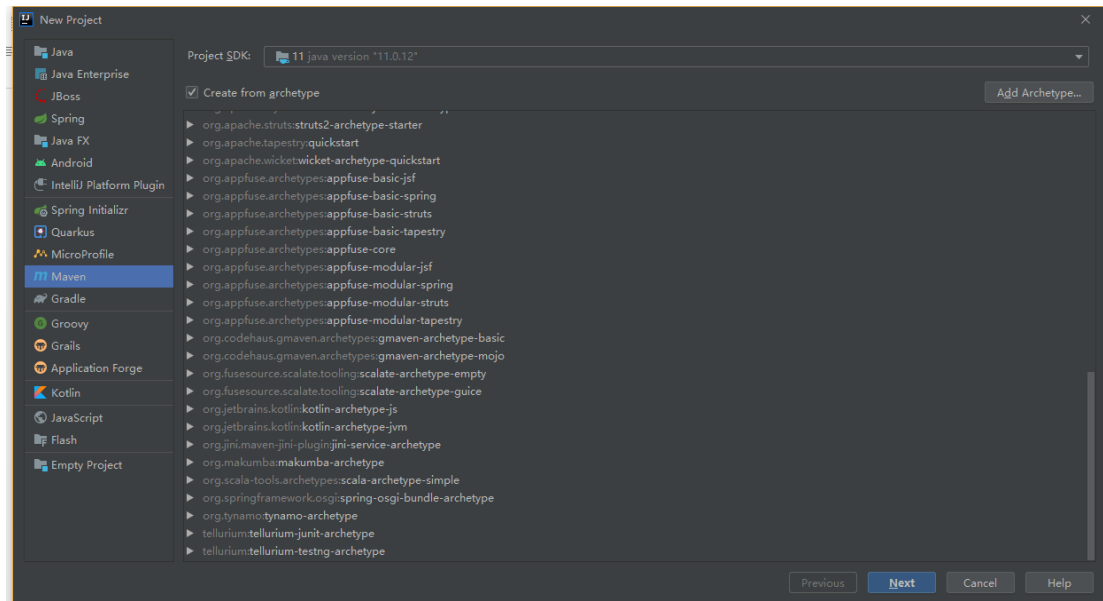
1. 设置 settings.xml 位置和仓库位置



