一 Maven 简介

1 什么是 maven

Maven 是一个项目管理工具

2 Maven 的作用是什么

简化 jar 包资源的使用：可以利用一段 jar 包资源描述信息，通过 maven 自动引入 jar 包

按标准进行团队开发：简化团队协作过程

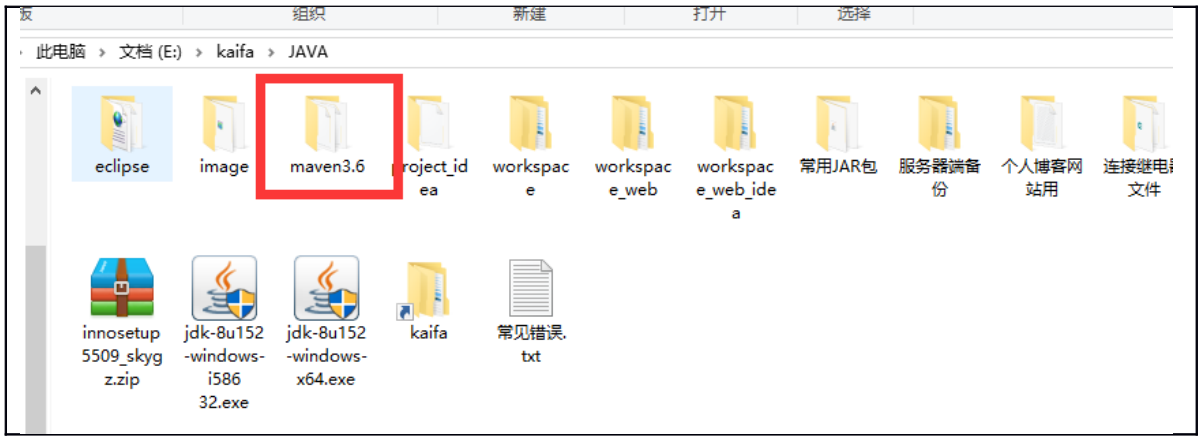
二 Maven 安装配置

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

1 环境准备

当前安装的为 3.6 版本，最低要求 JDK 版本为 1.7

2 解压到无中文无空格路径



3 Maven 家目录结构

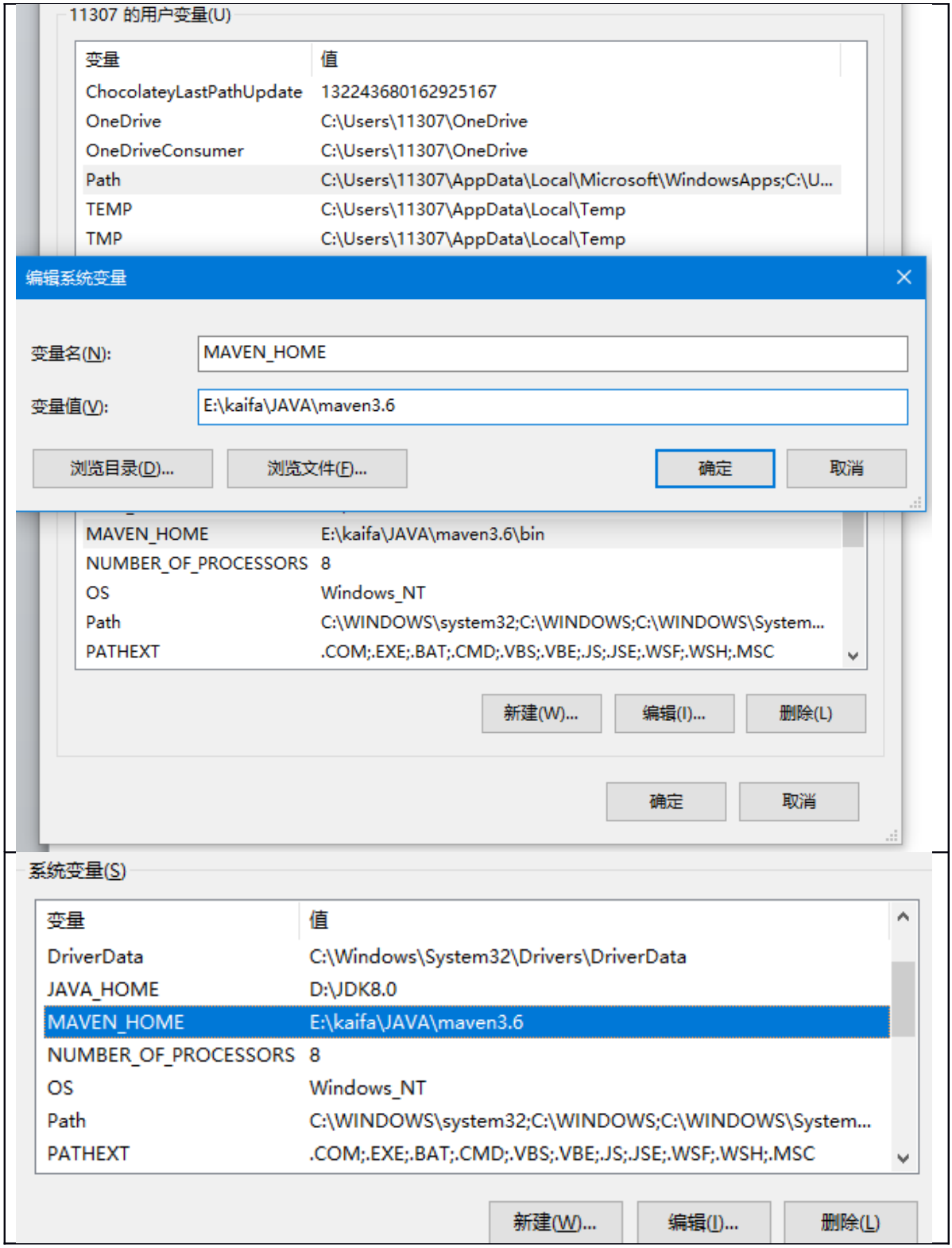
bin：maven 运行命令脚本文件

boot：启动框架，maven 也是用 java 编写的

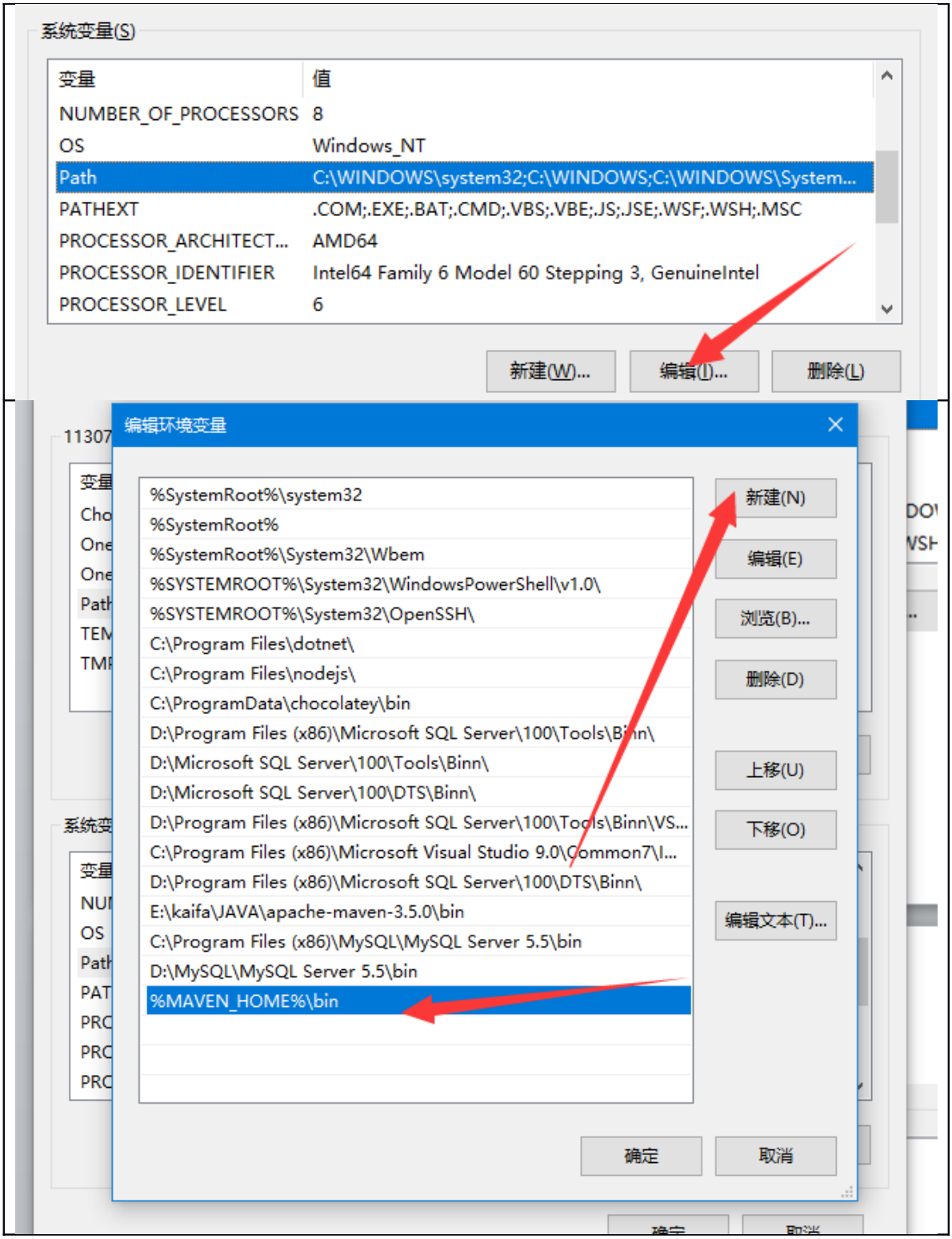
conf：配置文件的文件夹，例如 setting.xml

4 配置环境变量

4.A 添加一个 MAVEN_HOME 变量，指向 maven 家目录

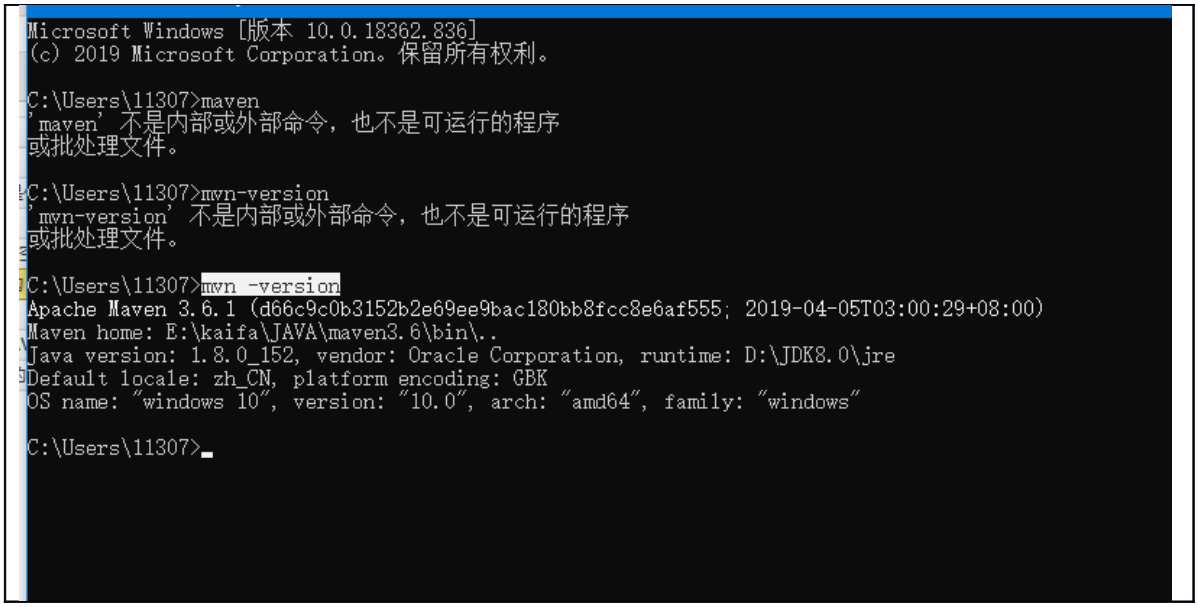


4.B 添加家目录中的 bin 目录到系统 path 环境变量中



4.C 验证安装结果

打开新 Cmd 窗口输入 mvn –version 出现如下界面即表示安装成功



三　Maven 的库

1　Maven 库的概念

Maven 中实现项目的管理，使用一些插件，jar 包等内容，都是作为一种资源需要保存在一个库中(repository)

