

Maven依赖管理项目构建工具

一、Maven简介

1、为什么学习Maven

1.1、Maven是一个依赖管理工具

①jar 包的规模

随着我们使用越来越多的框架，或者框架封装程度越来越高，项目中使用的jar包也越来越多。项目中，一个模块里面用到上百个jar包是非常正常的。

比如下面的例子，我们只用到 SpringBoot、SpringCloud 框架中的三个功能：

- Nacos 服务注册发现
- Web 框架环境
- 视图模板技术 Thymeleaf

最终却导入了 106 个 jar 包：

```
org.springframework.security:spring-security-rsa:jar:1.0.9.RELEASE:compile
com.netflix.ribbon:ribbon:jar:2.3.0:compile
org.springframework.boot:spring-boot-starter-thymeleaf:jar:2.3.6.RELEASE:compile
commons-configuration:commons-configuration:jar:1.8:compile
org.apache.logging.log4j:log4j-api:jar:2.13.3:compile
org.springframework:spring-beans:jar:5.2.11.RELEASE:compile
org.springframework.cloud:spring-cloud-starter-netflix-ribbon:jar:2.2.6.RELEASE:compile
org.apache.tomcat.embed:tomcat-embed-websocket:jar:9.0.39:compile
com.alibaba.cloud:spring-cloud-alibaba-commons:jar:2.2.6.RELEASE:compile
org.bouncycastle:bcprov-jdk15on:jar:1.64:compile
org.springframework.security:spring-security-crypto:jar:5.3.5.RELEASE:compile
org.apache.httpcomponents:httpasyncclient:jar:4.1.4:compile
com.google.j2objc:j2objc-annotations:jar:1.3:compile
com.fasterxml.jackson.core:jackson-databind:jar:2.11.3:compile
io.reactivex:rxjava:jar:1.3.8:compile
ch.qos.logback:logback-classic:jar:1.2.3:compile
org.springframework:spring-web:jar:5.2.11.RELEASE:compile
io.reactivex:rxnetty-servo:jar:0.4.9:runtime
org.springframework:spring-core:jar:5.2.11.RELEASE:compile
io.github.openfeign.form:feign-form-spring:jar:3.8.0:compile
io.github.openfeign.form:feign-form:jar:3.8.0:compile
com.netflix.ribbon:ribbon-loadbalancer:jar:2.3.0:compile
org.apache.httpcomponents:httpcore:jar:4.4.13:compile
org.thymeleaf.extras:thymeleaf-extras-java8time:jar:3.0.4.RELEASE:compile
org.slf4j:jul-to-slf4j:jar:1.7.30:compile
com.atguigu.demo:demo09-base-entity:jar:1.0-SNAPSHOT:compile
org.yaml:snakeyaml:jar:1.26:compile
org.springframework.boot:spring-boot-starter-logging:jar:2.3.6.RELEASE:compile
```

io.reactivex:rxnetty-contexts:jar:0.4.9:runtime
org.apache.httpcomponents:httpclient:jar:4.5.13:compile
io.github.openfeign:feign-core:jar:10.10.1:compile
org.springframework.boot:spring-boot-starter-aop:jar:2.3.6.RELEASE:compile
org.hdrhistogram:HdrHistogram:jar:2.1.9:compile
org.springframework:spring-context:jar:5.2.11.RELEASE:compile
commons-lang:commons-lang:jar:2.6:compile
io.prometheus:simpleclient:jar:0.5.0:compile
ch.qos.logback:logback-core:jar:1.2.3:compile
org.springframework:spring-webmvc:jar:5.2.11.RELEASE:compile
com.sun.jersey:jersey-core:jar:1.19.1:runtime
javax.ws.rs:jsr311-api:jar:1.1.1:runtime
javax.inject:javax.inject:jar:1:runtime
org.springframework.cloud:spring-cloud-openfeign-core:jar:2.2.6.RELEASE:compile
com.netflix.ribbon:ribbon-core:jar:2.3.0:compile
com.netflix.hystrix:hystrix-core:jar:1.5.18:compile
com.netflix.ribbon:ribbon-transport:jar:2.3.0:runtime
org.springframework.boot:spring-boot-starter-json:jar:2.3.6.RELEASE:compile
org.springframework.cloud:spring-cloud-starter-openfeign:jar:2.2.6.RELEASE:compile
com.fasterxml.jackson.module:jackson-module-parameter-names:jar:2.11.3:compile
com.sun.jersey.contribs:jersey-apache-client4:jar:1.19.1:runtime
io.github.openfeign:feign-hystrix:jar:10.10.1:compile
io.github.openfeign:feign-slf4j:jar:10.10.1:compile
com.alibaba.nacos:nacos-client:jar:1.4.2:compile
org.apache.httpcomponents:httpcore-nio:jar:4.4.13:compile
com.sun.jersey:jersey-client:jar:1.19.1:runtime
org.springframework.cloud:spring-cloud-context:jar:2.2.6.RELEASE:compile
org.glassfish:jakarta.el:jar:3.0.3:compile
org.apache.logging.log4j:log4j-to-slf4j:jar:2.13.3:compile
com.fasterxml.jackson.datatype:jackson-datatype-jsr310:jar:2.11.3:compile
org.springframework.cloud:spring-cloud-commons:jar:2.2.6.RELEASE:compile
org.aspectj:aspectjweaver:jar:1.9.6:compile
com.alibaba.cloud:spring-cloud-starter-alibaba-nacos-discovery:jar:2.2.6.RELEASE:compile
com.google.guava:listenablefuture:jar:9999.0-empty-to-avoid-conflict-with-guava:compile
com.alibaba.spring:spring-context-support:jar:1.0.10:compile
jakarta.annotation:jakarta.annotation-api:jar:1.3.5:compile
org.bouncycastle:bcpkix-jdk15on:jar:1.64:compile
com.netflix.netflix-commons:netflix-commons-util:jar:0.3.0:runtime
com.fasterxml.jackson.core:jackson-annotations:jar:2.11.3:compile
com.google.guava:guava:jar:29.0-jre:compile
com.google.guava:failureaccess:jar:1.0.1:compile
org.springframework.boot:spring-boot:jar:2.3.6.RELEASE:compile
com.fasterxml.jackson.datatype:jackson-datatype-jdk8:jar:2.11.3:compile
com.atguigu.demo:demo08-base-api:jar:1.0-SNAPSHOT:compile
org.springframework.cloud:spring-cloud-starter-netflix-archaius:jar:2.2.6.RELEASE:compile
org.springframework.boot:spring-boot-autoconfigure:jar:2.3.6.RELEASE:compile
org.slf4j:slf4j-api:jar:1.7.30:compile
commons-io:commons-io:jar:2.7:compile
org.springframework.cloud:spring-cloud-starter:jar:2.2.6.RELEASE:compile
org.apache.tomcat.embed:tomcat-embed-core:jar:9.0.39:compile
io.reactivex:rxnetty:jar:0.4.9:runtime
com.fasterxml.jackson.core:jackson-core:jar:2.11.3:compile

```
com.google.code.findbugs:jsr305:jar:3.0.2:compile
com.netflix.archaius:archaius-core:jar:0.7.6:compile
org.springframework.boot:spring-boot-starter-web:jar:2.3.6.RELEASE:compile
commons-codec:commons-codec:jar:1.14:compile
com.netflix.servo:servo-core:jar:0.12.21:runtime
com.google.errorprone:error_prone_annotations:jar:2.3.4:compile
org.attoparser:attoparser:jar:2.0.5.RELEASE:compile
com.atguigu.demo:demo10-base-util:jar:1.0-SNAPSHOT:compile
org.checkerframework:checker-qual:jar:2.11.1:compile
org.thymeleaf:thymeleaf-spring5:jar:3.0.11.RELEASE:compile
commons-fileupload:commons-fileupload:jar:1.4:compile
com.netflix.ribbon:ribbon-httpclient:jar:2.3.0:compile
com.netflix.netflix-commons:netflix-statistics:jar:0.1.1:runtime
org.unbescape:unbescape:jar:1.1.6.RELEASE:compile
org.springframework:spring-jcl:jar:5.2.11.RELEASE:compile
com.alibaba.nacos:nacos-common:jar:1.4.2:compile
commons-collections:commons-collections:jar:3.2.2:runtime
javax.persistence:persistence-api:jar:1.0:compile
com.alibaba.nacos:nacos-api:jar:1.4.2:compile
org.thymeleaf:thymeleaf:jar:3.0.11.RELEASE:compile
org.springframework:spring-aop:jar:5.2.11.RELEASE:compile
org.springframework.boot:spring-boot-starter:jar:2.3.6.RELEASE:compile
org.springframework.boot:spring-boot-starter-tomcat:jar:2.3.6.RELEASE:compile
org.springframework.cloud:spring-cloud-netflix-ribbon:jar:2.2.6.RELEASE:compile
org.springframework:spring-expression:jar:5.2.11.RELEASE:compile
org.springframework.cloud:spring-cloud-netflix-archaius:jar:2.2.6.RELEASE:compile
```

