大家好，上一小节中我们主要对Linux常用的命令做了介绍。本小节中，我们对Maven相关知识点以及其常用命令进行交流与学习。 2

Maven是什么？（掌握）

Maven 是一个跨平台的强大构建工具，可以实现**自动化构建过程**，从**“清理、编译、测试、生成报告、打包和部署”**都可以使用成熟的插件，**通过简单的命令**实现，避免了重复的构建过程。

Maven 不仅是一个优秀的项目构建工具，还是一个**依赖管理工具**，它提供了强大的中央仓库，能够帮助我们自动下载依赖。

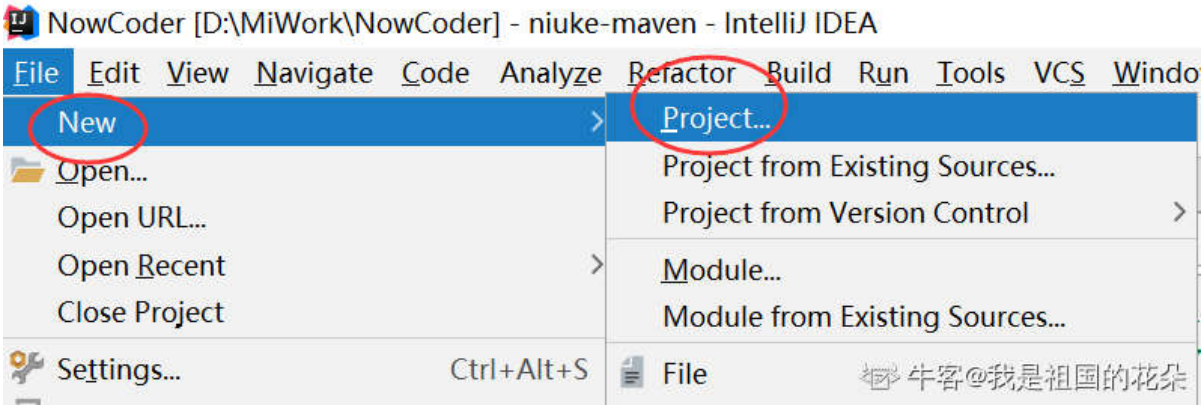
Maven 提供了一个很好的解决方案。Maven 通过使用GroupId和ArtifactId来标识每一个构件（依赖），也就是通过一个坐标系准确地定位到了一个 Java 类库。**Maven 的中央仓库**几乎可以找到任何流行的开源类库，通过在项目 POM 文件中配置，都可以免费下载。

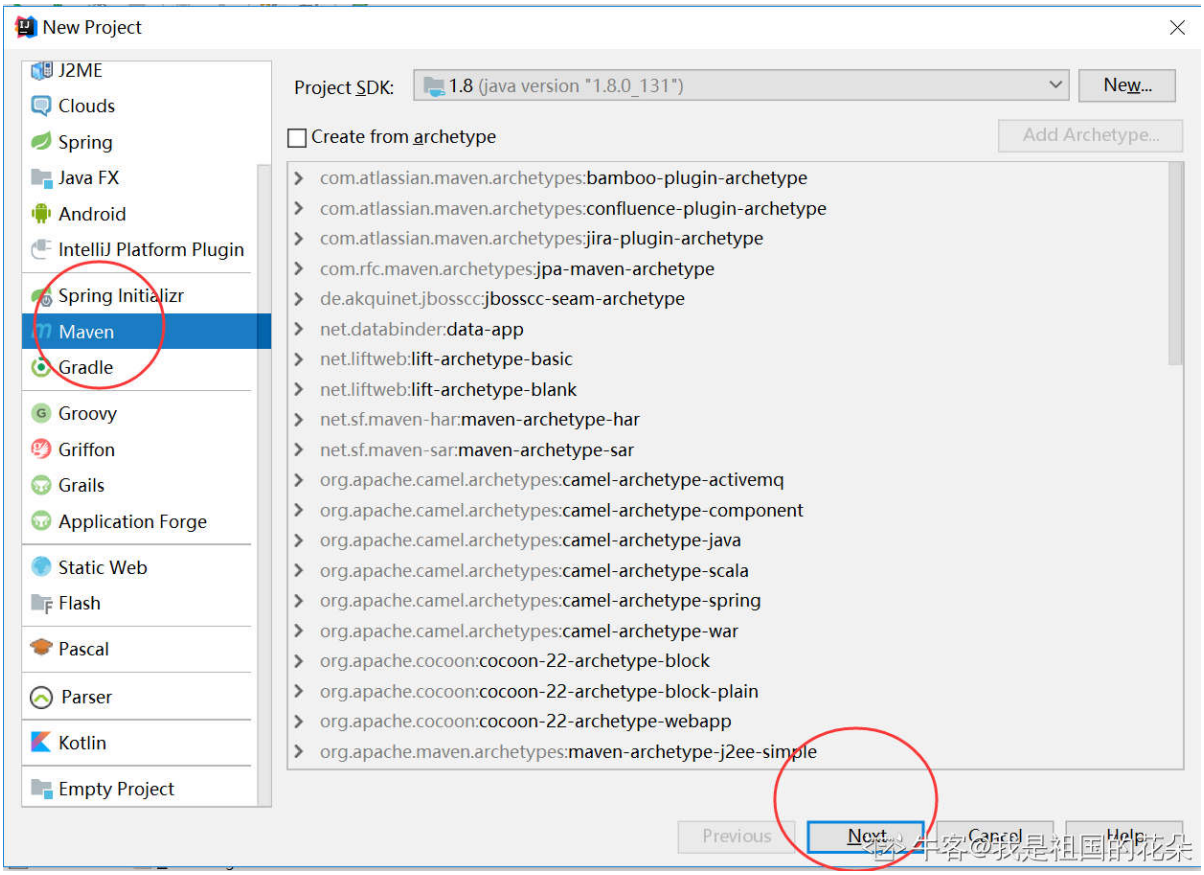
我们来总结下**Maven的作用：**

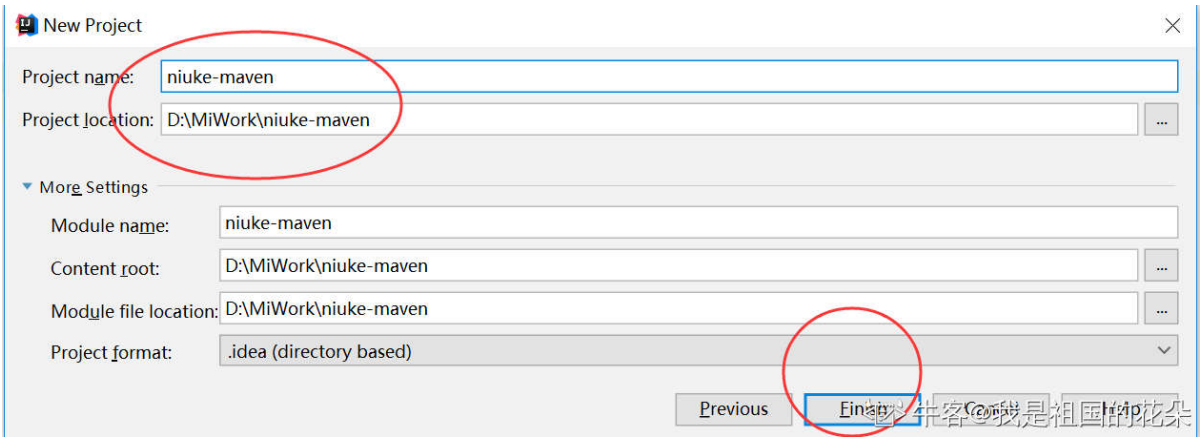