2　Maven 库的种类

2.A　远程库

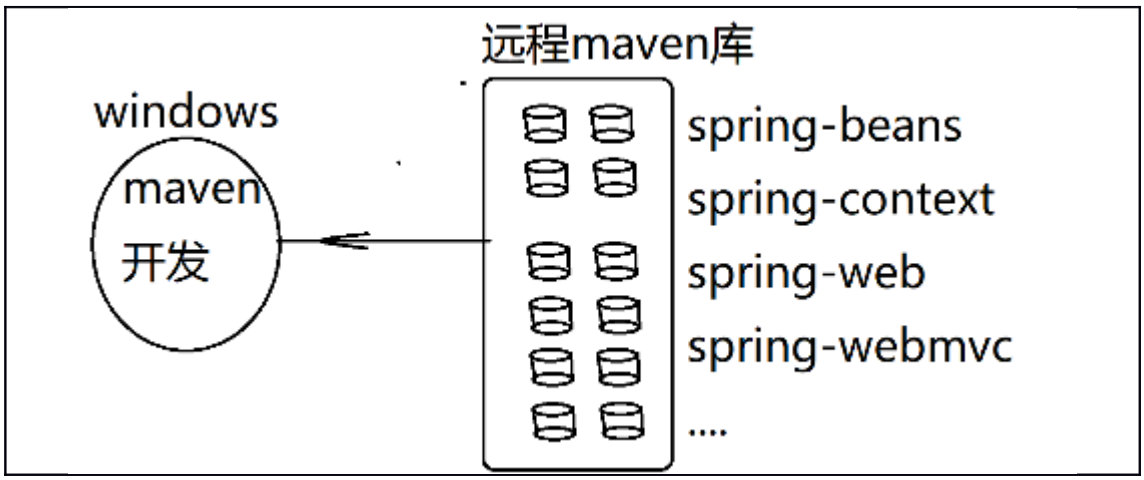
保存在一个远程服务器的 maven 资源的库，可以被 maven 客户端远程连接使用的库。

maven 默认使用的远程库叫 maven 的中央库，由 maven 社区去维护的，我们使用这种远程库不能进行资源的编辑和新增的，只能读取。Maven 社区将全世界常用的资源加入到中央库中，供所有开发者使用。

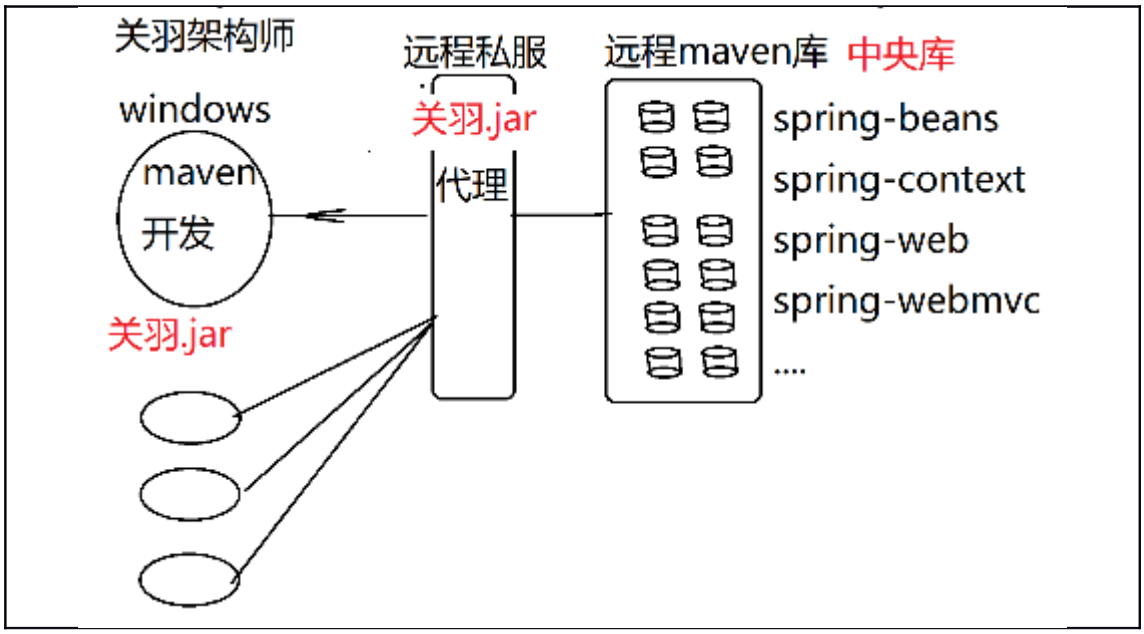
只要客户端需要网络连接远程服务器获取服务器的资源，这个远程服务器中的资源就是远程库。

远程库分为二种:中央库(只读)，远程私服(可读可修改，并且利用远程私服代理访问

中央库)。



远程私服：弥补了中央库不能写入、存储自定义的资源的缺陷

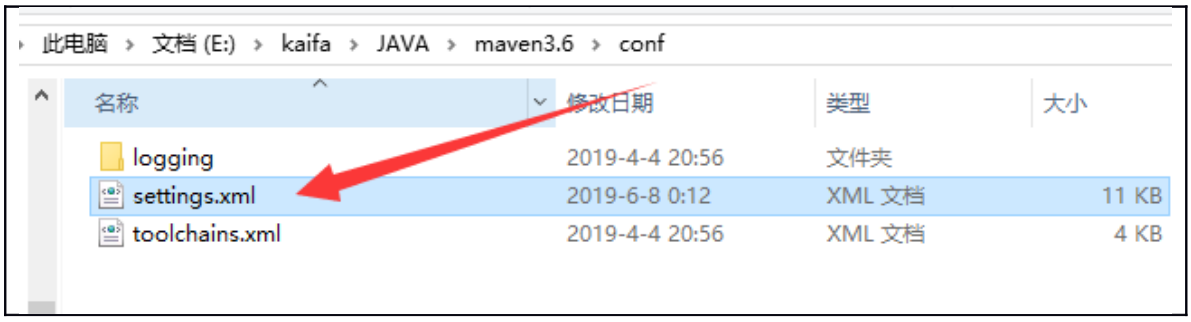


2.B 本地库

如果每次 maven 运行都要占用网络带宽，实现资源下载。非常浪费时间，开发效率不高。所以 maven 准备了本地库，把从远程下载的资源放到本地保存，下次再使用相同资源，就不用再下载了，而是直接使用本地库的内容。

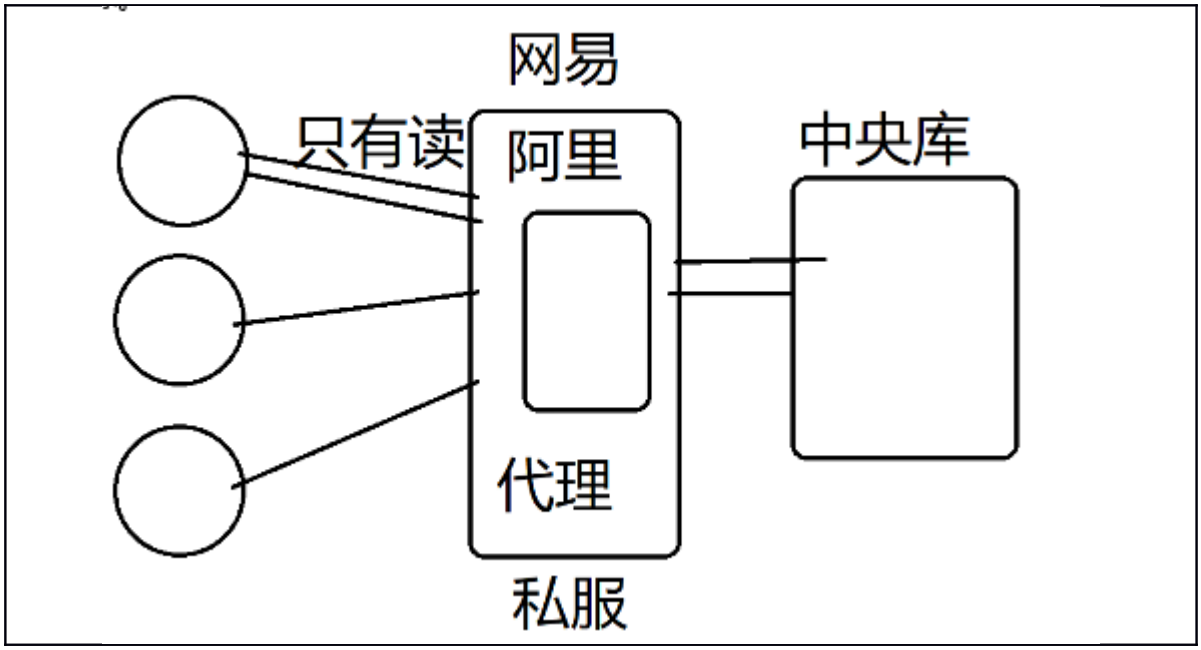
3 配置 maven 的库

3.A 找到配置文件 maven 家目录/conf/setting.xml



3.B 准备国内镜像(远程私服)

如果网络连接的是 maven 中央库使用资源（外国服务器，速度相当慢。）。可以通过配置国内的镜像，解决这个问题。就是实现代理访问中央库，所以国内网速可以迅速下载资源。



这里我们配置的阿里的镜像:

```
<mirror>
    <id>nexus-aliyun</id>
    <mirrorOf>central</mirrorOf>
```

```
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    </mirror>
```

```
    </mirror>
    -->
    <mirror>
        <id>nexus-aliyun</id>
        <mirrorOf>central</mirrorOf>
        <name>Nexus aliyun</name>
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    </mirror>
  </mirrors>
```

3.C 本地库配置

本地电脑上新建一个的文件夹

然后在 settings.xml 文件的 settings 标签内配置该文件夹

```
<localRepository>E:/kaifa/JAVA/maven_localrepository</localRepository>
```

```
<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.apa
  <!-- 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:/kaifa/JAVA/maven_localrepository</localRepository>
  <!-- interactiveMode
```