而如果使用 Maven 来引入这些 jar 包只需要配置三个『依赖』：

```
1  <!-- Nacos 服务注册发现启动器 -->
2  <dependency>
3      <groupId>com.alibaba.cloud</groupId>
4      <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
5  </dependency>
6
7  <!-- web启动器依赖 -->
8  <dependency>
9      <groupId>org.springframework.boot</groupId>
10     <artifactId>spring-boot-starter-web</artifactId>
11 </dependency>
12
13 <!-- 视图模板技术 thymeleaf -->
14 <dependency>
15     <groupId>org.springframework.boot</groupId>
16     <artifactId>spring-boot-starter-thymeleaf</artifactId>
17 </dependency>
```

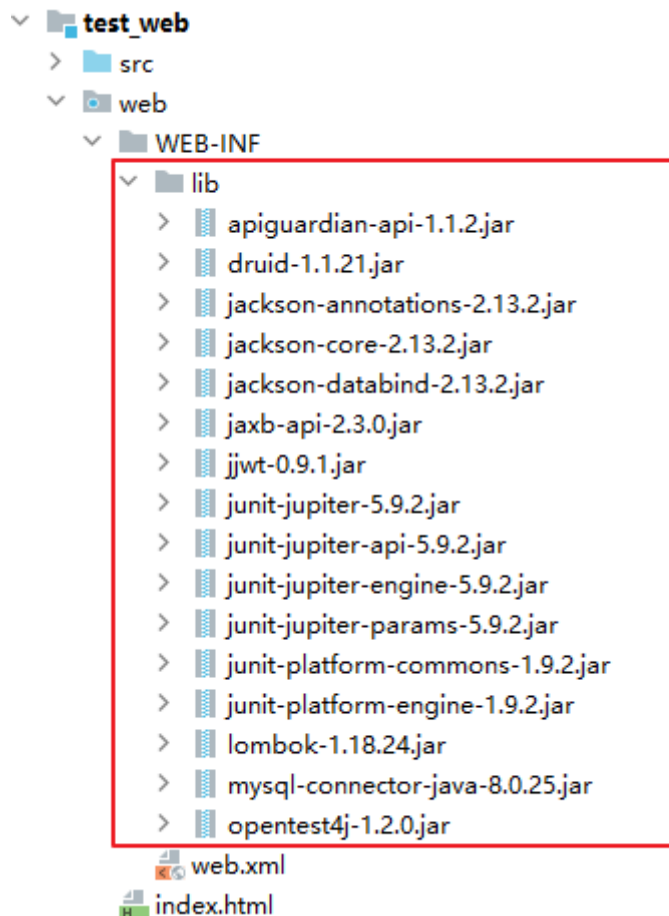
②jar包的来源问题

- 这个jar包所属技术的官网。官网通常是英文界面，网站的结构又不尽相同，甚至找到下载链接还发现需要通过特殊的工具下载。
- 第三方网站提供下载。问题是不规范，在使用过程中会出现各种问题。
 - jar包的名称
 - jar包的版本