> **Maven 可以帮助我们自动构建项目，减少重复劳动；**
> **Maven 可以帮助我们自动下载依赖，减少手工劳动。**

接下来，我们先来创建一个Maven项目吧。

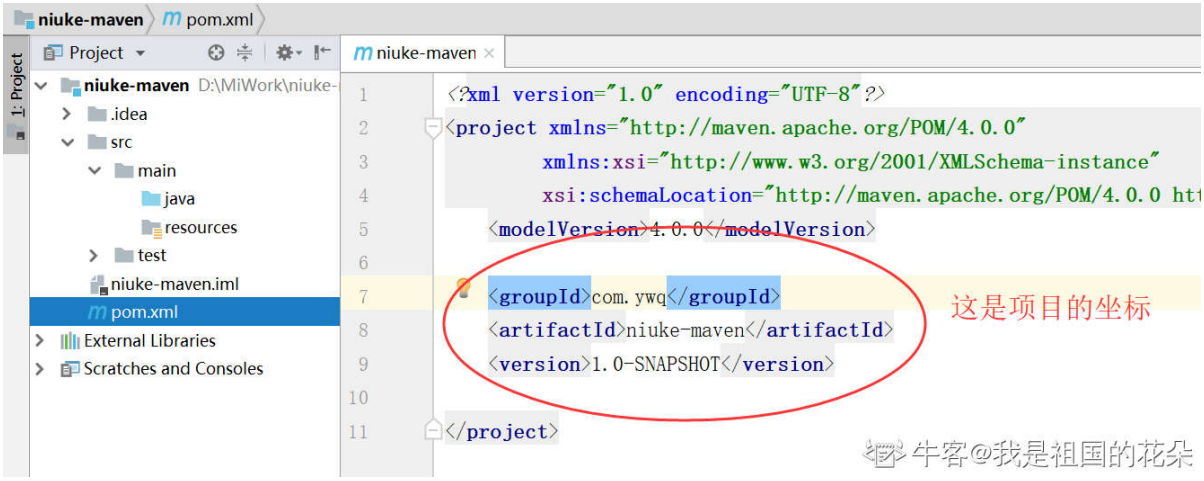
创建Maven项目：（使用IDEA来创建）







Maven项目创建之后，目录结构如下所示：



前面，我们说了Maven可以帮助我们自动下载依赖，，通过在pom.xml文件中配置所需的依赖，Maven可以自动帮助我们从仓库中获取依赖，并且下载导入项目中。如下所示：

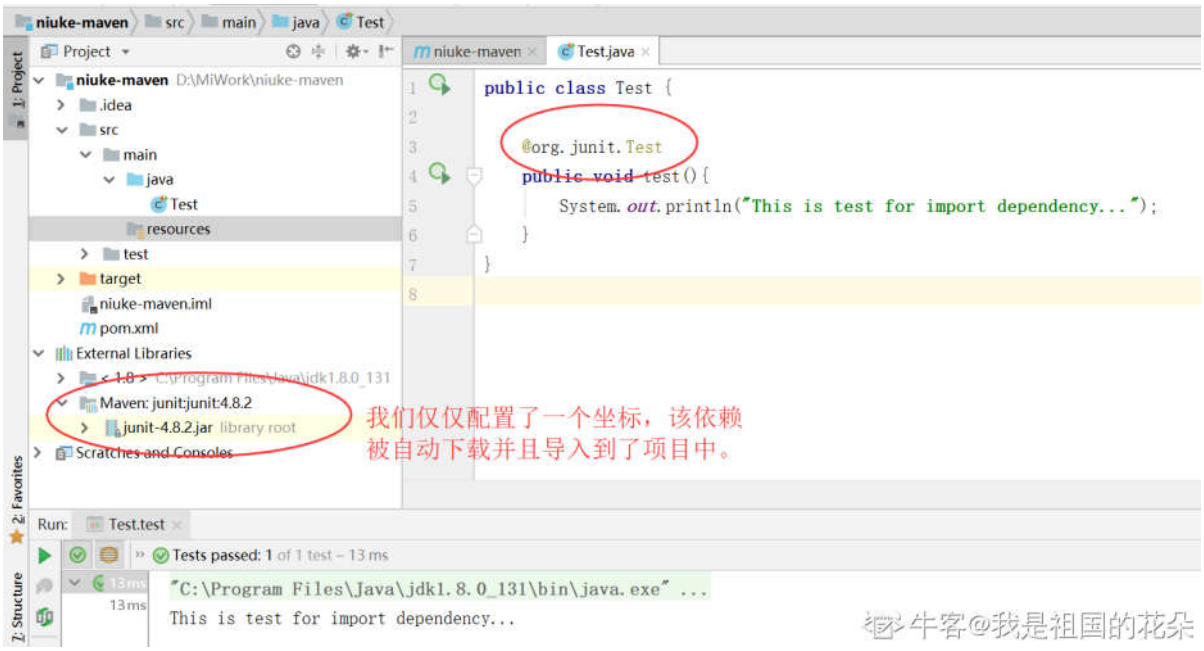
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.ywq</groupId>
    <artifactId>niuke-maven</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.2</version>
        </dependency>
    </dependencies>

</project>
```

然后，我们可以使用单元测试，运行之后，效果如下图所示：