四 Maven 的生命周期

了解生命周期的含义。

掌握 maven 生命周期不同环节的命令。

掌握 maven 入门创建。

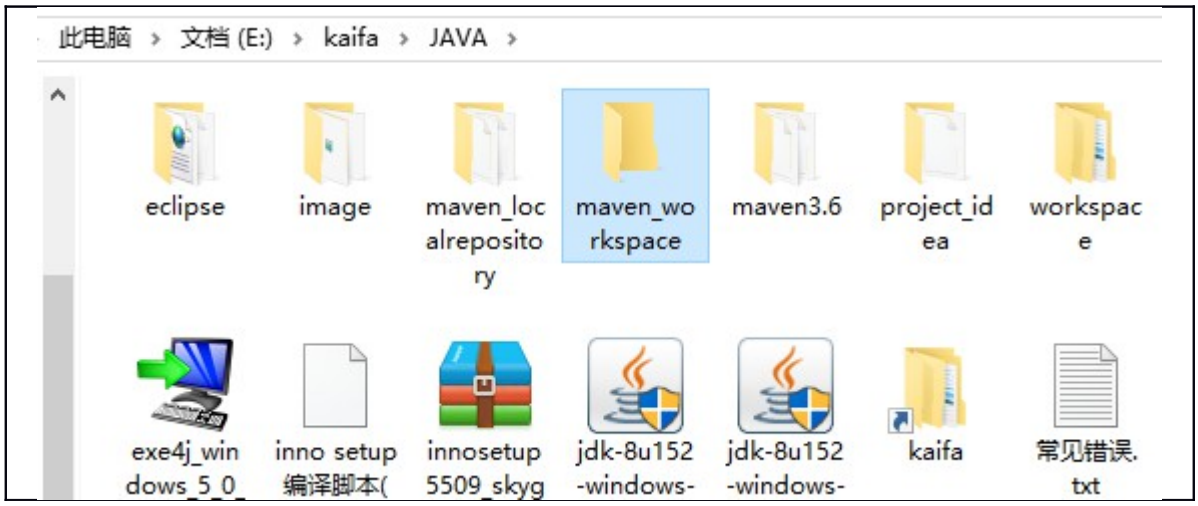
1 Maven 的生命周期

Maven 作为项目管理工具，负责项目从无到有所经历的每一个环节，这个过程就是 maven 的生命周期。

包括：创建，编码，编译，测试，打包，安装，发布。

2 创建 maven 工作空间(文件夹)

2.A 创建工作空间

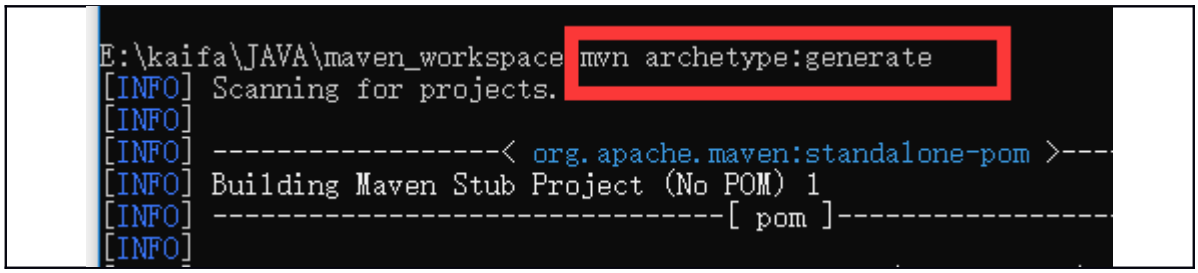


3 mvn archtype:generate（创建第一个 maven 项目）

在工作空间目录中执行命令 **mvn archtype:generate**

archetype：骨架

generate：转化生成



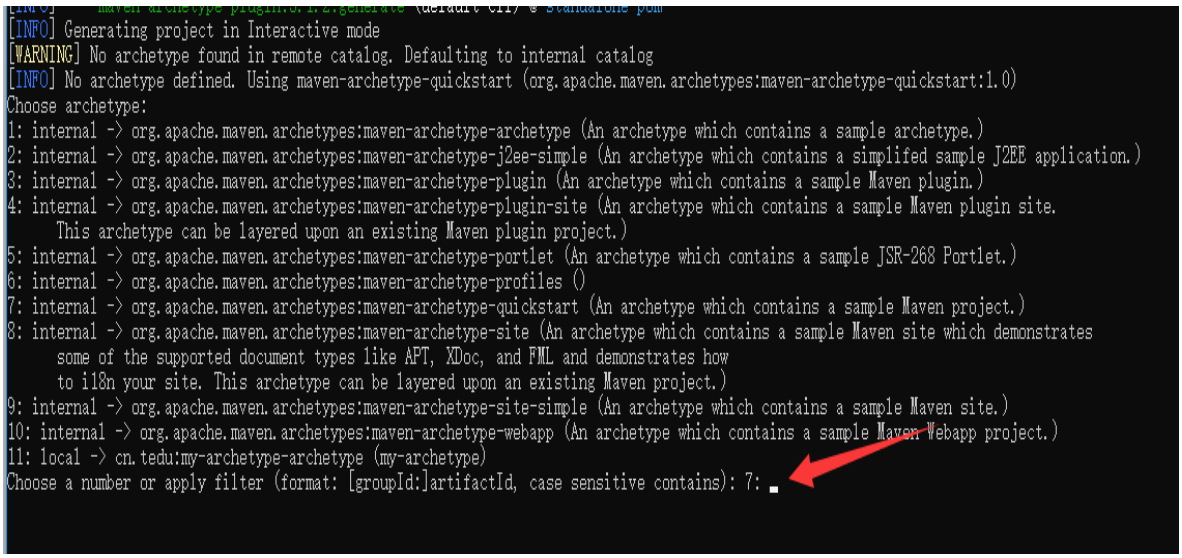
3.A 选择骨架，maven 中已经成型的项目骨架

什么是骨架：

maven 中已经成型的项目结构，根据选择的不同骨架，创建一个 maven

项目时生成不同的默认结构。默认选择 7 号骨架 quickstart，比较标准的 java 工程结构。

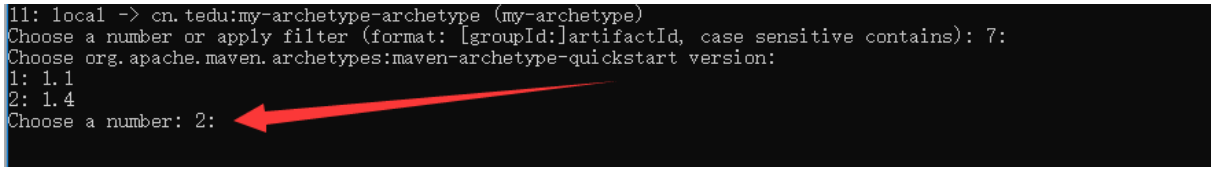
默认选 7



3.B 选择骨架版本

本地库自定义配置的，内容中有 1.1 1.4 二个版本的 quickstart 骨架

默认选择高版本 2



3.C 填写项目基本信息

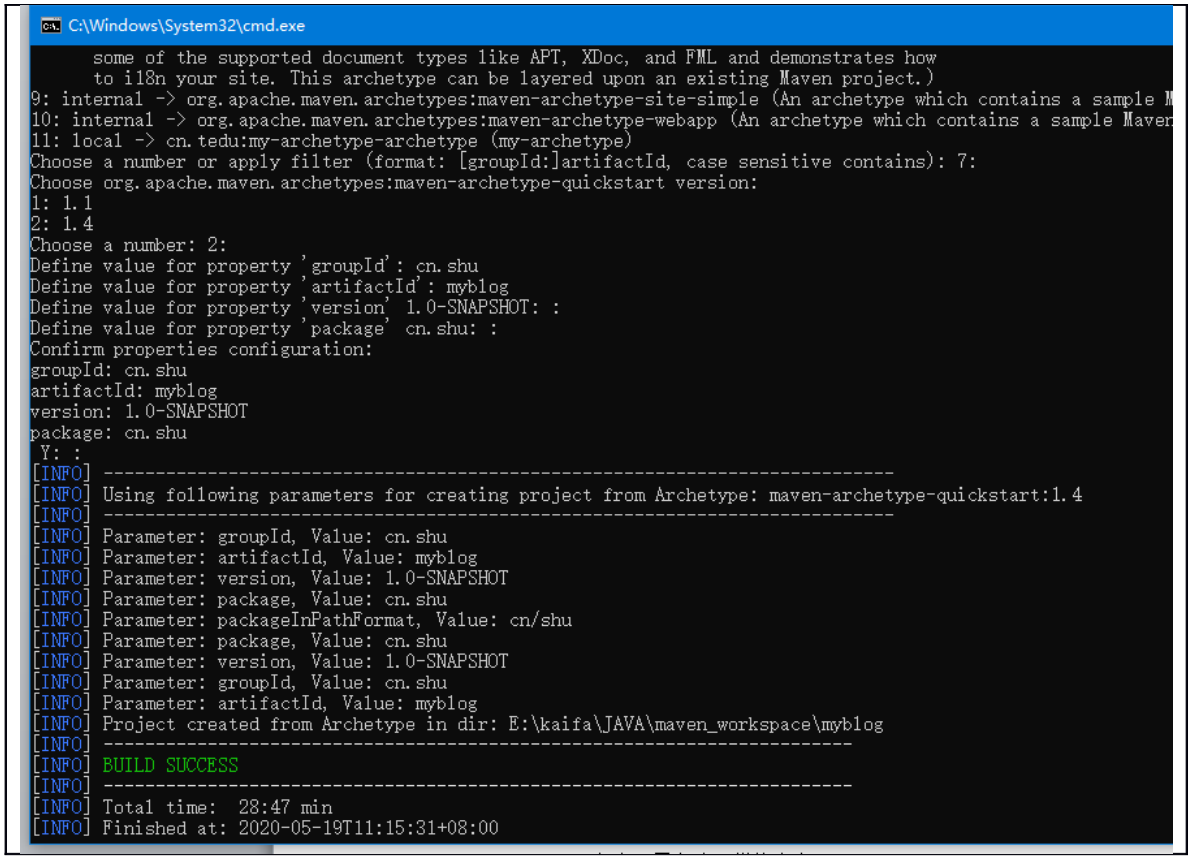
groupId: 表示组织 id，一般会使用一个项目，一个公司的域名倒写。

artifactId：一般为项目名称、模块名字

version：当前项目版本 默认 1.0-SNAPSHOT

package: 默认创建的包路径 默认使用 groupId

group、artifactId、version 可唯一确定一个项目



4 Maven 项目的目录机构

所选择的骨架，值会决定创建项目中默认有哪些文件、配置、报告等出现， maven 项目由一个固定的结构的

根目录：

src:文件夹，项目的代码配置源数据

main: 源代码

java:包和.java 文件

resources：配置文件，例如 applicationConext.xml

test：测试代码

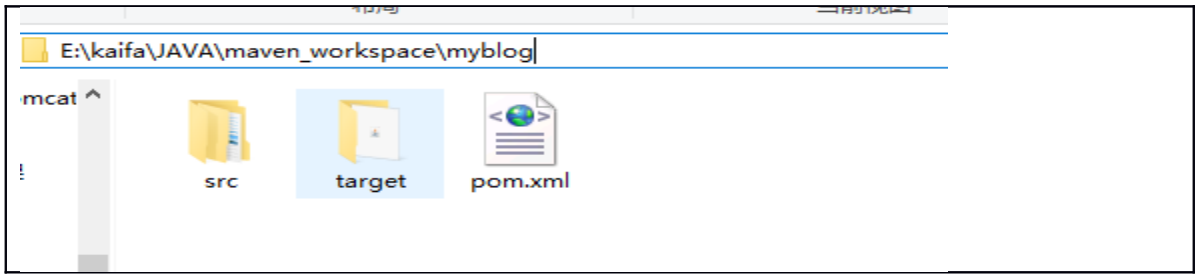
java:

resources：配置文件，测试代码如果没有测试中 resources 支持，将会加载 main

的 resources 的测试代码

target：项目的编译 class 文件、报告、打包等输出的目录

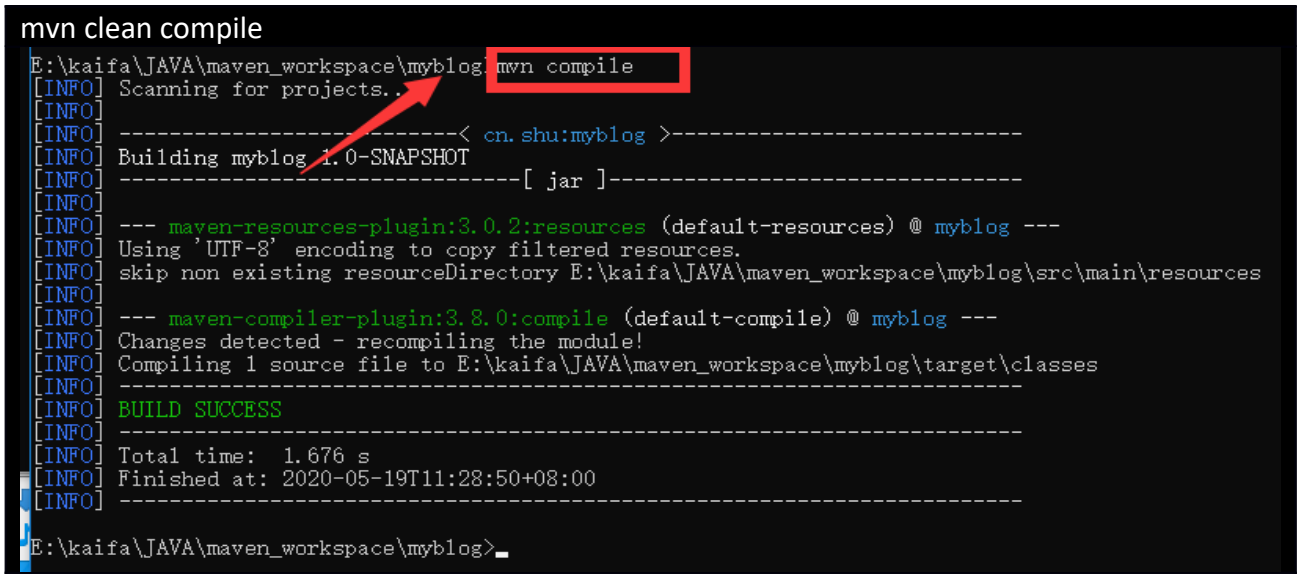
pom.xml：maven 项目核心管理 xml 文件，maven 所有命令对应这个项目一定会读取 pom.xml 进行管理



5 mvn compile (Maven 代码编译)

每一个 mvn 命令都需要在当前项目下运行，不能在工作空间中

通常会与清理 target 目录同时使用



6 mvn clean (清空 target)

target 保存了很多项目的输出文件，为了不影响下一次的使用，一般在调用命令，生成 target 文件之前，先清空一下，防止文件有冲突。

```
E:\kaifa\JAVA\maven_workspace\myblog>mvn clean
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------------< cn.shu:myblog >----------------------------
[INFO] Building myblog 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ myblog ---
[INFO] Deleting E:\kaifa\JAVA\maven_workspace\myblog\target
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.521 s
[INFO] Finished at: 2020-05-19T11:30:24+08:00
[INFO] ------------------------------------------------------------------------

E:\kaifa\JAVA\maven_workspace\myblog>
```

7 mvn test (测试)

在我们编写项目过程中，除了编译，写完代码时，也要进行测试的运行， maven 提供了测似的命令，并且它生成测试报告，方便观察纠错

```
E:\kaifa\JAVA\maven_workspace\myblog>mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------------< cn.shu:myblog >----------------------------
[INFO] Building myblog 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ myblog ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\kaifa\JAVA\maven_workspace\myblog\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ myblog ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:testResources (default-testResources) @ myblog ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\kaifa\JAVA\maven_workspace\myblog\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ myblog ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to E:\kaifa\JAVA\maven_workspace\myblog\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ myblog ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running cn.shu.AppTest
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.048 s - in cn.shu.AppTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.096 s
[INFO] Finished at: 2020-05-19T11:37:55+08:00
```

8 mvn package (打包)