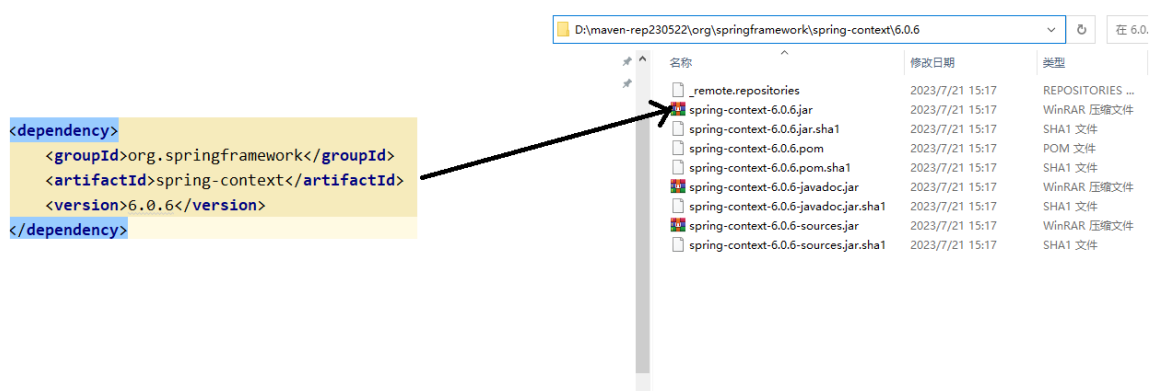
- jar包内的具体细节
- 而使用 Maven 后，依赖对应的 jar 包能够**自动下载**，方便、快捷又规范。

③jar包的导入问题

在web工程中，jar包必须存放在指定位置：



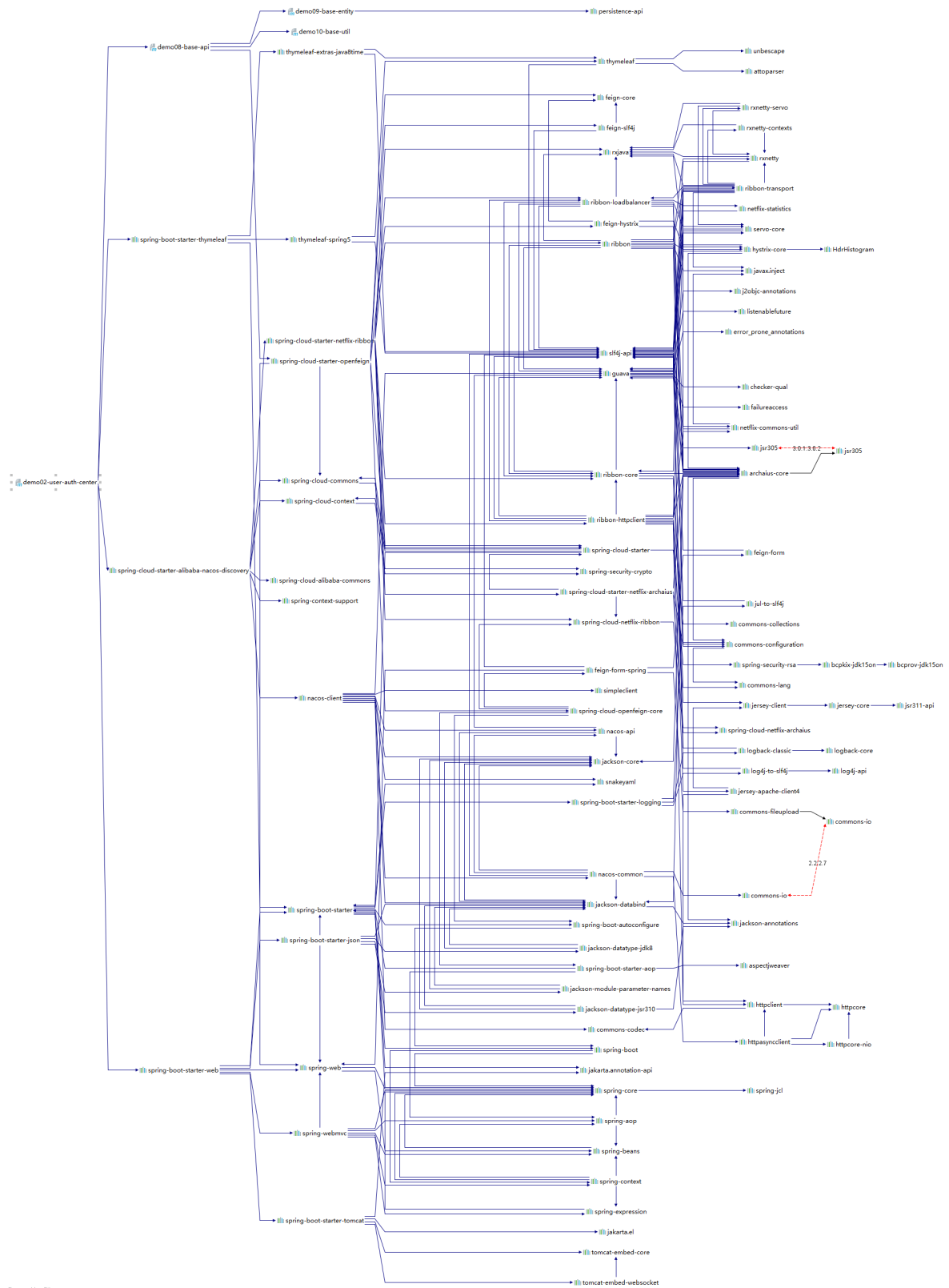
在使用Maven之后，通过配置依赖(jar包)的坐标，查找本地仓库中相应jar包，若本地仓库没有，则统一从镜像网站或中央仓库中下载：



④jar包之间的依赖

框架中使用的 jar 包，不仅数量庞大，而且彼此之间存在错综复杂的依赖关系。依赖关系的复杂程度，已经上升到了完全不能靠人力手动解决的程度。另外，jar 包之间有可能产生冲突。进一步增加了我们在 jar 包使用过程中的难度。