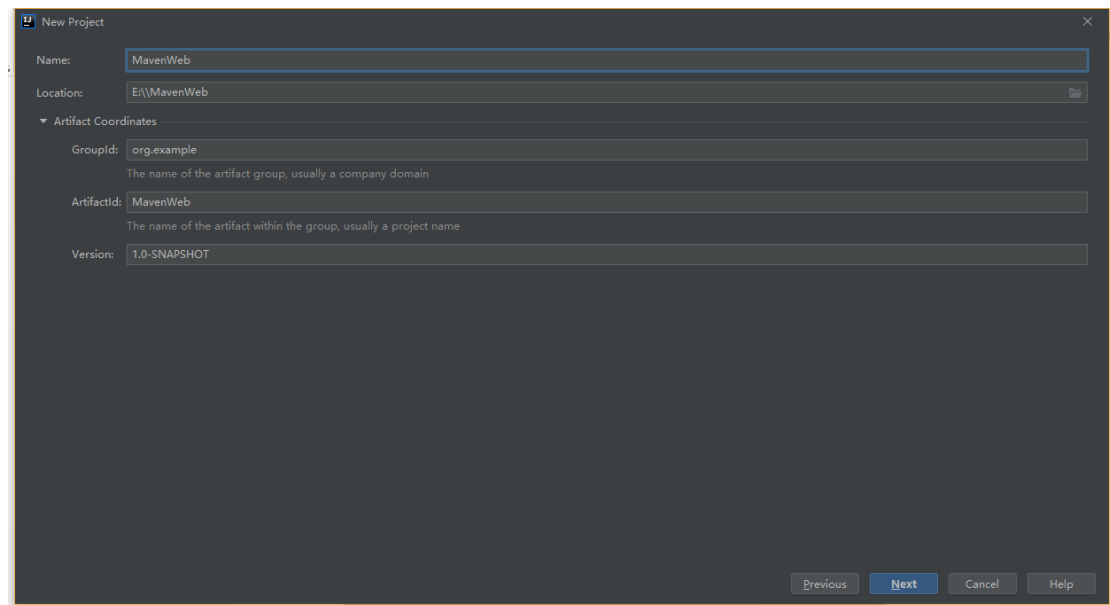
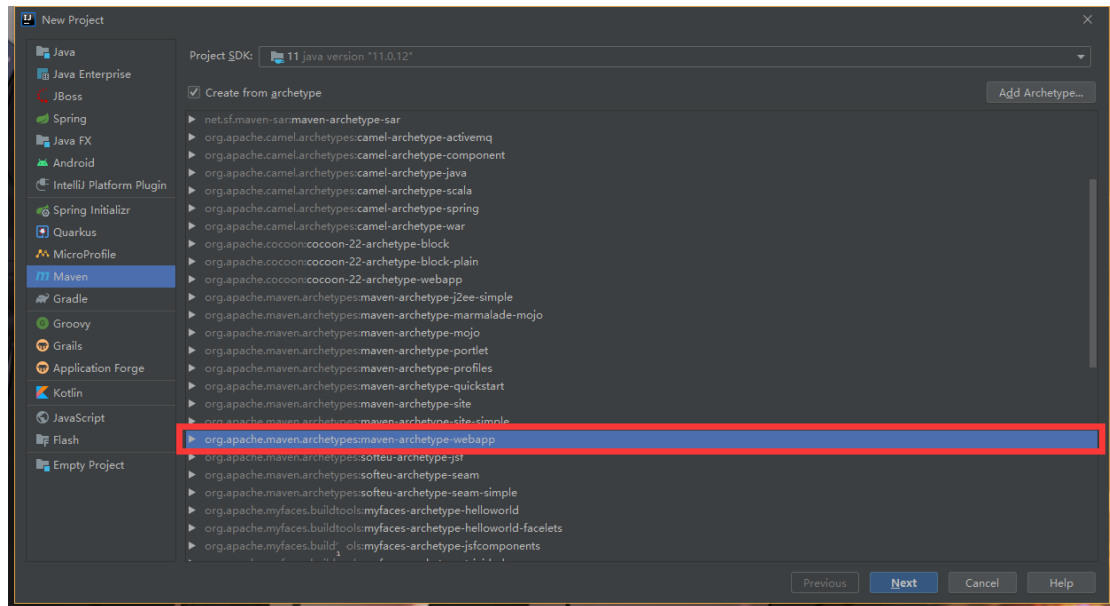
## 2. 设置版本