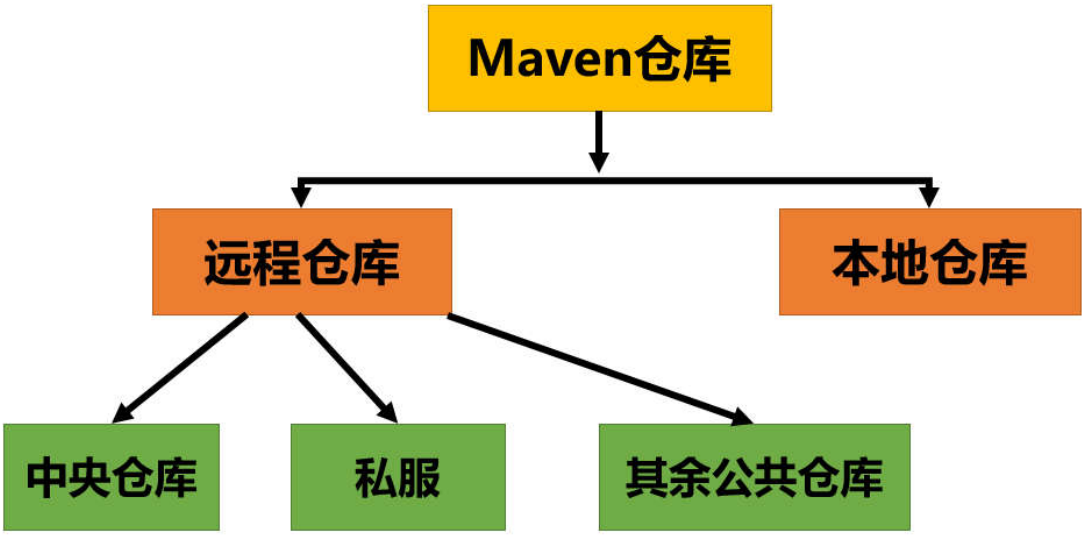


看到这里，大家有没有豁然开朗的感觉？是否还能想起自己曾经“满大街”去寻找下载依赖，然后手动导入到项目中，最后还需要build一样，出现一个像奶瓶一样的包才算是完成了一个依赖包的引入的痛苦？

这就是Maven给我们带来的好处。当然了，聪明的你肯定会有疑问，**当我们在pom中引入了依赖的坐标之后，Maven是如何去寻找依赖的？或者说是去哪里寻找并且下载依赖包的呢？**别急，听我慢慢道来。

Maven的仓库

Maven 的仓库分为本地仓库和远程仓库，远程仓库又分为中央仓库、私（si）服及其它公共库。



Maven仓库示意图：

- **本地仓库：**就是 Maven 在本地（我们的计算机上）存储构件（依赖的 JAR 包等）的仓库，默认是在用户的.m2/repository/目录下。
- **远程仓库：**分为中央仓库、私（si）服及其它公共库。用户是在和私（si）服打交道，包括上传和下载构件。当私（si）服满足不了我们的下载构件需求时，私（si）服会和中央仓库或者其余公共仓库交互，将用户需要的构件缓存在私（si）服仓库中。

什么是私（si）服?