下面是前面例子中 jar 包之间的依赖关系：



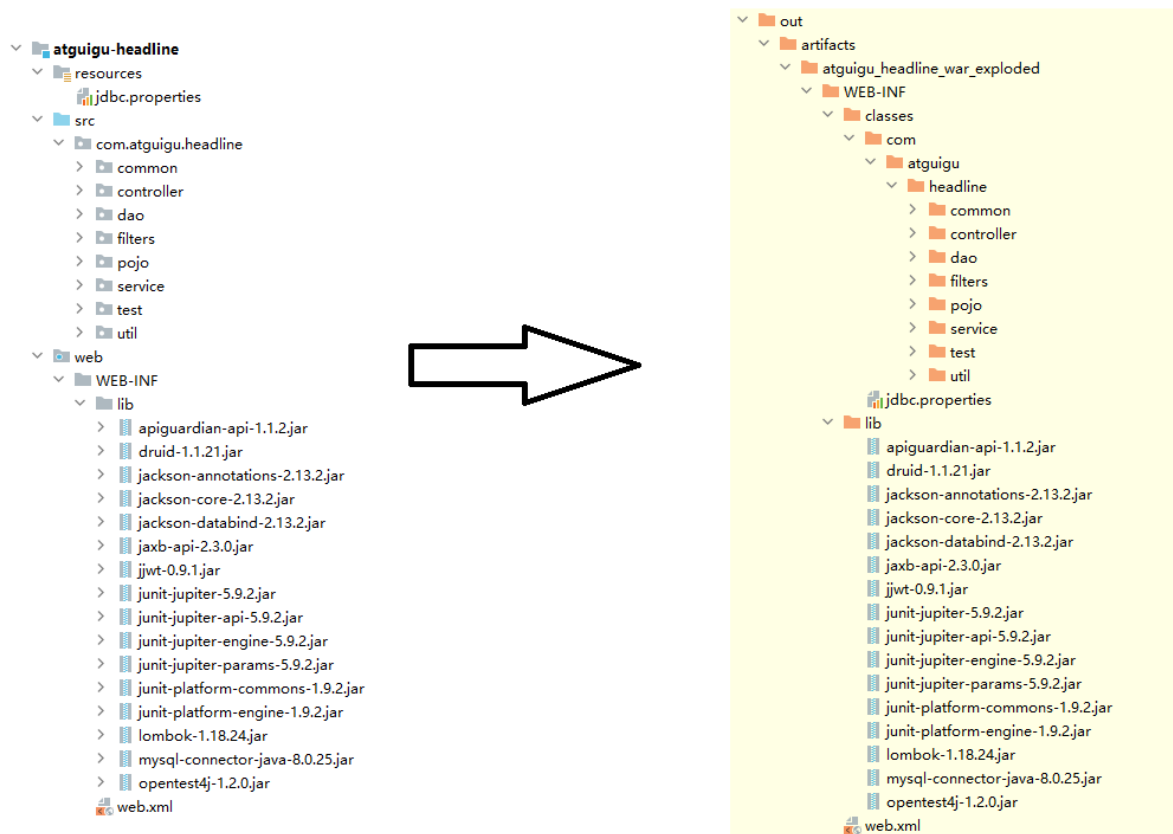
而实际上 jar 包之间的依赖关系是普遍存在的，如果要由程序员手动梳理无疑会增加极高的学习成本，而这些工作又对实现业务功能毫无帮助。