当项目实现了所有代码的编写，编译完成，测试也完成，将 maven 项目打成 jar 包或者 war 包，到该运行的服务器运行，提供给其他人使用。

从日志可以看到，靠后的操作都包括了前面的环节，例如打包时会自动编译、测试等

例如：可以指定参数 -DskipTests 跳过测试

通常和 clean 连用

mvn clean package -DskipTests

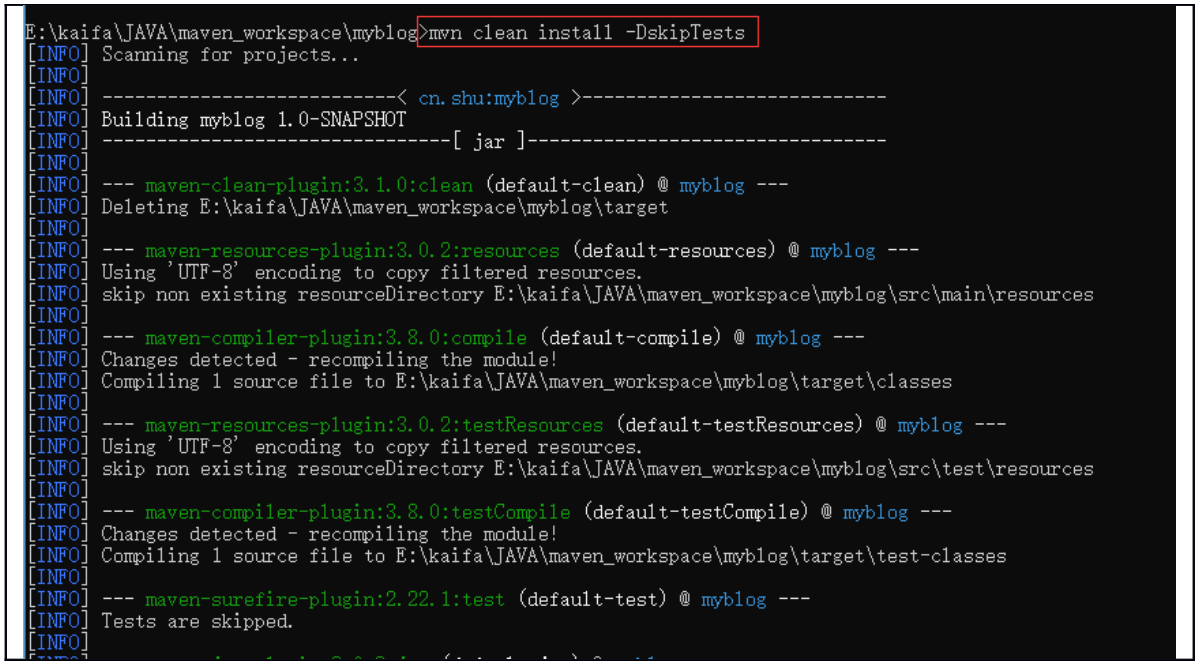
```
E:\kaifa\JAVA\maven_workspace\myblog>mvn clean package -DskipTests
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< cn.shu:myblog >-------------------------
[INFO] Building myblog 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ myblog ---
[INFO] Deleting E:\kaifa\JAVA\maven_workspace\myblog\target
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:resources (default-resources) @ myblog ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\kaifa\JAVA\maven_workspace\myblog\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ myblog ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to E:\kaifa\JAVA\maven_workspace\myblog\target\classes
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:testResources (default-testResources) @ myblog ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory E:\kaifa\JAVA\maven_workspace\myblog\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ myblog ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to E:\kaifa\JAVA\maven_workspace\myblog\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ myblog ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:3.0.2:jar (default-jar) @ myblog ---
[INFO] Building jar: E:\kaifa\JAVA\maven_workspace\myblog\target\myblog-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

9 mvn install (安装)

将项目的 jar 包作为库的结构生成在本地。

作为库的资源，放到远程库使用，必须满足库资源的结构，当某个项目 jar 包打包完成，想要放到远程库给别人连接读取使用，必须先经过本地库的安装过程，将项目的 jar 包作为库的结构生成在本地，才能进行上传发布到远程私服。

通常使用组合命令



10 mvn deploy (发布)

当一个 maven 项目进行安装之后，本地库具备了该项目的所有库资源结果（maven 一切皆资源）。可以通过发布命令，将项目的资源上传到远程私服。

目前没有远程私服，没有设置远程私服的登录名密码，权限地址等等，发布不能成功

五 Idea 中使用 maven

文件夹中直接开发 maven 项目比较麻烦

1 IDEA 中配置 maven

配置 maven 家目录、setting.xml 路径、本地仓库

Setting(当前项目)和 Setting for New Projects(新项目)都需要设置

1.A Setting

IDEA 中,通过 File 菜单我们可以从 SETTING 中修改 MAVEN 的配置,基本主要包括 Maven 的家目录,settings.xml 文件加载,本地库的使用.虽然我们已经在安装的 maven

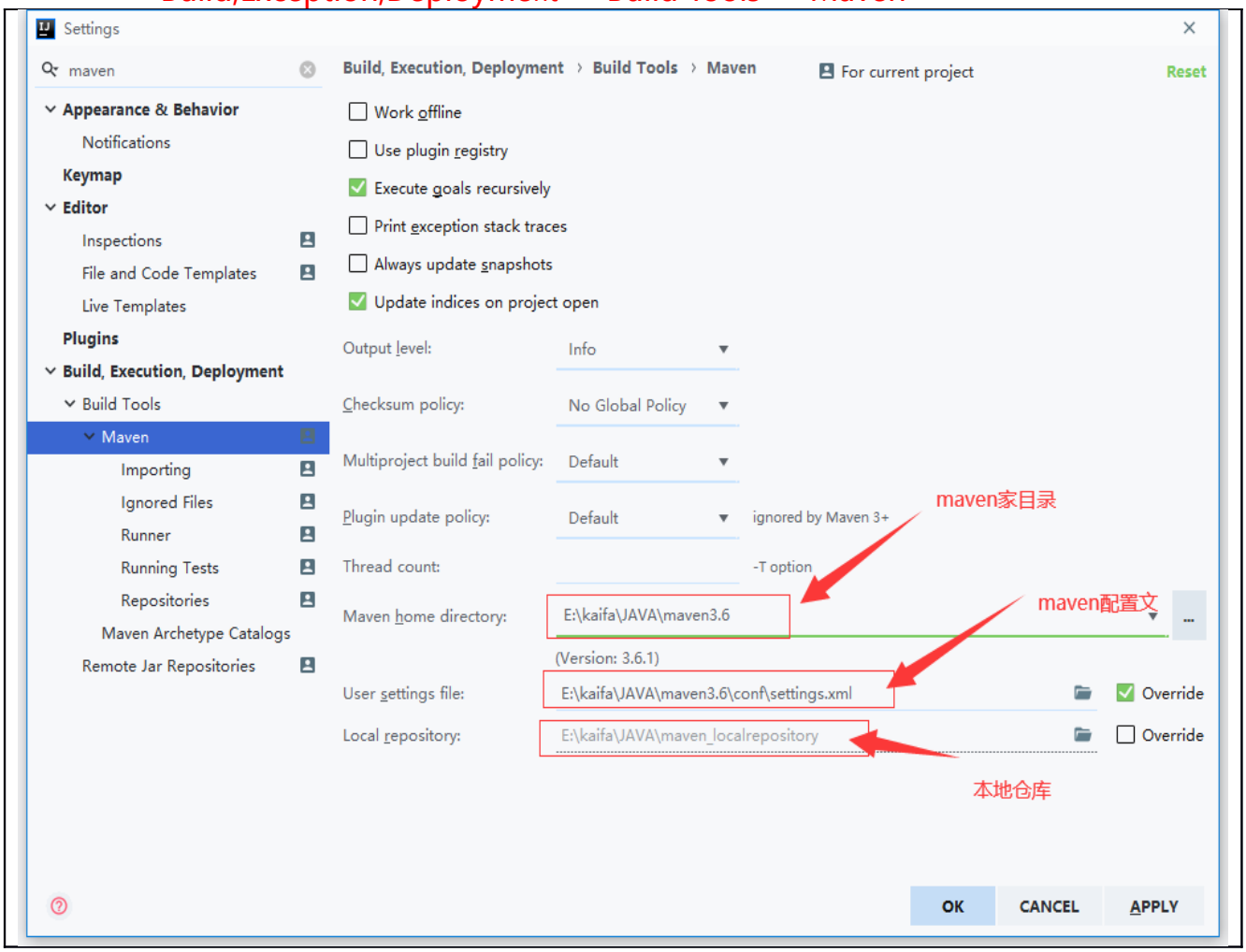
中完成了本地库配置,但是 IDE 工具中默认也有一套环境,可以单独使用,但是我们最好和自己的环境配置成一样的.方便后续的使用.

找到菜单中 FILE 里的 SETTINGS

File-->Settings

找到 MAVEN 配置项

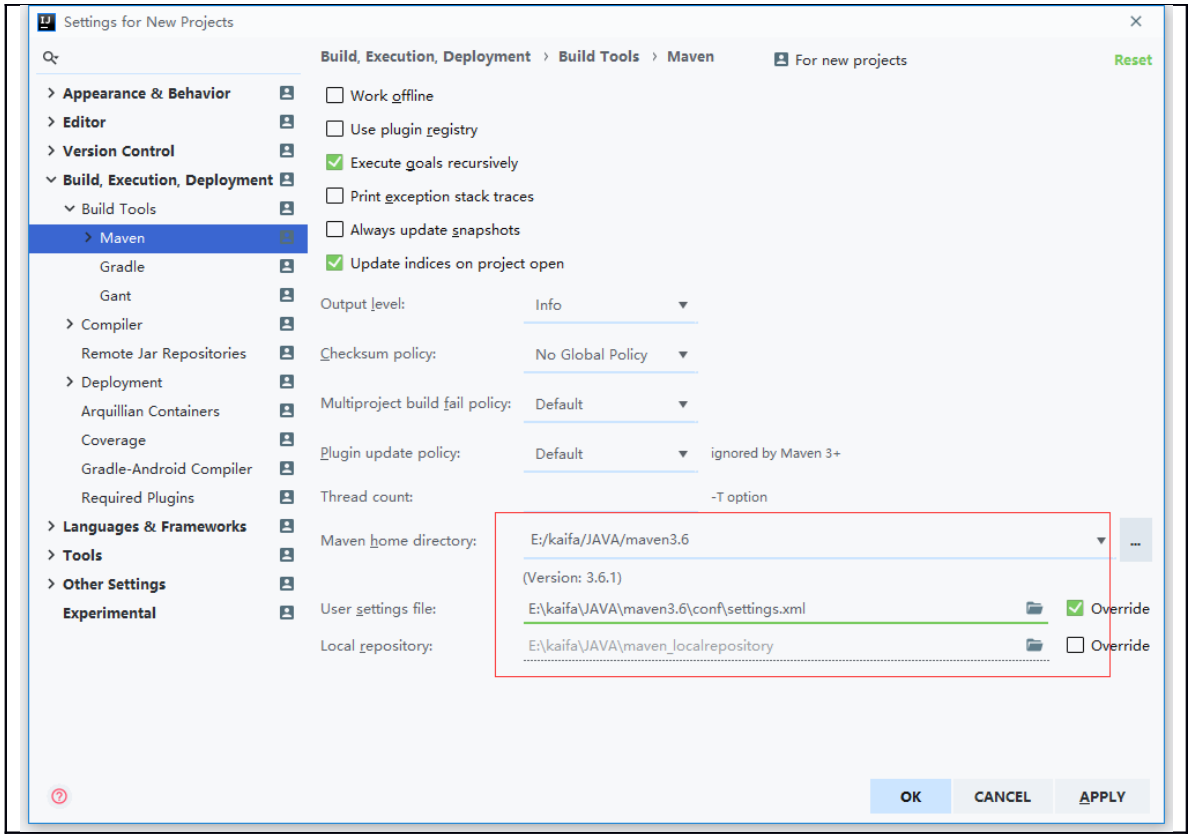
Build,Exception,Deployment-->Build Tools-->Maven



1.B Other Settings

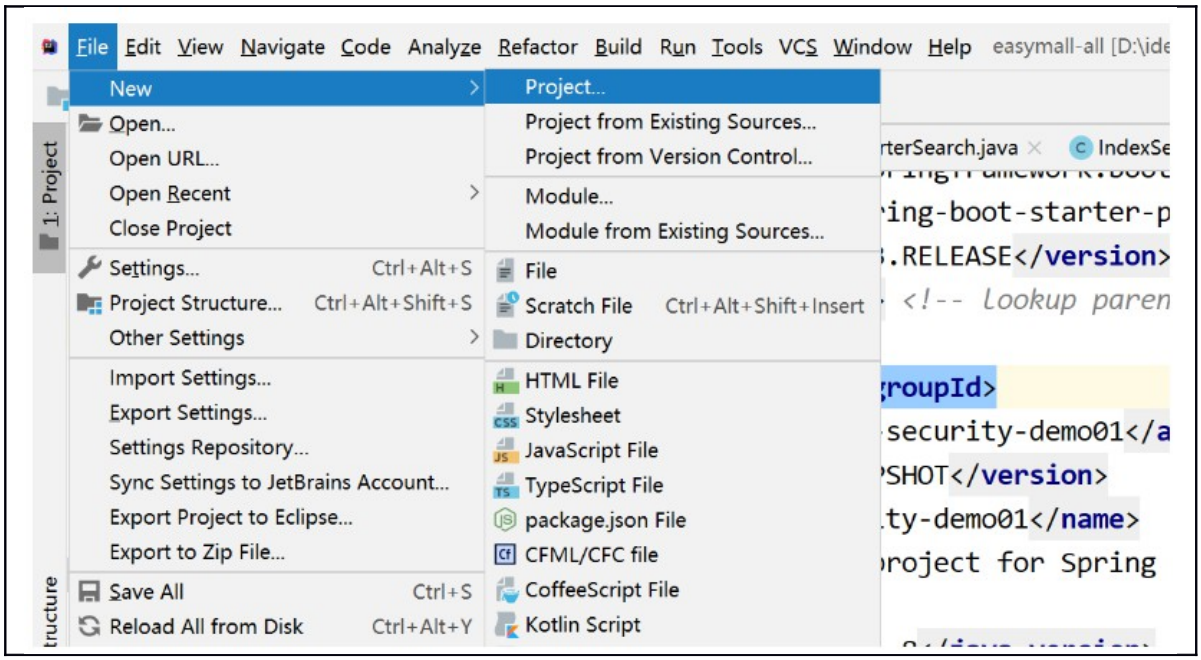
有的时候即使在 Settings 中配置了 maven,我们新创建的 Maven 工程还是会让我们重新选择一下 Maven 的配置,所以继续我们在 Orther Settings 中找到 Settings for new project.

在同样的 Maven 下重新配置一些根目录,配置文件和本地库,然后我们就可以创建 maven 项目了.



2 idea 创建 maven 项目

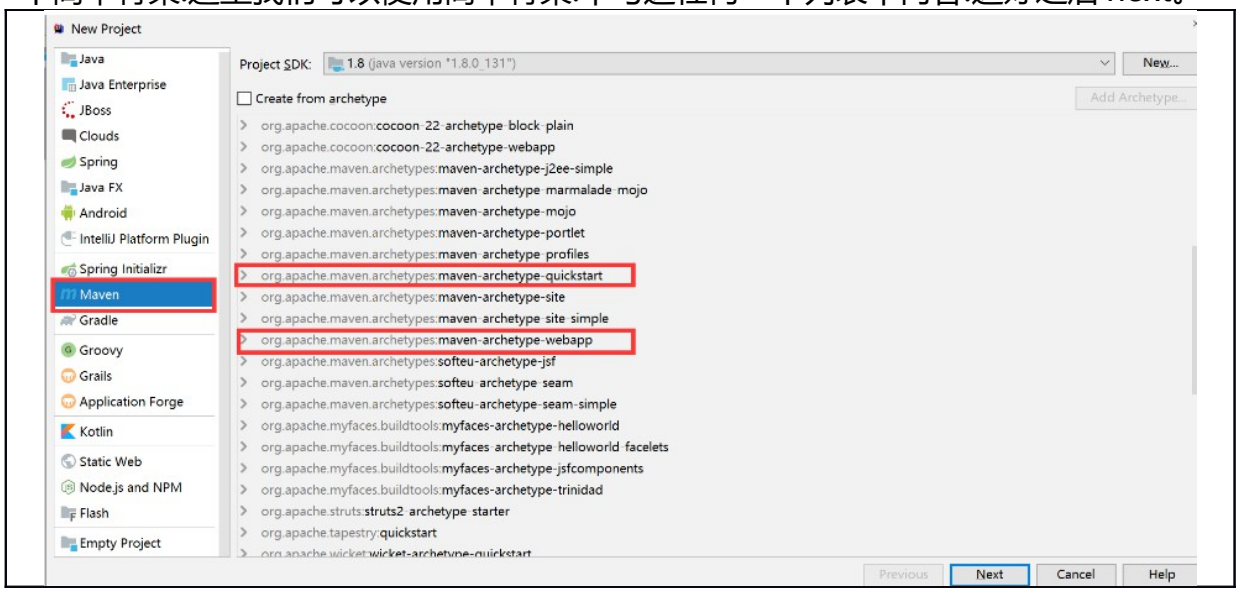
2.A File-->New-->Project.



2.B 选择 MAVEN 骨架

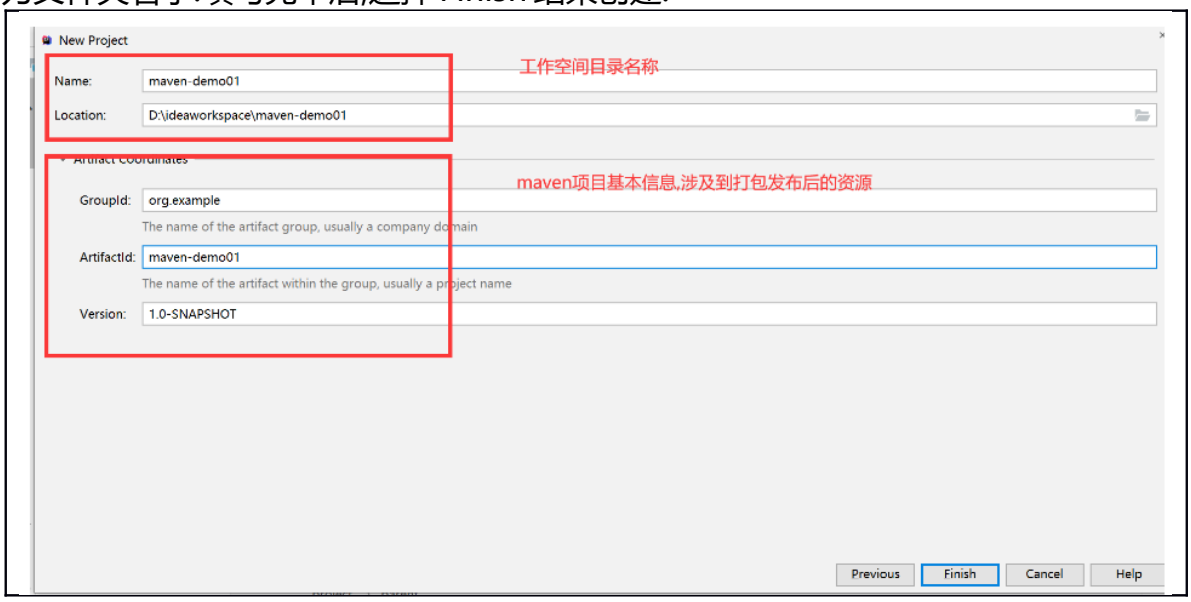
这里 Idea 的插件给我们提供了更多的选择,一般来讲开发一个 java 项目选择使用 maven-archetype-quickstart。

创建 web 项目使用 maven-archetype-webapp.如果什么都不选择,maven 会创建一个简单骨架.这里我们可以使用简单骨架.不勾选任何一个列表中内容.选好之后 next。



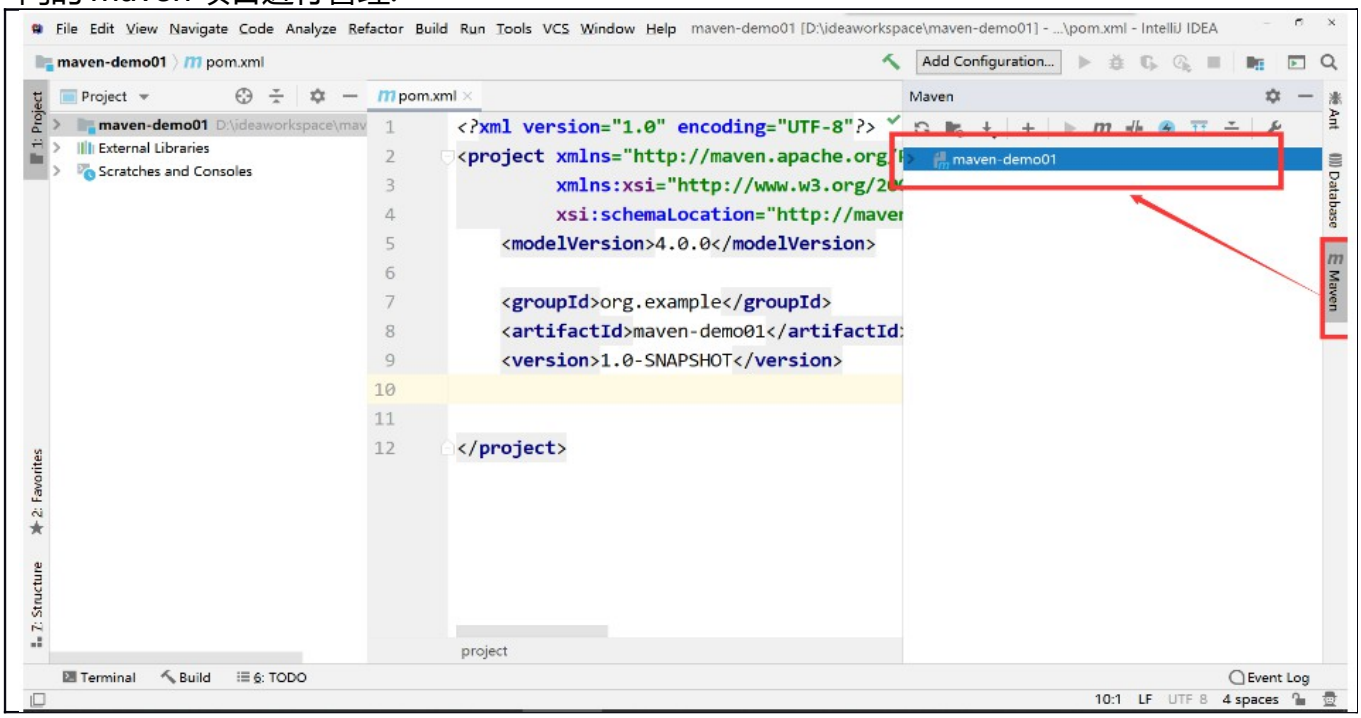
2.C 填写项目基本信息

这里我们填写下面的 maven 信息,项目的目录名称会自动选择 maven 的 artifactId 作为文件夹名字.填写完毕后,选择 Finish 结束创建.

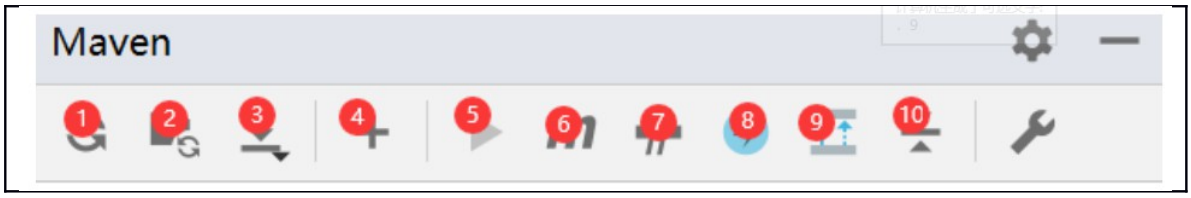


3 Idea 中的 maven 命令使用

当我们成功在 Idea 创建一个 Maven 项目之后,右侧会出现 maven 的管理窗口,针对当前工作空间的 maven 项目进行管理.



3.A maven 管理窗口按钮的作用



1. **Reimport All Maven Project 导入刷新**: 将当前工作空间中所有 maven 的 pom 文件重新导入,防止 ide 工具不认识项目中的 maven 元素
2. **Generate Sources And Update Folders For All Project 生成刷新**: 在编写了代码的情况下,有插件支持,可以将源码生成到 target 下.
3. **Download Sources and/or Document 下载源码和文档**: 对于 maven 使用的 jar 包可以已下载源码和文档
4. **add Maven Projects 添加 maven 工程**: 可 3 分以在当前窗口添加其他的 maven 项目来管理
5. **Run maven build 运行 maven 项目构建**: 可以对项目进行 build 创建,也是需要插件支持
6. **Execute Maven goal 运行 maven 命令**: idea 插件自动配置 maven 命令,如果想要运行自定义 maven 命令可以从这里进入.
7. **Toggle Offline Mode 强行离线模式**: 强制使用本地库资源不再联网获取资源.
8. **Toggle 'Skip Test' Mode 强制跳过测试**: 测试运行,测试代码编译都不会执行
9. **show dependencies**: 展开当前项目 maven 所有使用 jar 包资源

10. **collaps all 收起**：将展开的内容收起。

六 Maven 的 pom 文件

在每一个 maven 项目中,都存在一个 pom 文件,这是一个 maven 的核心配置文件,我们必须掌握其中的常用标签才能够使用好 maven

1 Pom 文件作用

POM(Project Object Model,项目对象模型),是 maven 管理项目的核心思想,所以每个项目中都有一个 pom.xml.而 maven 就是通过加载 pom.xml 来获取项目的全部信息,从而实现依赖,构建,测试,打包,生成报告的所有功能.

2 基本项目信息

在 maven 第一个案例项目中,我们使用了简单骨架创建功能,出现了 pom.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>
    <groupId>org.example</groupId>
    <artifactId>maven-demo01</artifactId>
    <version>1.0-SNAPSHOT</version>
</project>
```