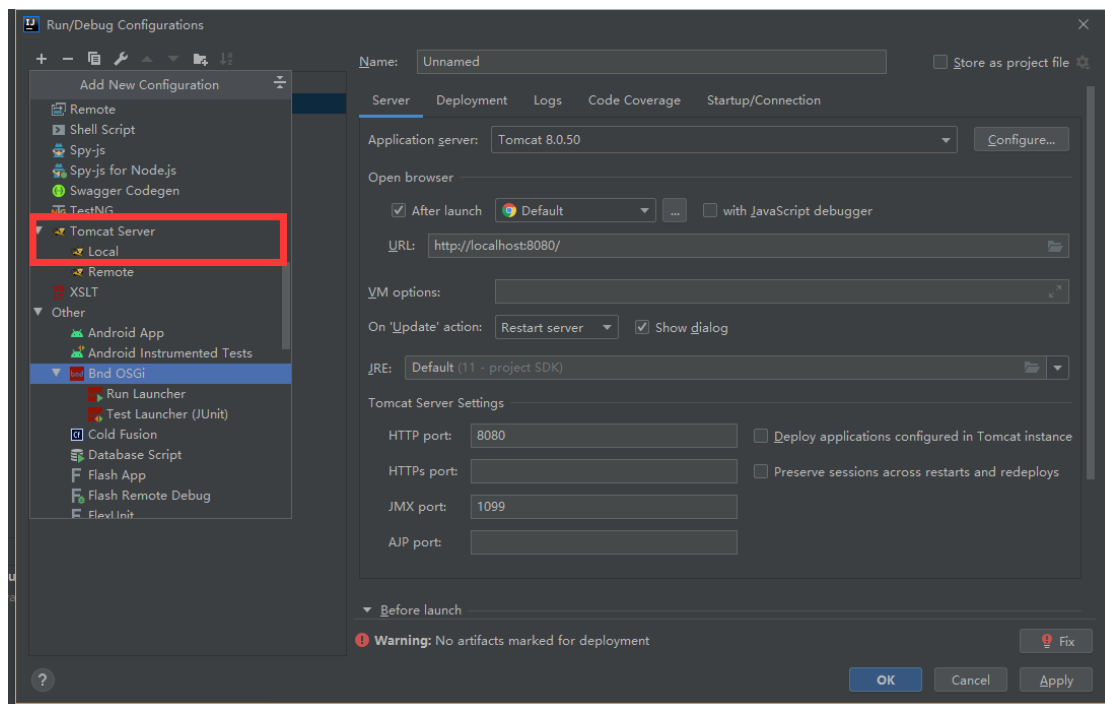
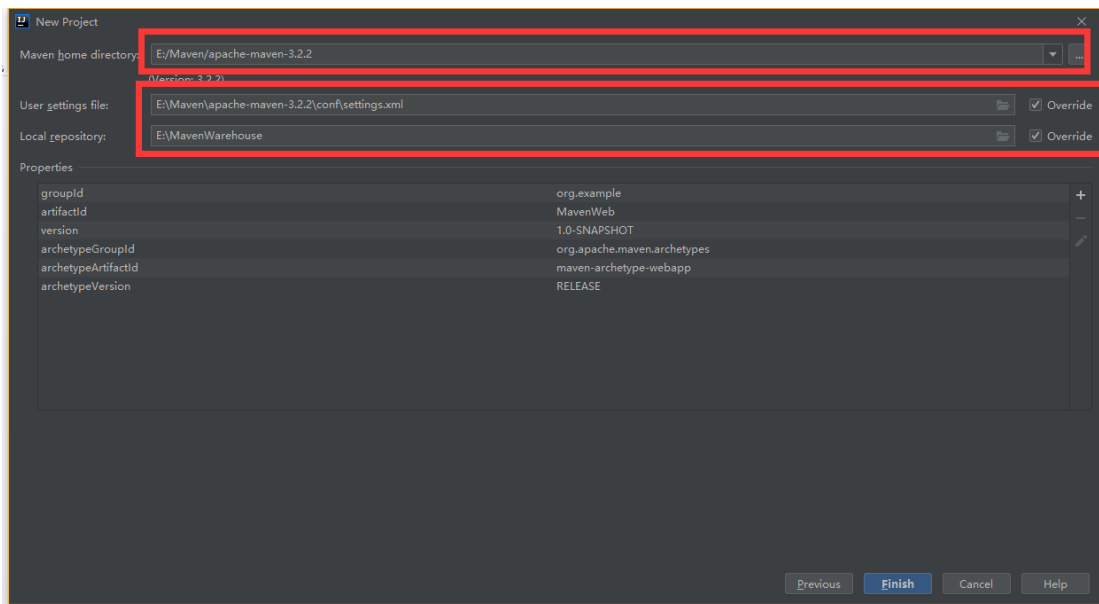