而使用 Maven 则几乎不需要管理这些关系，极个别的地方调整一下即可，极大的减轻了我们的工作量。

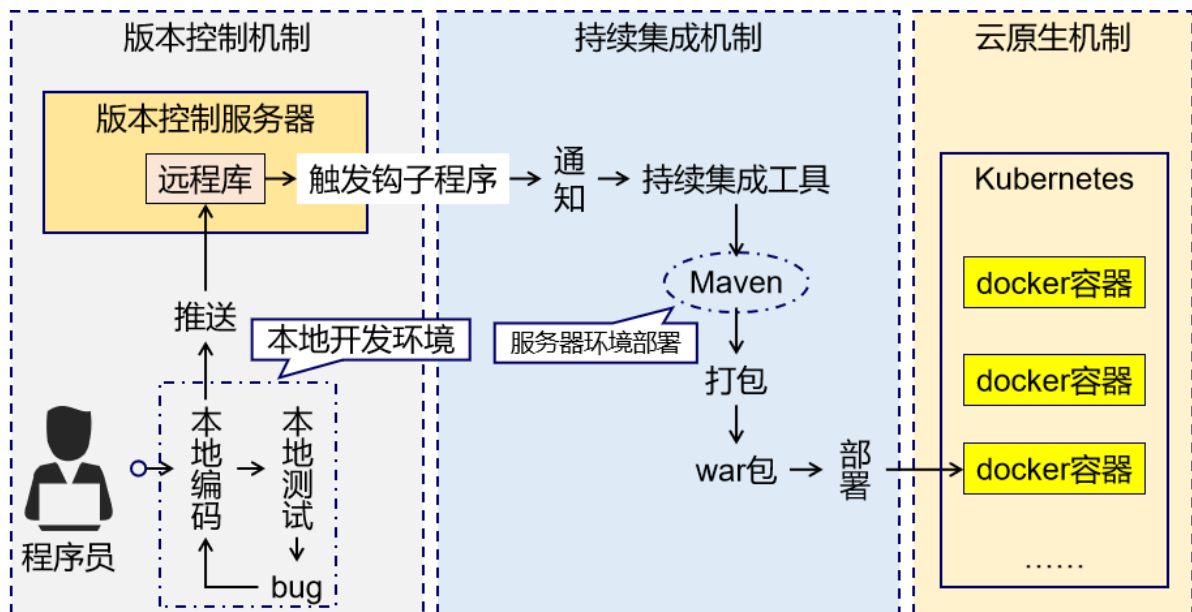
1.2、Maven是一个构建工具

①你没有注意过的构建

你可以不使用 Maven，但是构建必须要做。当我们使用 IDEA 进行开发时，构建是 IDEA 替我们做的。



②脱离 IDE 环境仍需构建



1.3、结论

- 管理规模庞大的 jar 包，需要专门工具。
- 脱离 IDE 环境执行构建操作，需要专门工具。

2. Maven介绍

<https://maven.apache.org/what-is-maven.html>

Maven 是一款为 Java 项目管理构建、依赖管理的工具（软件），使用 Maven 可以自动化构建、测试、打包和发布项目，大大提高了开发效率和质量。

Maven 就是一个软件，掌握安装、配置、以及基本功能（项目构建、依赖管理）的理解和使用即可！

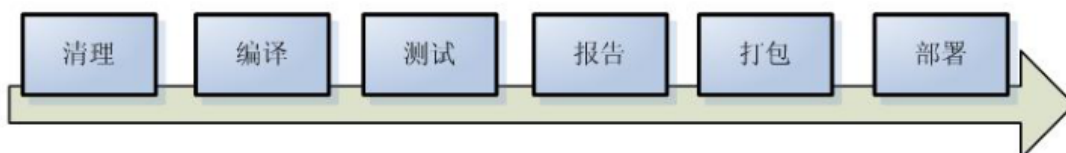
1. 依赖管理：

Maven 可以管理项目的依赖，包括自动下载所需依赖库、自动下载依赖需要的依赖并且保证版本没有冲突、依赖版本管理等。通过 Maven，我们可以方便地维护项目所依赖的外部库，避免版本冲突和转换错误等，而我们仅仅需要编写配置即可。

2. 构建管理：

项目构建是指将源代码、配置文件、资源文件等转化为能够运行或部署的应用程序或库的过程

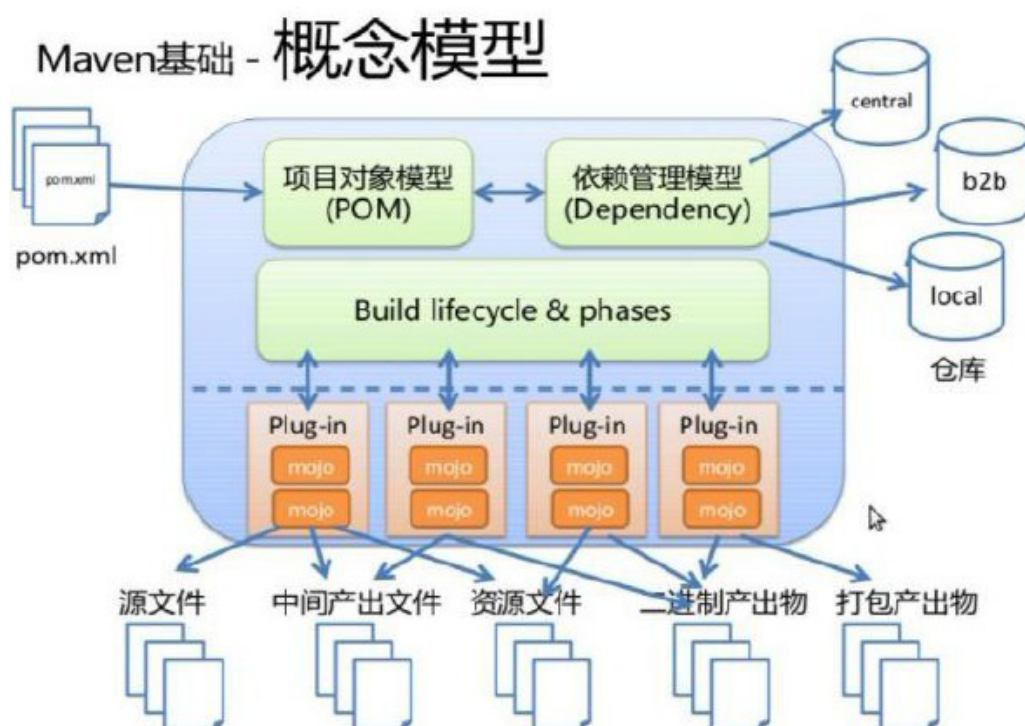
Maven 可以管理项目的编译、测试、打包、部署等构建过程。通过实现标准的构建生命周期，Maven 可以确保每一个构建过程都遵循同样的规则和最佳实践。同时，Maven 的插件机制也使得开发者可以对构建过程进行扩展和定制。主动触发构建，只需要简单的命令操作即可。



场景1： 例如我们项目需要第三方依赖如：Druid连接池、MySQL数据库驱动和Jackson JSON等处理。那么我们可以将想要的依赖项的信息编写到Maven工程的配置文件，Maven就会自动下载并复制这些依赖项到项目中，无需自己导入jar包，管理jar!

场景2： 项目完成开发，我们想要打成war部署到服务器中，使用maven的构建命令可以快速打包！节省大量时间！

3. Maven软件工作原理模型图（了解）



二、Maven安装和配置

1. Maven安装

<https://maven.apache.org/docs/history.html>








各个工具选用版本：

工具	版本
Maven	3.8.8
JDK	17
IDEA	2022.2

安装条件： maven需要本机安装java环境、必需包含java_home环境变量！

软件安装： 右键解压即可（绿色免安装）

软件结构：

 bin	2023/3/8 13:58	文件夹	
 boot	2023/3/8 13:58	文件夹	
 conf	2023/3/8 13:58	文件夹	
 lib	2023/3/8 13:58	文件夹	
 LICENSE	2023/3/8 13:58	文件	17 K
 NOTICE	2023/3/8 13:58	文件	6 K
 README.txt	2023/3/8 13:58	TXT 文件	3 K