在这个 pom.xml 中仅有该项目的基本信息,包括

- modelVersion:描述这个 POM 文件是遵从哪个版本的项目描述符,目前我们正在使用的 maven 都是 4.0.0
- groupId:当前项目的全路径名称 ， 一般域名倒写，例如：cn.shu
- artifactId:当前项目的模块名称 ， 项目名称、模块名称等
- version:当前项目的版本

所以说,无论创建什么项目这些内容是必须存在的,否则无法描述项目,maven 也无法管理项目

七　Maven 的依赖

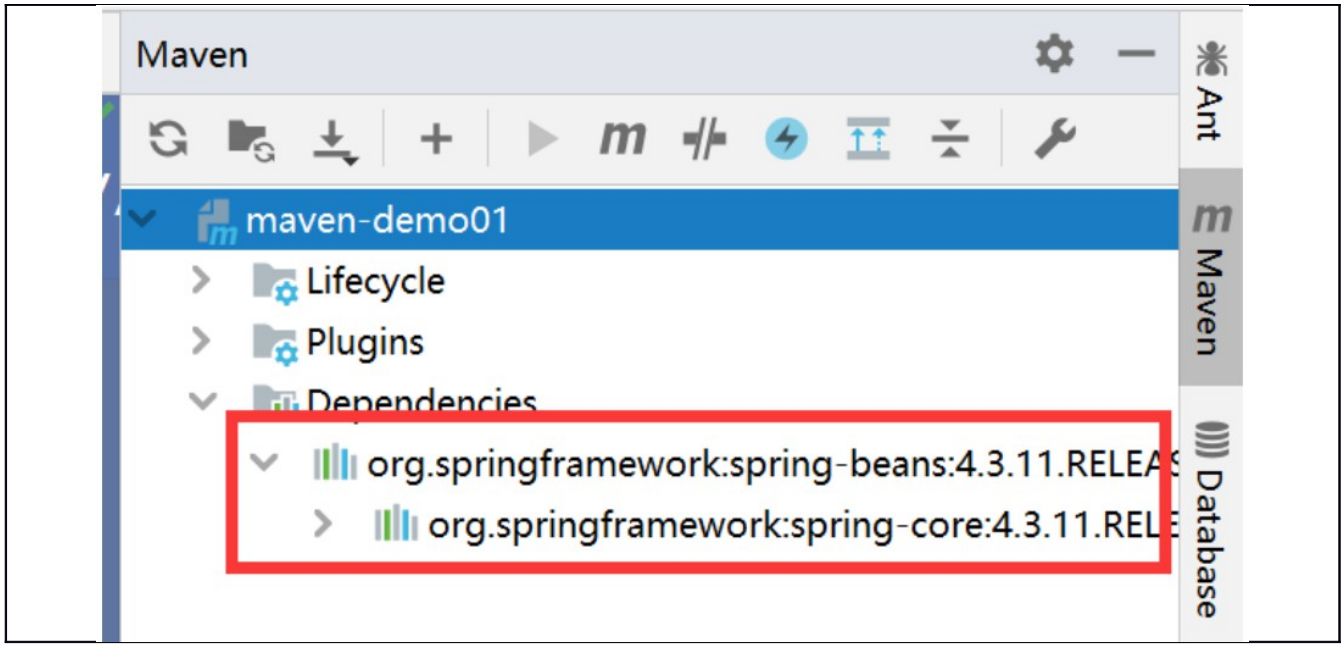
当一个项目开始编写代码时,需要引入很多的 jar 包资源,这时可以利用 maven 的依赖标签来使用和管理这些 jar 包资源（和之前导入 jar 包有区别）

1　dependencies

在 pom 文件中,可以使用 dependencies 标签,其中包含若干个 dependency 标签来实现对依赖资源 jar 包的直接使用.比如当我们想开发与 spring-beans 有关的代码时,可以引入这个依赖.

```
<?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>
    <groupId>org.example</groupId>
    <artifactId>maven-demo01</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <!--spring-beans-->
        <dependency>
            <!--groupId:对应一个文件夹结构 org/springframework-->
            <groupId>org.springframework</groupId>
            <!--artifactId:对应一个同名文件夹 spring-beans-->
            <artifactId>spring-beans</artifactId>
            <!--version:对应一个同名文件夹 4.3.13.RELEASE-->
            <version>4.3.11.RELEASE</version>
        </dependency>
    </dependencies>
</project>
```

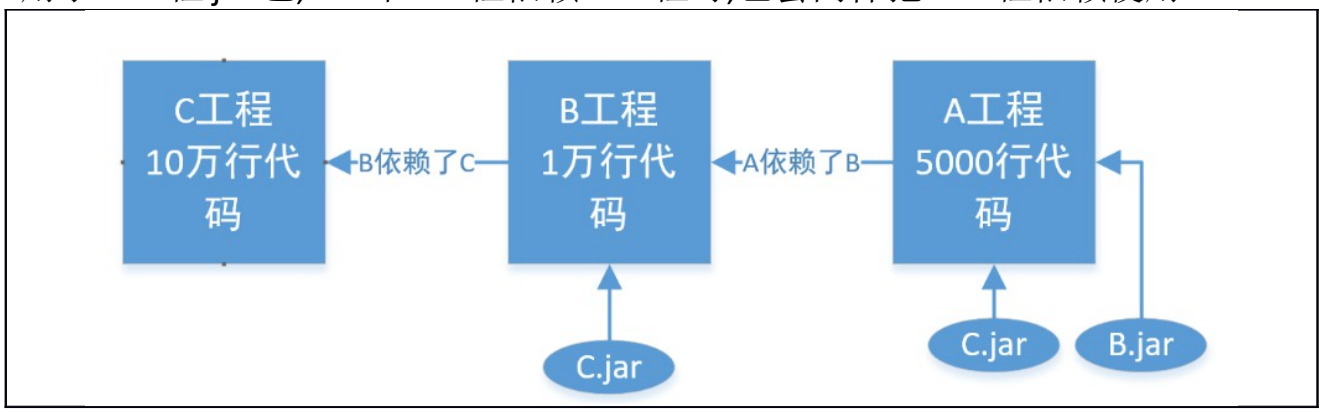
引入依赖的作用就相当于是在项目里我们导入了这个 jar 包,这时你就可以开始使用它了.我们还可以在右侧 maven 管理窗口看到这个依赖的引入.