> **私（si）服是一种特殊的远程仓库**，是在局域网内的仓库服务，私（si）服代（dai）理广域网上的远程仓库，供局域网内的 Maven 用户使用。

什么是中央仓库?

> Maven 提供了一个中央仓库，其地址为：http://repo.maven.apache.org/maven2 ，该仓库包含了绝大多数流行的开源 Java 构件，以及源码、作者信息、SCM 信息、许可证信息等。据不完全统计，每个月中央仓库大概会接受全世界 Java 程序员大概 1 亿次访问，其重要性不言而喻。

好了，了解了Maven仓库的概念，你依然不理解其为什么可以自动取下载依赖并且导入到项目中。那么，我们再来看下Maven的配置文件settings.xml文件吧。

settings.xml文件

当我们下载安装好 Maven 时，在其安装目录的 conf 下存在一个 settings.xml 的配置文件，这是一个全局的 Maven 配置文件，为了不影响这台计算机上的其他用户，一般选择将该文件copy到 ~/.m2/ 下边，作为一个用户层面的配置文件。

settings.xml 文件中主要包含以下的元素：

- **localRepository**：本地仓库的目录。默认是用户目录下面的 .m2/repository 目录。
- **interactiveMode**：表示是否使用交互模式，默认是 true；如果设为 false，那么当 Maven 需要用户进行输入的时候，它会使用一个默认值。
- **offline**：表示是否离线，默认是 false。这个属性表示在 Maven 进行项目编译和部署等操作时，是否允许 Maven 进行联网来下载信息等。
- **mirrors**：**定义一系列的远程仓库的镜像**，用于缓解远程仓库的压力。
- **profiles**：用于指定一系列的 profile。
- **activeProfiles**：指定当前正在活跃的 profile。
- **servers**：表示当需要连接到一个远程服务器的时候需要使用到的验证方式。

2

通过settings.xml，我们可以进行配置，比如仓库地址的配置，授权验证配置等，这里给出一个比较完整的settings.xml配置文件，以供大家学习参考使用。

```
<?xml version="1.0" encoding="UTF-8"?>
    <settings>
    <mirrors>
      <mirror>
        <!--This sends everything else to /public -->