bin： 含有Maven的运行脚本

boot： 含有plexus-classworlds类加载器框架

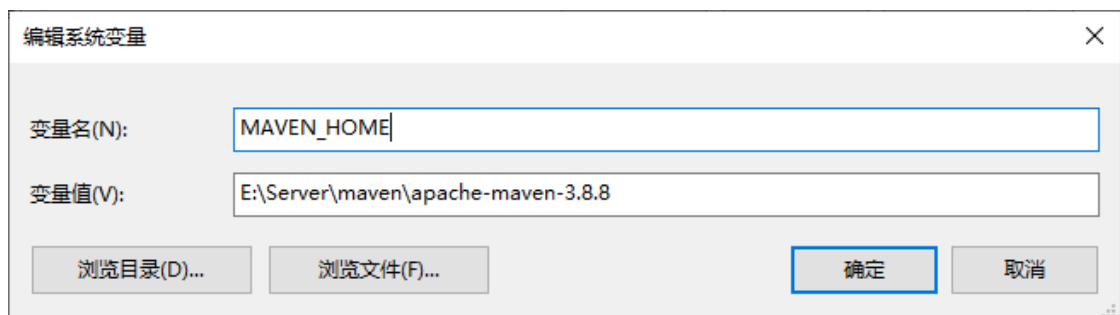
conf： 含有Maven的核心配置文件

lib： 含有Maven运行时所需要的Java类库

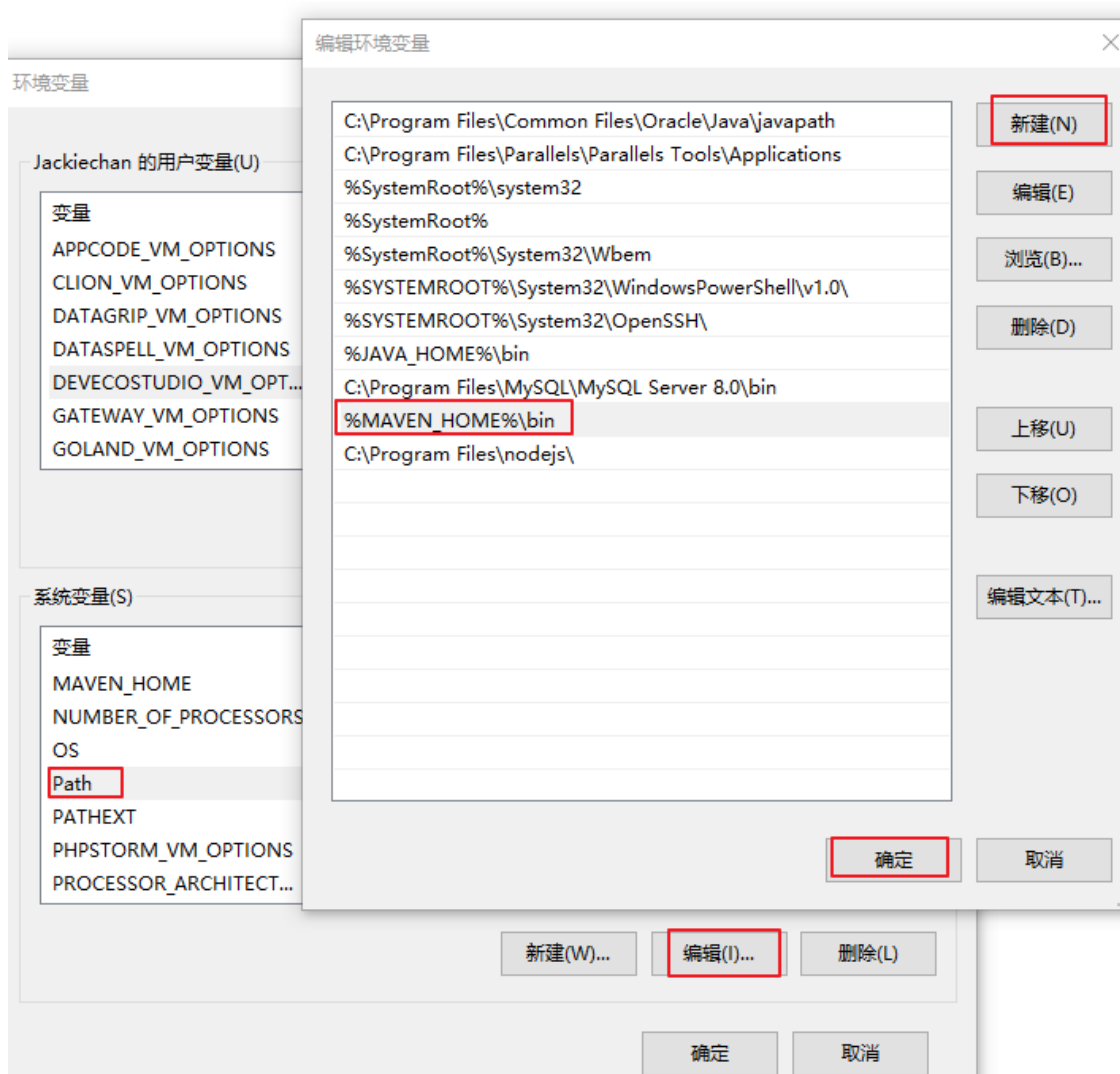
LICENSE、NOTICE、README.txt： 针对Maven版本，第三方软件等简要介绍

2. Maven环境配置

1. 配置MAVEN_HOME



2. 配置Path



3. 命令测试 (cmd窗口)

```
1 mvn -v
2 # 输出版本信息即可，如果错误，请仔细检查环境变量即可！
```

3. Maven功能配置

我们需要需改 **maven/conf/settings.xml** 配置文件，来修改maven的一些默认配置。我们主要休要修改的有三个配置：

- 1.依赖本地缓存位置（本地仓库位置）
- 2.maven下载镜像
- 3.maven选用编译项目的jdk版本

1. 配置本地仓库地址

```
1 <!-- localRepository
2 | The path to the local repository maven will use to store artifacts.
3 |
4 | Default: ${user.home}/.m2/repository
5 <localRepository>/path/to/local/repo</localRepository>
6 -->
7 <!-- conf/settings.xml 55行 -->
8 <localRepository>D:\maven-repository</localRepository>
```

2. 配置国内阿里镜像

```
1  <!--在mirrors节点(标签)下添加中央仓库镜像 160行附近-->
2  <mirror>
3      <id>alimaven</id>
4      <name>aliyun maven</name>
5      <url>http://maven.aliyun.com/nexus/content/groups/public/</url>
6      <mirrorOf>central</mirrorOf>
7  </mirror>
```