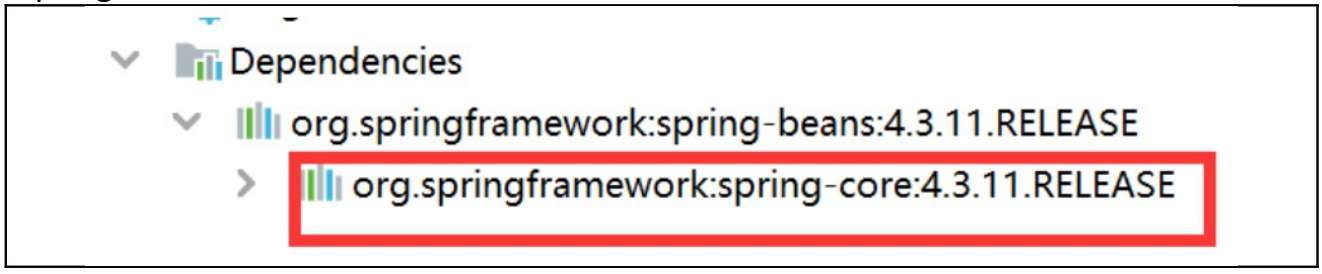
MAVEN 提供了一个查询搜索的库的网站 https://mvnrepository.com/去搜索你想要的依赖详细信息。

2 依赖的传递性

我们要知道一个依赖的 jar 包,其本质也是一个 java 工程,所以在开发这个 java 工程时也会用到其他的 jar 包资源,在 maven 中,这种相互的依赖关系是可以进行传递的. 比如 A 工程开发需要 B 工程的 jar 包支持, 而 B 工程又在开发时使用了 C 工程 jar 包, 当 A 工程依赖 B 工程时,也会同样把 C 工程依赖使用.



maven 工程利用依赖的传递性可以实现很多简单的依赖配置,不需要重复复杂的实现.比如案例中 spring-beans,不仅使用了 spring-beans 还传递过来了 spring-core



3 去除传递性(<exclusions>)

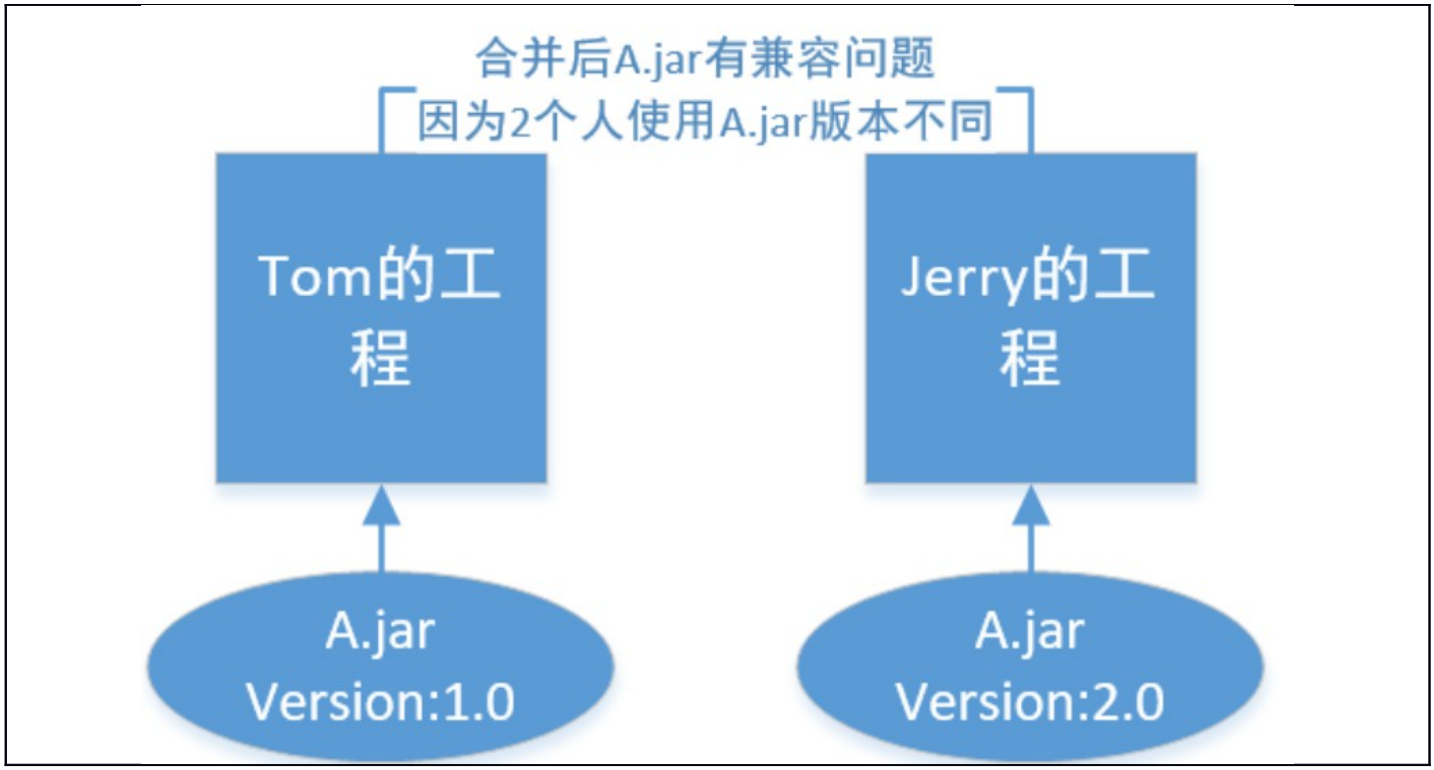
有的时候我们本来依赖的一个 jar 包资源,而这个 jar 包资源传递给了当前 工程很多不需要的依赖,这时为了最简配置依赖资源,我们可以从这个依赖中将不需要的去除,使用<exclusions>标签实现.

```
<dependencies>
        <!--spring-beans-->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
            <version>4.3.11.RELEASE</version>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework</groupId>
                    <artifactId>spring-core</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
```

上例中，引入了 spring-beans 包，但传递了 spring-core,此时通过 exclusions，去除不需要的 jar 包看到标签的结构,是 exlusions 包含了若干个 exclusion 所以当传递的多个依赖都需要去除时,可以在其中包含多个 exclusion

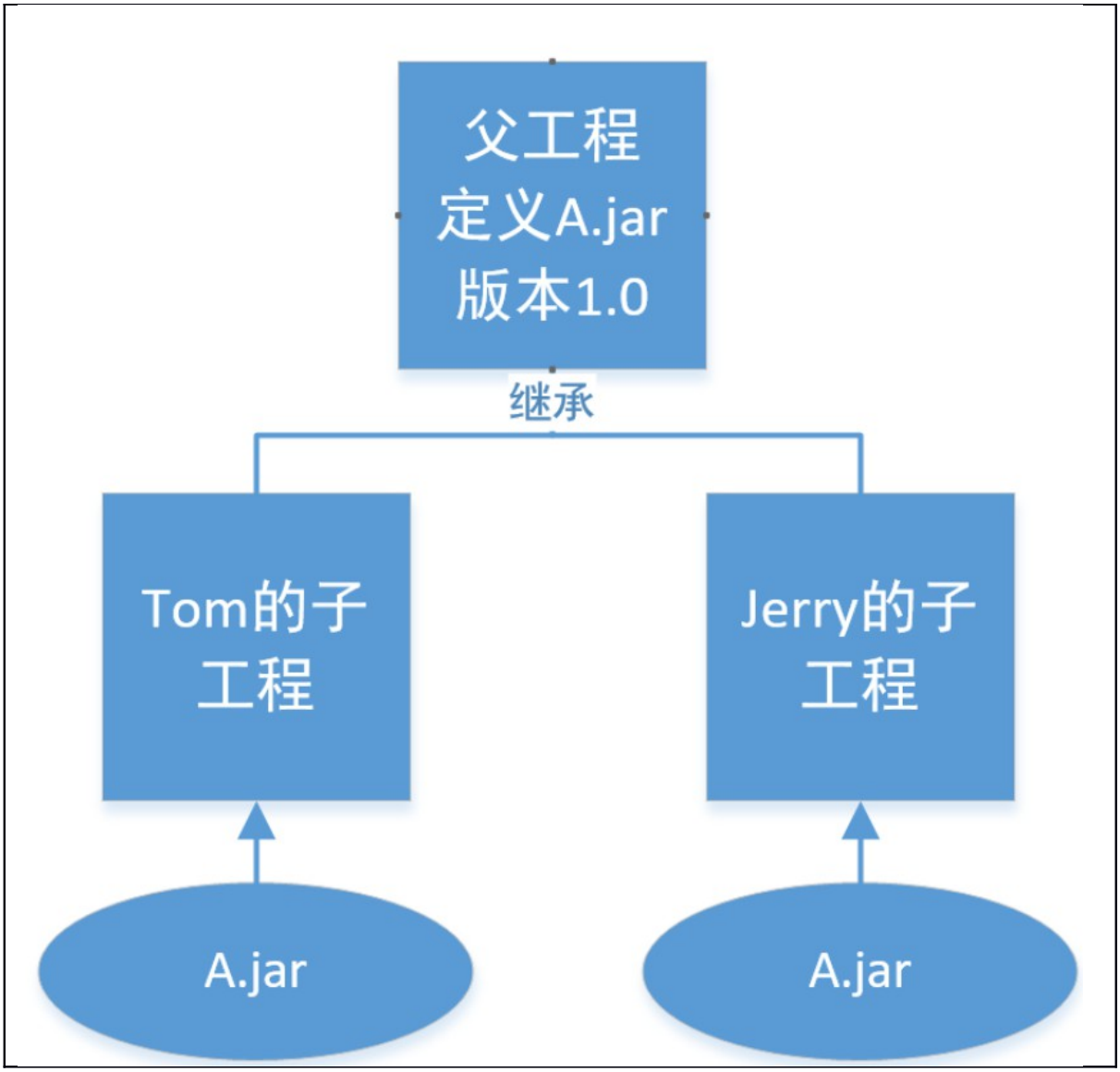
八 Maven 的继承

我们先来看一个应用场景,使用 maven 将一个项目分给多个人并行开发,最终多人开发的多个工程整合成一个项目系统使用,这时就需要考虑不同开发人员在开发过程中使用的资源版本(例如 jar 包版本)问题,一旦出现版本不兼容会造成测试成本过高,项目不能及时上线的严重问题.