        <id>nexus</id>  // 镜像id
        <mirrorOf>*</mirrorOf>   // 表示代（dai）理所有仓库
        <url>http://nexus.d.xxx.net/nexus/content/groups/public</url>  // 
      </mirror>
    </mirrors>
    <profiles>
      <profile>
        <id>development</id>
        <repositories> // 构件的仓库
          <repository>
            <id>central</id>
            <url>http://nexus</url>
            <releases><enabled>true</enabled></releases>
            <snapshots><enabled>true</enabled></snapshots>
          </repository>
        </repositories>
       <pluginRepositories> // 插件的仓库
          <pluginRepository>
            <id>central</id>
            <url>http://nexus</url>
            <releases><enabled>true</enabled></releases>
            <snapshots><enabled>true</enabled></snapshots>
          </pluginRepository>
        </pluginRepositories>
      </profile>
    </profiles>
     <activeProfiles>
      <activeProfile>development</activeProfile> // 对于所有的POM，上边定义的
    </activeProfiles>
    <servers>
      <server>
        <id>archiva.internal</id>  // release版本的用户名和密码
        <username>yangwenqiang</username>
```

```
39          <password>pwdpwd</password>
40        </server>
41         <server>
42          <id>archiva.snapshots</id> // snapshot版本的用户名和密码
43          <username>yangwenqiang</username>
44          <password>pwdpwd</password>
45          </server>
46         </servers>
47       </settings>
```

读到这里，就比较清晰了。我们在pom文件中配置的依赖坐标，Maven通过去指定的仓库中去下载依赖，并且导入到了项目中，实现依赖的自动下载。接下来，我们看下Maven如何实现自动构建项目吧~

Maven的生命周期

Maven 的生命周期是对所有的构建过程进行的抽象和统一。在大量项目的构建过程中，Maven 总结出了一套高度完善的，易于扩展的生命周期，包括项目的清理、初始化、编译、测试、打包、集成测试、验证、部署和生成站点等构建步骤。

Maven 提供了三套独立的生命周期：**clean、default 和 site**，接下来我们分别介绍三套生命周期。

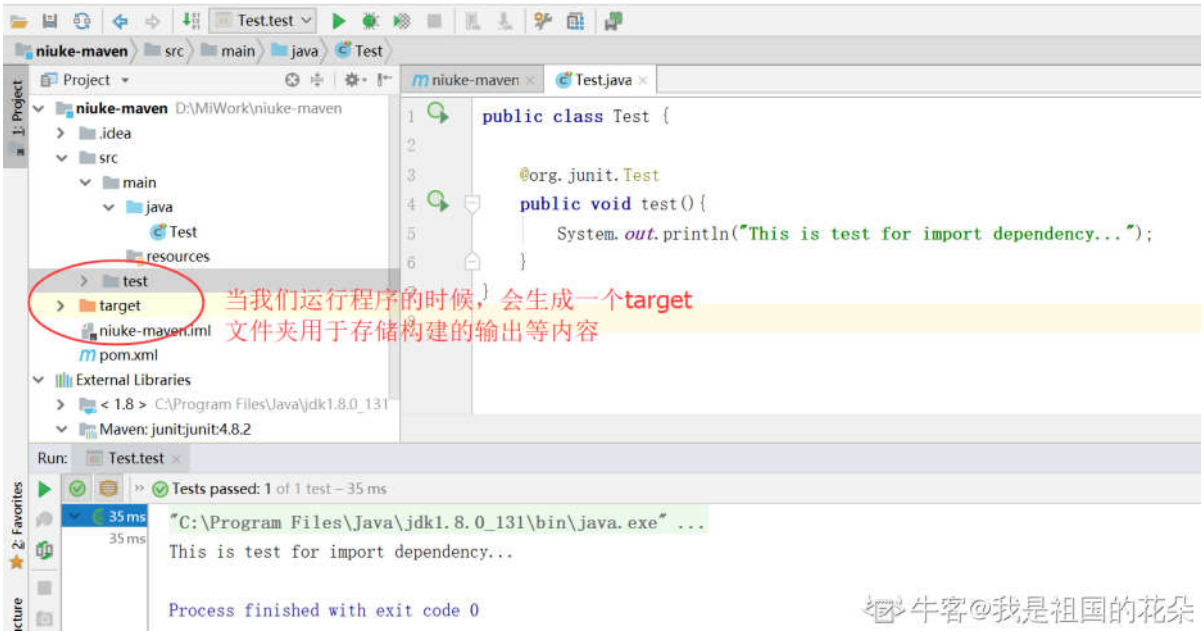
clean 生命周期

clean 生命周期的目的是清理项目，删除前一次构建在 target 文件夹下生成的各个 JAR 包等，它包含以下三个阶段：

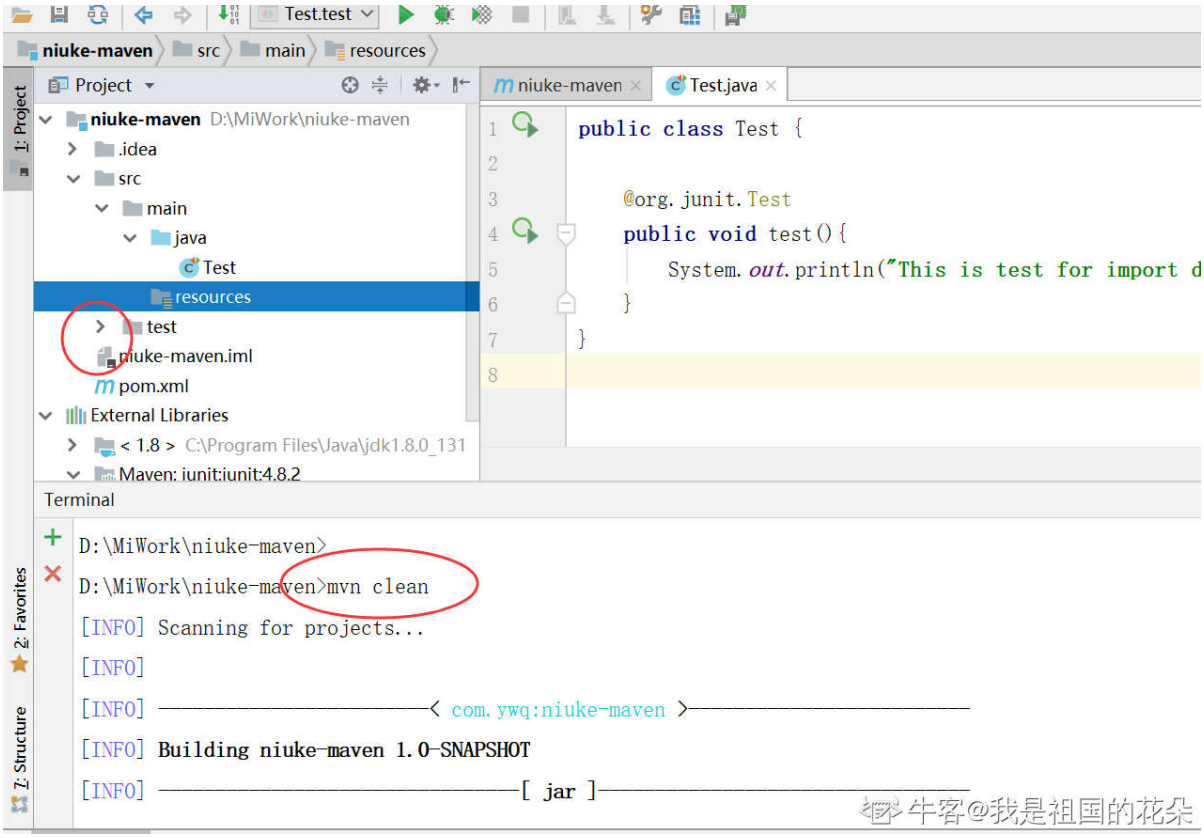
- pre-clean 执行一些清理前需要完成的工作
- clean 清理上一次构建生成的文件
- post-clean 执行一些清理后需要完成的工作

命令演示：

我们在命令行中输入 mvn clean 就是在调用 clean 生命周期的 clean 阶段，实际执行了 pre-clean 和 clean 阶段。



当我们执行了mvn clean之后，可以看到上次生成的target文件夹被删除清理掉了。



site生命周期

site生命周期的目的是建立和发布项目站点，Maven 可以给予 POM 所包含的信息，生成一个站点，方便团队交流和发布项目信息，其生命周期阶段包含：

- pre- site
- site 生成项目站点文档
- post-site
- site-deploy 将生成的项目站点发布到服务器上