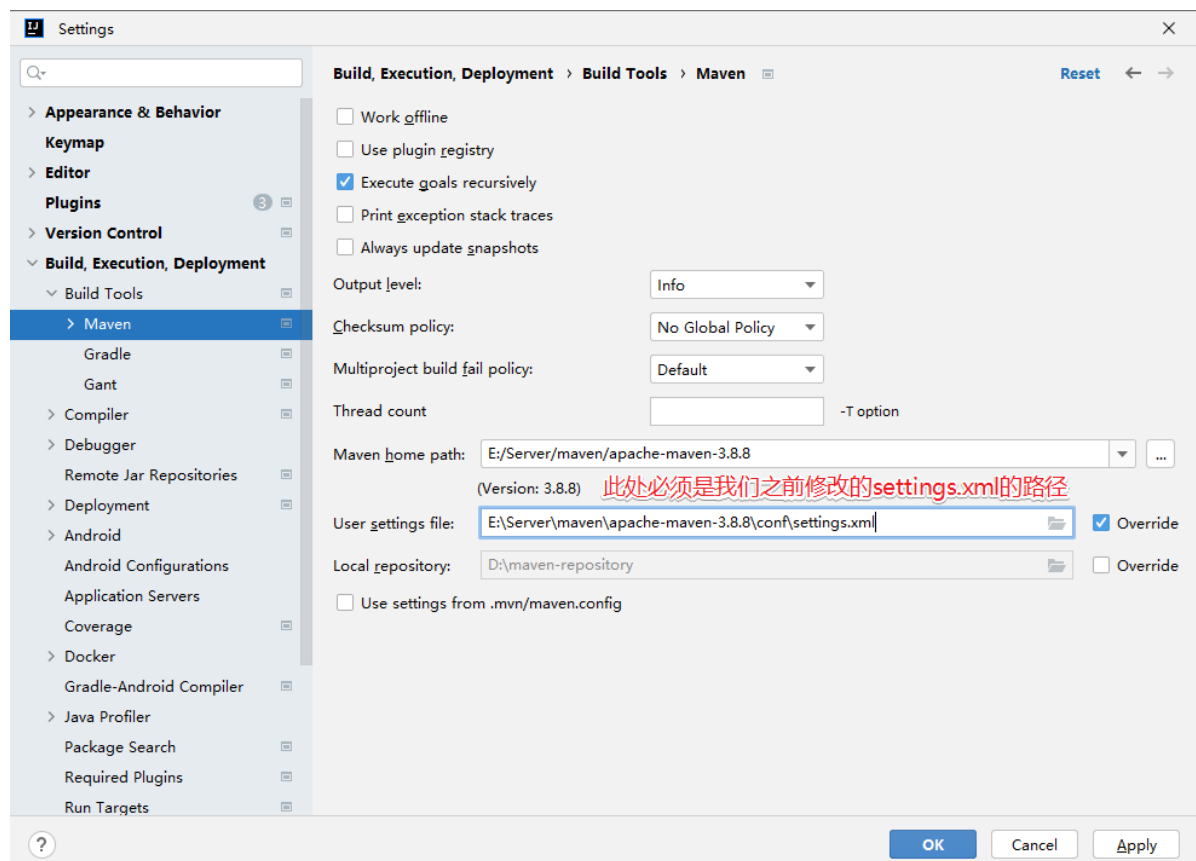
3. 配置jdk17版本项目构建

```
1  <!--在profiles节点(标签)下添加jdk编译版本 268行附近-->
2  <profile>
3      <id>jdk-17</id>
4      <activation>
5          <activeByDefault>true</activeByDefault>
6          <jdk>17</jdk>
7      </activation>
8      <properties>
9          <maven.compiler.source>17</maven.compiler.source>
10         <maven.compiler.target>17</maven.compiler.target>
11         <maven.compiler.compilerVersion>17</maven.compiler.compilerVersion>
12     </properties>
13 </profile>
```

4. IDEA配置本地Maven软件

我们需要将配置好的maven软件，配置到idea开发工具中即可！ 注意：idea工具默认自带maven配置软件，但是因为没有修改配置，建议替换成本地配置好的maven！

选择本地maven软件



注意：

- 1、如果本地仓库地址不变化，只有一个原因，就是maven/conf/settings.xml配置文件编写错误！仔细检查即可！
- 2、一定保证User settings file对应之前修改的settings.xml的路径，若不一致，选中Override复选框，手动选择配置文件

三、基于IDEA创建Maven工程

1. 概念梳理Maven工程的GAVP

Maven工程相对之前的项目，多出一组gavp属性，gav需要我们在创建项目的时候指定，p有默认值，我们先了解下这组属性的含义：

Maven 中的 GAVP 是指 GroupId、ArtifactId、Version、Packaging 等四个属性的缩写，其中前三个是必要的，而 Packaging 属性为可选项。这四个属性主要为每个项目在maven仓库中做一个标识，类似人的姓名！有了具体标识，方便后期项目之间相互引用依赖等！