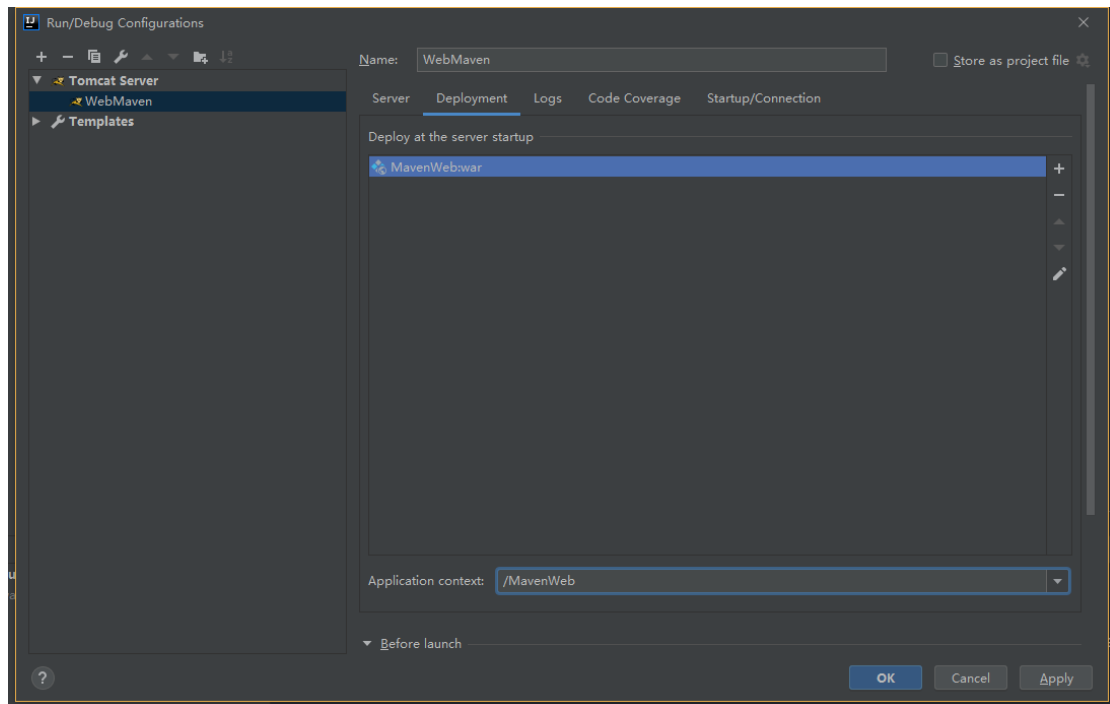
## IDEA 创建 Maven 工程



## IDEA 创建 Maven-Web 工程







## 导入 Jar 包

```
<!--Maven 说明-->
<dependencies>
  <!--导入标签-->
  <dependency>
    <!--jar 包的上级目录-->
    <groupId>org.springframework</groupId>
    <!--导入的 jar 包名-->
    <artifactId>spring-tx</artifactId>
    <!--版本号-->
    <version>4.1.1.RELEASE</version>
  </dependency>
</dependencies>
```

## 依赖问题

当多个 Maven 联合的时候配置一个 XML 多个项目也会导入

当有一些不需要传递的时候

在版本号下面加入 Exclusions 标签

```

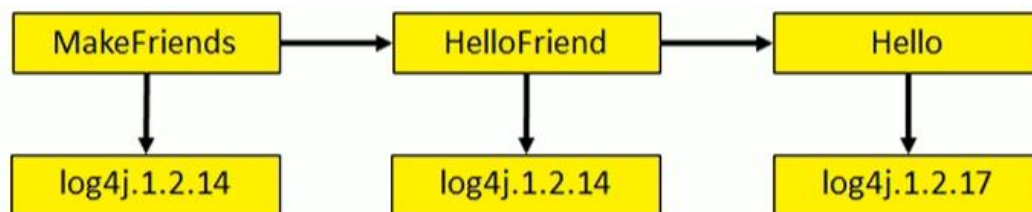
<exclusions>
  <exclusion>
    <!--坐标-->
    <groupId>com.github.spring-boot-ext</groupId>
    <artifactId>spring-boot-starter-ext</artifactId>
  </exclusion>
</exclusions>

```

## 依赖原则

作用:解决工程 jar 包冲突问题

情景一:路径最短选哪个



情景二:如果路径一样,谁先声明谁优先



## 统一依赖版本号

1. 添加全局的<properties>标签,然后在里面添加一个自定义标签
2. 使用\${}替换下面版本号

```

<properties>
  <banben>4.1.1.RELEASE</banben>
</properties>

<!--Maven 说明-->
<dependencies>

```

```

<!--导入标签-->
<dependency>
  <!--jar 包的上级目录-->
  <groupId>org.springframework</groupId>
  <!--导入的 jar 包名-->
  <artifactId>spring-tx</artifactId>
  <!--版本号-->
  <version>${banben}</version>
</dependency>
</dependencies>

```

## 继承

### 作用:

由于只有 `compile` 有传递性, `test` 没有传递性, 所有有的时候 Maven 工程很多时候 `junit` 版本不统一, 就需要用到继承

### 实现:

将 `junit` 依赖版本声明在“父”工程中, 在子工程声明依赖时不声明版本, 这样就继承了父类的依赖版本

1. 创建父工程
2. 创建子工程
3. 认贼作父

```

<!-- 子工程中声明父工程 -->
<parent>
  <groupId>com.atguigu.maven</groupId>
  <artifactId>Parent</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <!-- 以当前文件为基准的父工程pom.xml文件的相对路径 -->
  <relativePath>../Parent/pom.xml</relativePath>
</parent>

```



#### 4. 删除重复部分

#### 5. 在父工程中声明依赖管理

```
<!-- 配置依赖的管理 -->
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.9</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

#### 6. 删除子工程版本号

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>test</scope>
  </dependency>
```

注意:配置继承后要先安装父工程再安装子工程

### 聚合

作用:一件安装各个模块工程

实现: 在父工程是 **pom.xml** 中添加子工程的路径

然后 **mvn install** 父工程就统一安装

```
<!-- 配置聚合 -->
<modules>
  <!-- 指定各个子工程的相对路径 -->
  <module>../Hello</module>
  <module>../HelloFriend</module>
  <module>../MakeFriends</module>
</modules>
```

## 查找依赖网址

<https://mvnrepository.com/artifact/org.springframework/spring-webmvc/5.3.1>