site生命周期主要是用来生成站点，比如说当前项目的帮助文档站点等信息，我们简单了解即可，实际工作中一般不会使用到。

2

default 生命周期

default 生命周期**定义了真正构建项目中需要执行的所有步骤**，它包含的阶段如下：

- validate
- initialize
- generate-sources
- process-sources
- generate-resources
- process-resources
- **compile：编译项目的主源码**
- process-classes
- generate-test-sources
- process-test-sources
- generate-test-resources
- process-test-resources
- test-compile
- process-test-classed
- test：使用单元测试框架运行测试，测试代码不会被打包或部署
- prepare-package
- **package：接受编译好的代码，打包成可发布的格式，JAR/WAR 等**
- pre-integration-test
- integration-test
- post-integration-test
- verify
- **install：安装构件到本地仓库**
- **deploy：发布构件到远程仓库**

命令演示：

mvn test：调用 default 生命周期的 test 阶段，实际执行了 validate 到 test 阶段之间的所有阶段。

mvn clean package：调用 clean 生命周期的 clean 阶段和 default 生命周期的 package 阶段，实际执行了 pre-clean 和 clean 阶段和 default 生命周期 validate 到 package 阶段之间的所有阶段。

mvn clean install：调用 clean 生命周期的 clean 阶段和 default 生命周期的 package 阶段，实际执行了 pre-clean 和 clean 阶段和 default 生命周期 validate 到 install 阶段之间的所有阶段。

mvn clean deploy：调用 clean 生命周期的 clean 阶段和 default 生命周期的 package 阶段，实际执行了 pre-clean 和 clean 阶段和 default 生命周期 validate 到 deploy 阶段之间的所有阶段。

2

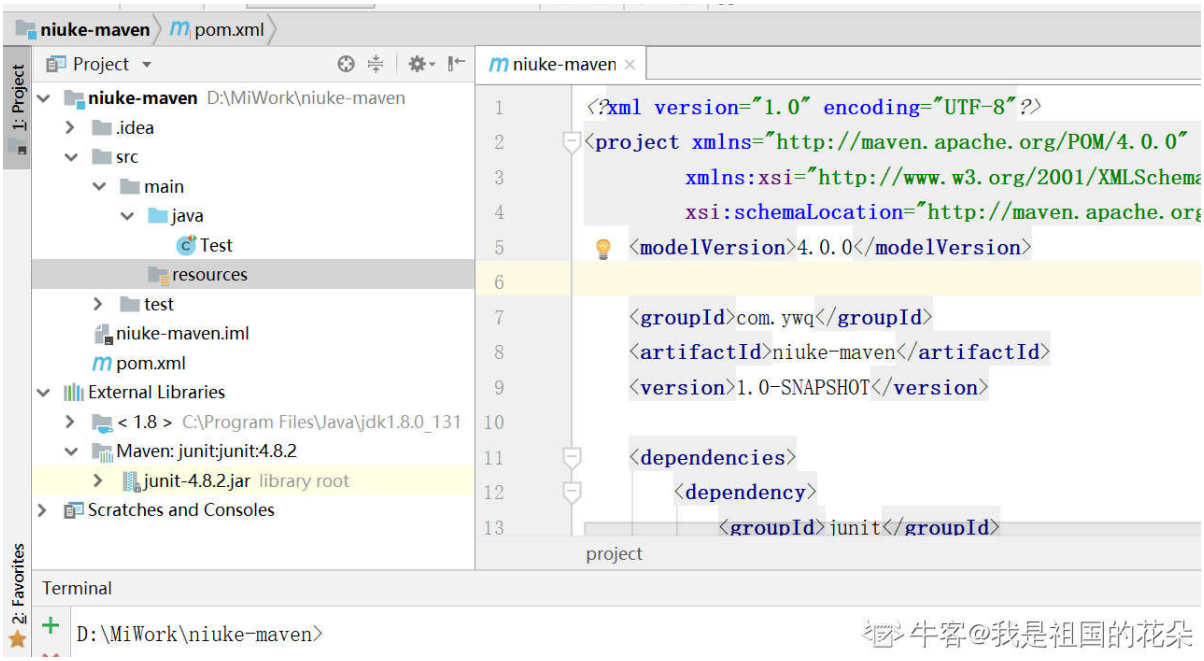