1 继承的意义

为了统一一个项目多模块并行开发的资源版本,maven 出现了继承的功能,可以将一个大型项目分为父工程和子工程,其中父工程的唯一作用就是定义所有子模块工程的资源版本



上图中,还是 2 个模块开发者使用 A.jar,由于继承了父工程,定义了 A.jar 的版本为 1.0,所以子工程的 A.jar 不需要在添加版本的描述,直接就使用的是 1.0 版本的 A.jar

2 继承的实现

2.A 创建父工程

保证父工程的 pom.xml 文件中定义的打包 packaging 标签的类型是 pom,一个 java 项目默认是 jar 类型,web 项目默认是 war 类型,但是父工程必须是 pom 类型

```
<?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>
    <groupId>org.example</groupId>
```

```
    <artifactId>maven-parent</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>pom</packaging>
</project>
```

2.B 创建子工程

在子工程中使用 parent 标签,定义 groupId artifactId version 指向父工程.这样父工程的一些资源标签就可以被子工程使用了

注意:由于 idea 每个项目单独使用一个 workspace 工作空间所以父工程想要被继承,需要先安装到本地库 install,要不然就在 idea 中使用聚合(idea 的聚合)在父工程下直接创建 child 工程(module),这里我们选择第二种.

```
<?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">
    <parent>
        <artifactId>maven-parent</artifactId>
        <groupId>org.example</groupId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>maven-child</artifactId>
</project>
```

3 继承的内容

通过上述案例我们实现了一个父工程下有一个子工程继承的项目结构,那么子工程能从父工程继承哪些内容呢?本质上继承的就是标签

3.A properties

1 maven 项目中可以使用 properties 定义一些当前项目使用的变量名称,例如:

```
<properties>
    <java.version>1.8<java.version>
</properties>
```

子工程使用：这样依赖可以在当前项目中使用 java.version 来定义一些内容,例如:

```
<dependency>
    <groupId>cn.tedu<groupId>
    <artifactId>maven</artifactId>
    <version>${java.version}</version>
</dependency>
```

这种标签里的属性可以被子工程继承,子工程同样可以这样使用

3.B dependencies

一个项目的依赖,可以被子工程继承,而且是子工程无论用的到用不到依赖资源都会继承,虽然可以在父工程实现这样的结构,但是显然不是最合理的（因为子工程不需要也会继承）.

3.C dependencyManagement

这个标签用的最多的还是在父工程中实现依赖的版本管理,从而子工程不会被强制使用这些依赖,只会继承依赖管理中的版本号,这样可以留给子工程选择的空间

例如:

- 父工程:在依赖管理中添加 spring-beans 依赖

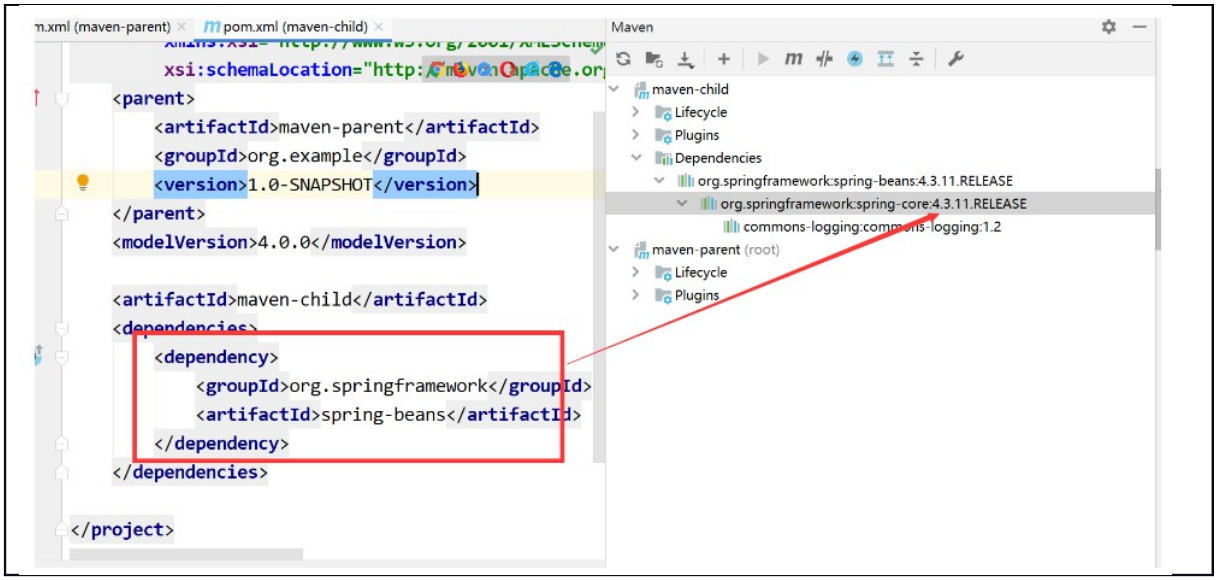
```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-beans</artifactId>
      <version>4.3.11.RELEASE</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

- 子工程

在依赖中只需要填写 groupId artifactId,version 直接继承自父工程

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-beans</artifactId>
  </dependency>
</dependencies>
```

这样相当于父工程管理的 4.3.11 就限制了子工程版本的内容



这就是父子继承的 maven 结构,一旦某个工程继承了一个父工程,那么他就会从父工程中获取很多已经定义好的资源,尤其是版本.

九　Maven 的聚合

当项目特别多时,我们总是需要对每一个 maven 项目进行编译,测试,打包,安装,发布的操作命令都会执行一次,这样是非常复杂的,可以利用 maven 的聚合来解决

1　聚合的意义

当使用父子继承关系来开发时,一定会存在大量的子工程,这时对于子工程可以使用 maven 的聚合工程来实现一键执行命令.所以聚合就是统一命令的操作

例如我编译父工程，子工程同时编译

2 聚合的实现

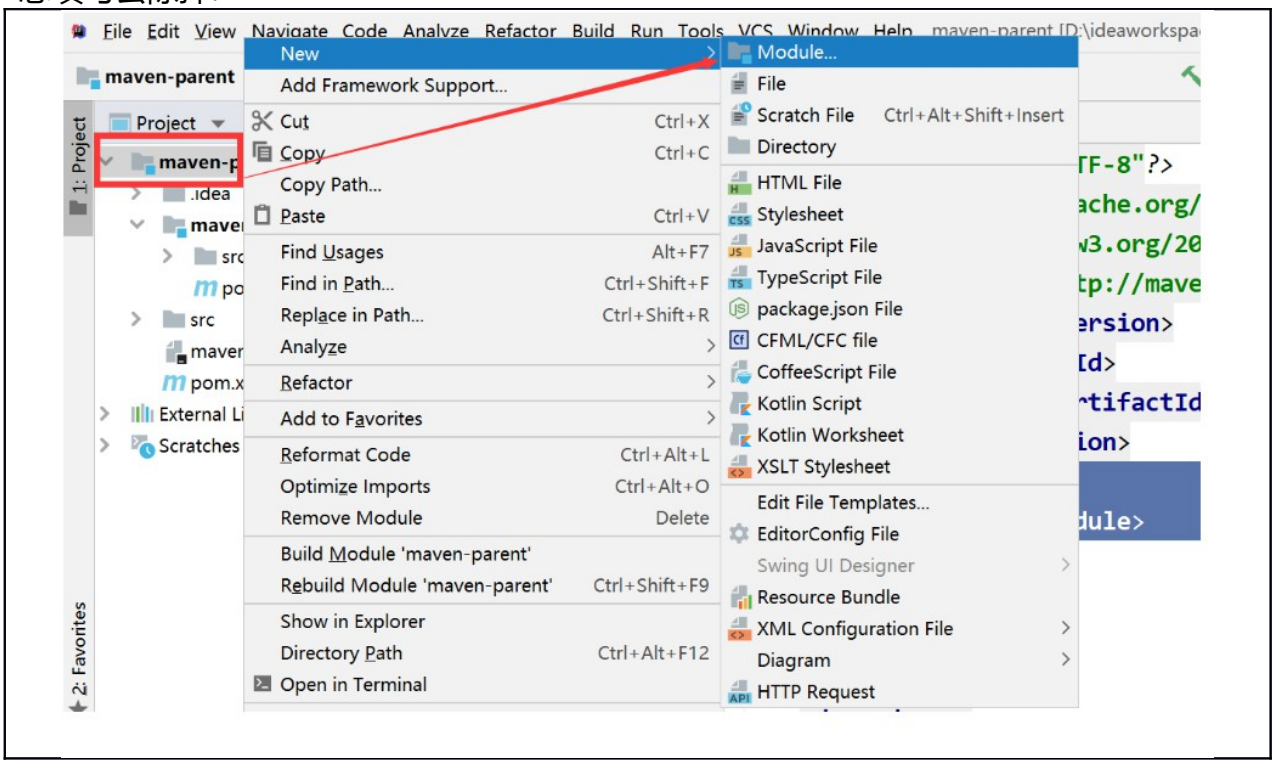
如果我们想要使用一个工程,通过对这个工程命令执行,就能够同时执行其他多个工程的同一个命令,这个工程就是聚合工程,在聚合工程中我们只需要添加一个<modules>标签,配置若干个<module>标签指向其他工程路径就可以完成.

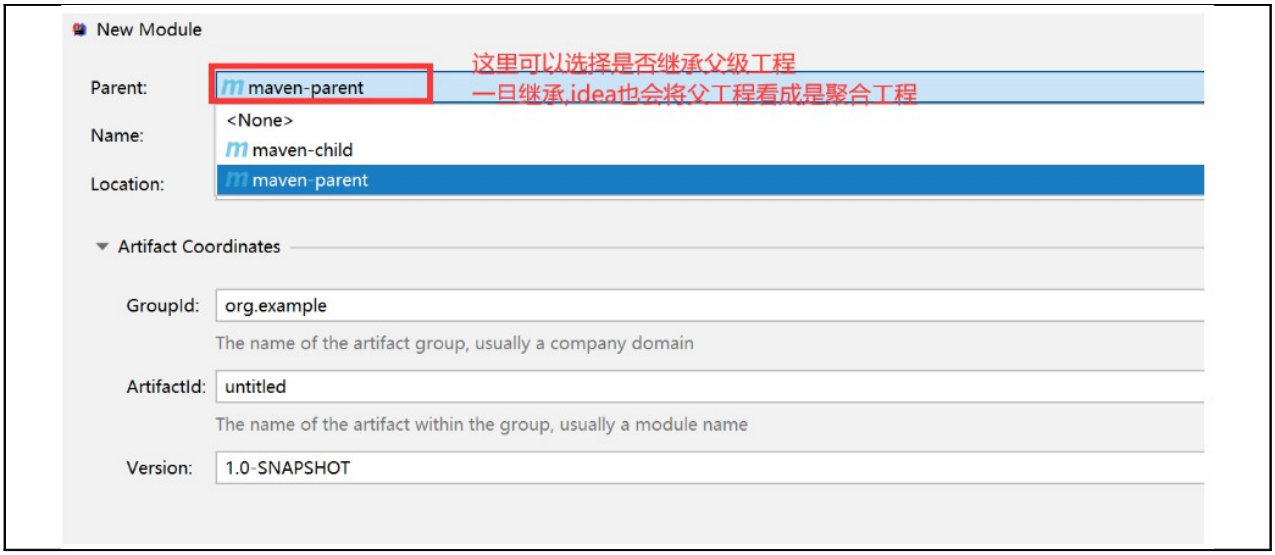
```
<modules>
  <module>maven-child</module>
</modules>
```

这个标签的含义就是,当前工程可以管理 maven-child,其项目目录是个相对路径,相对当前聚合工程的根目录.也可以使用绝对路径(d://project),或其他路径(../project)表示聚合.

3 Idea 中的聚合

由于 idea 存在聚合工程的概念,所以一旦使用 maven 工程右键 new 一个 module 会自动在当前工程添加聚合标签<modules>和<parent>标签,这些也可以通过构建项目时的基本信息填写去除掉.





我们在今后的开发过程中一般都会通过在一个父级工程中创建 module 聚合子工程来管理所有的代码项目结构.