常用构建命令功能：（重点掌握）

在我们日常的 Maven 使用中，一条条简单的命令，mvn clean、mvn package 等都是在执行 Maven 的某个生命周期阶段，**各个常用命令的功能如下**：

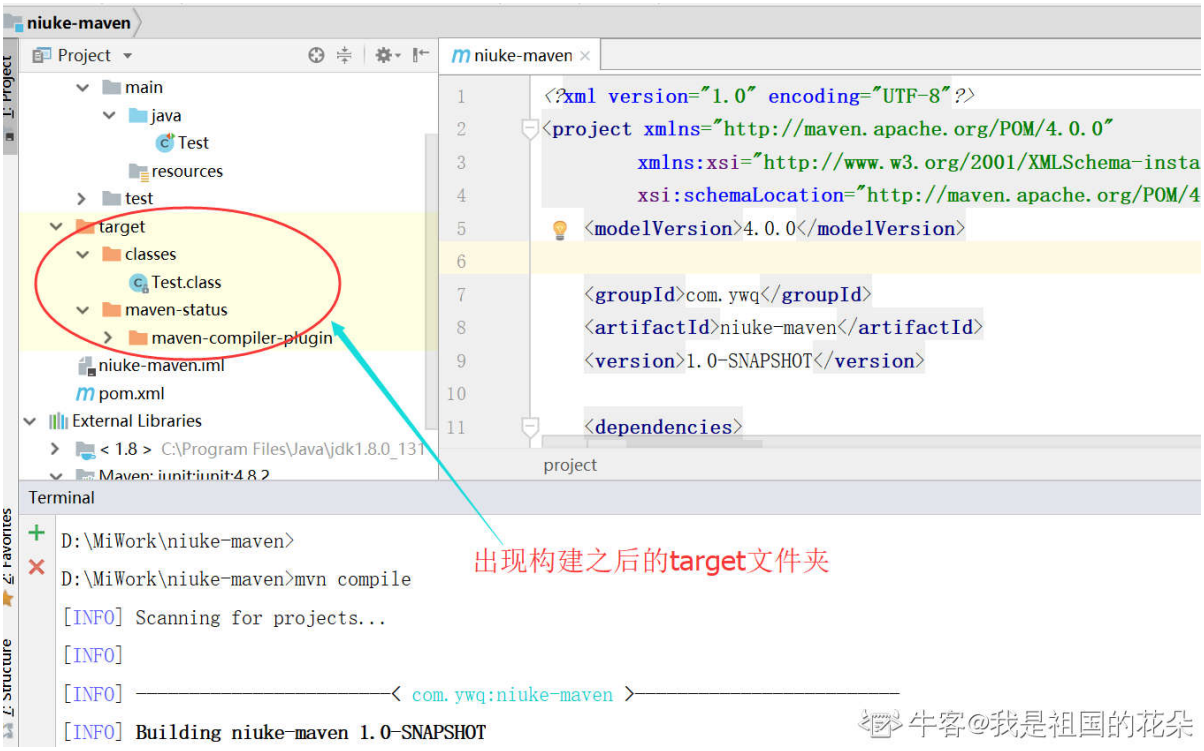
- **mvn clean**：删除上次打包生成的target文件夹。
- **mvn compile**：对源代码进行编译。
- **mvn clean package**：用于清除上次打包生成得target文件夹并且对项目进行打包。
- **mvn clean install**：将项目之前编译打包生成得target文件夹删除并且重新将项目打包，然后安装到本地仓库。
- **mvn clean deploy**：将项目之前编译打包生成得target文件夹删除并且重新将项目打包，然后将包发布到了远程仓库（私（si）服）。

在实际的Maven项目中，让我们来看下常用Maven构建命令的执行效果吧~

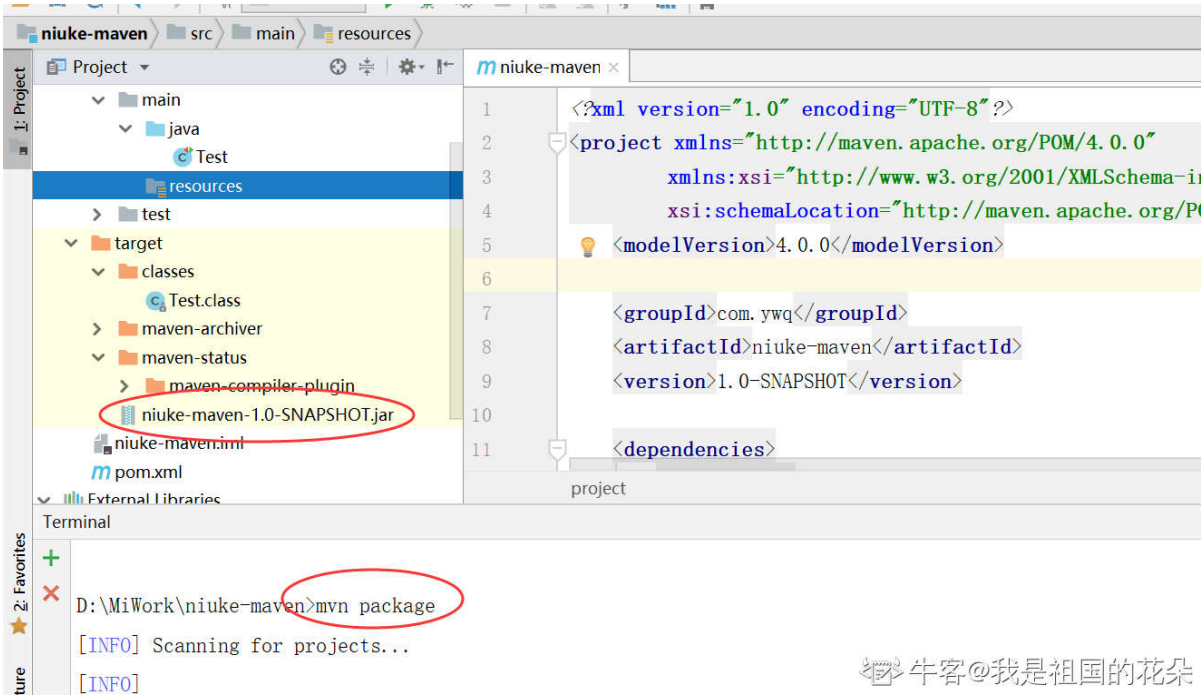
这是我们没有进行项目构建时候的项目结构图：



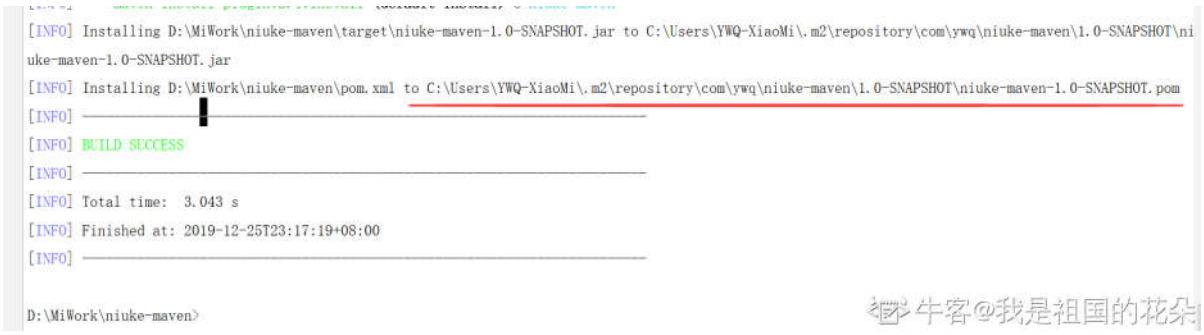
然后，我们执行mvn compile对项目进行编译：

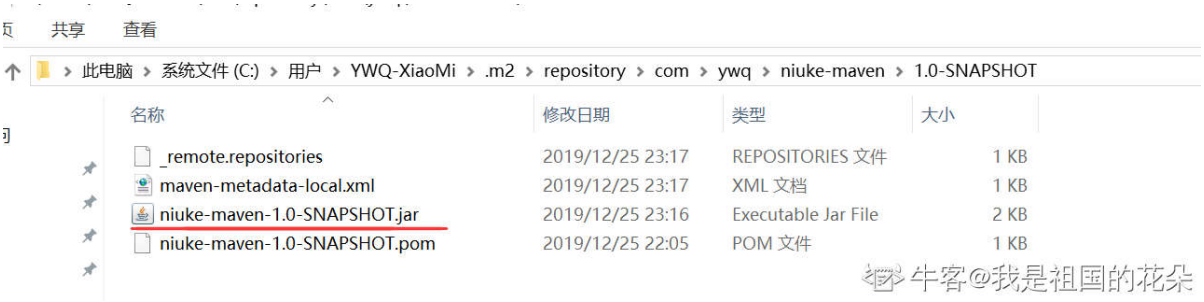


我们再来执行 mvn package，可以看到target下出现了所构建的当前项目的依赖包：



最后，我们执行mvn install可以看到当前项目被安装到了本地仓库中：





2

总结：

在日常开发工作中，Maven是常见的项目构建工具以及依赖管理工具。本小节中，我们对Maven的两大作用，即**下载依赖和构建项目**做了较为详细的介绍与交流。在面试中，一般会考察**Maven的基本概念以及如何使用命令来完成项目的构建**，希望大家可以熟练掌握。下一小节中，我们将介绍版本控制工具Git的相关知识点及其常用命令。

限于作者水平，文章中难免会有不妥之处。大家在学习过程中遇到我没有表达清楚或者表述有误的地方，欢迎随时在文章下边指出，我会及时关注，随时改正。另外，大家有任何话题都可以在下边留言，我们一起交流探讨。

讨论 评论

刘畅201904211606816 1#

打卡

发表于 2020-01-14 11:39:53 赞(0) 回复(0)

牧水s N 2#

打卡

发表于 2020-02-15 19:09:37 赞(0) 回复(0)