GAV遵循一下规则：

- 1) **GroupID 格式**：com.{公司/BU }.业务线.[子业务线]，最多 4 级。

说明：{公司/BU} 例如：alibaba/taobao/tmall/aliexpress 等 BU 一级；子业务线可选。

正例：com.taobao.tddl 或 com.alibaba.sourcing.multilang

- 2) **ArtifactID 格式**：产品线名-模块名。语义不重复不遗漏，先到仓库中心去查证一下。

正例：tc-client / uic-api / tair-tool / bookstore

- 3) **Version版本号格式推荐**：主版本号.次版本号.修订号

1) 主版本号：当做了不兼容的 API 修改，或者增加了能改变产品方向的新功能。

2) 次版本号：当做了向下兼容的功能性新增（新增类、接口等）。

3) 修订号：修复 bug，没有修改方法签名的功能加强，保持 API 兼容性。

例如：初始→1.0.0 修改bug→1.0.1 功能调整→1.1.1等

Packaging定义规则：

指示将项目打包为什么类型的文件，idea根据packaging值，识别maven项目类型！

packaging 属性为 jar（默认值），代表普通的Java工程，打包以后是.jar结尾的文件。

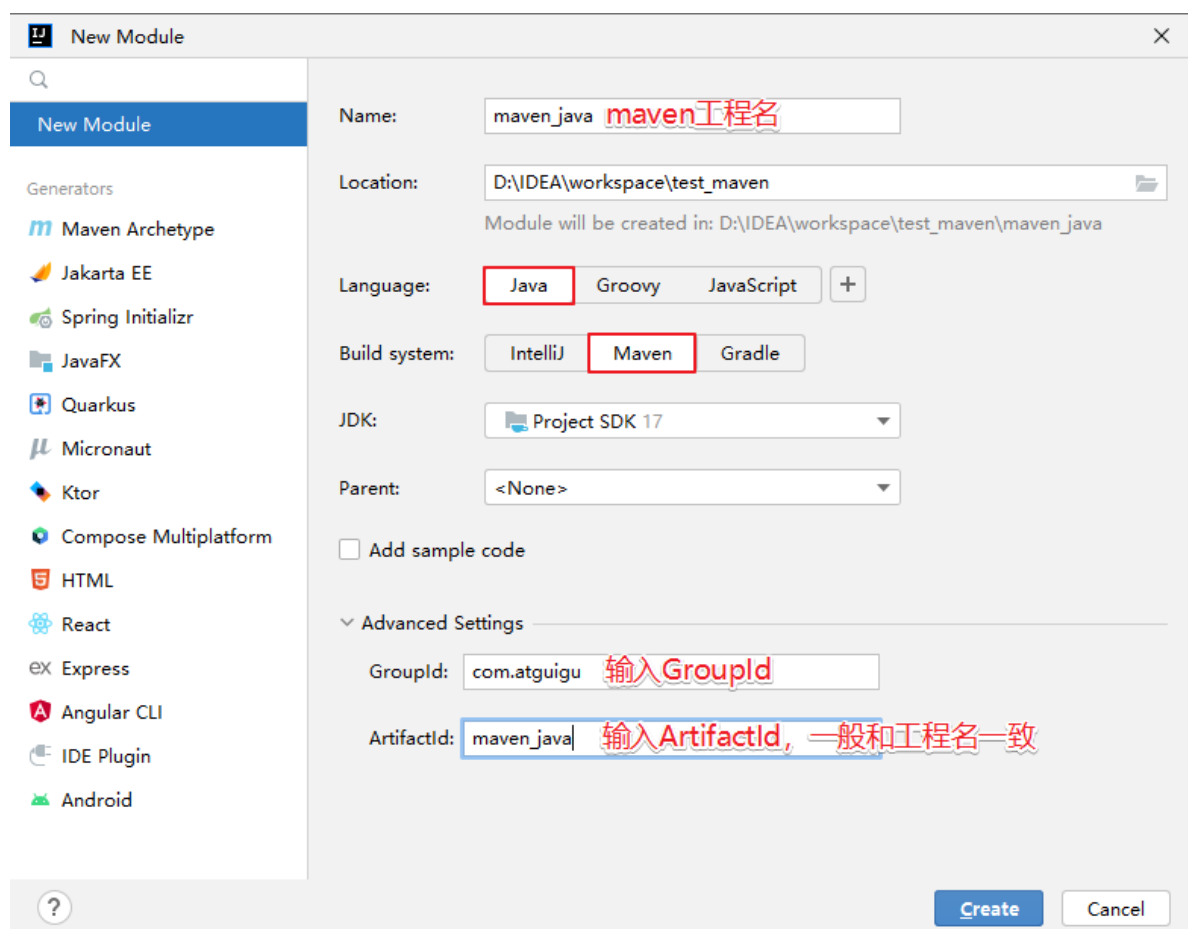
packaging 属性为 war，代表Java的web工程，打包以后.war结尾的文件。

packaging 属性为 pom，代表不会打包，用来做继承的父工程。

2. Idea构建Maven Java SE工程

注意：此处省略了version，直接给了一个默认值：**1.0-SNAPSHOT**

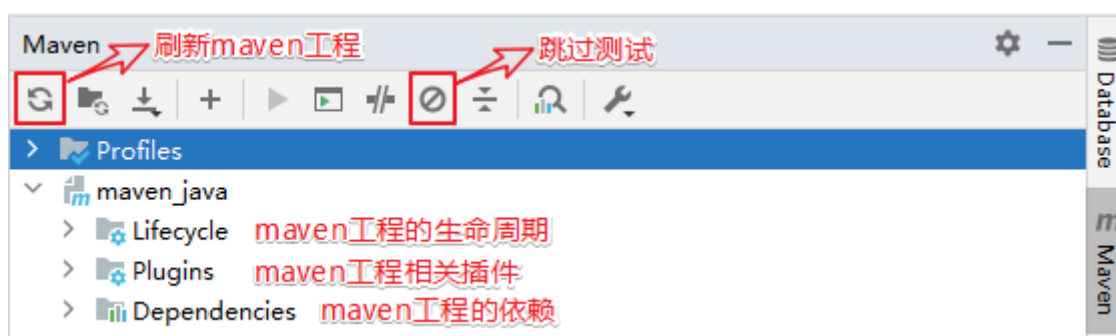
自己后期可以在项目中随意修改！



创建工程之后，若第一次使用maven，或者使用的是新的**本地仓库**，idea右下角会出现以下进度条，表示maven正在下载相关插件，等待下载完毕，进度条消失即可



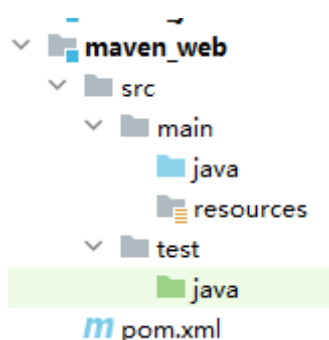
验证maven工程是否创建成功，当创建完毕maven工程之后，idea中会自动打开Maven视图，如下图：



3. Idea构建Maven Java Web工程

1. 手动创建

1. 创建一个maven的javase工程



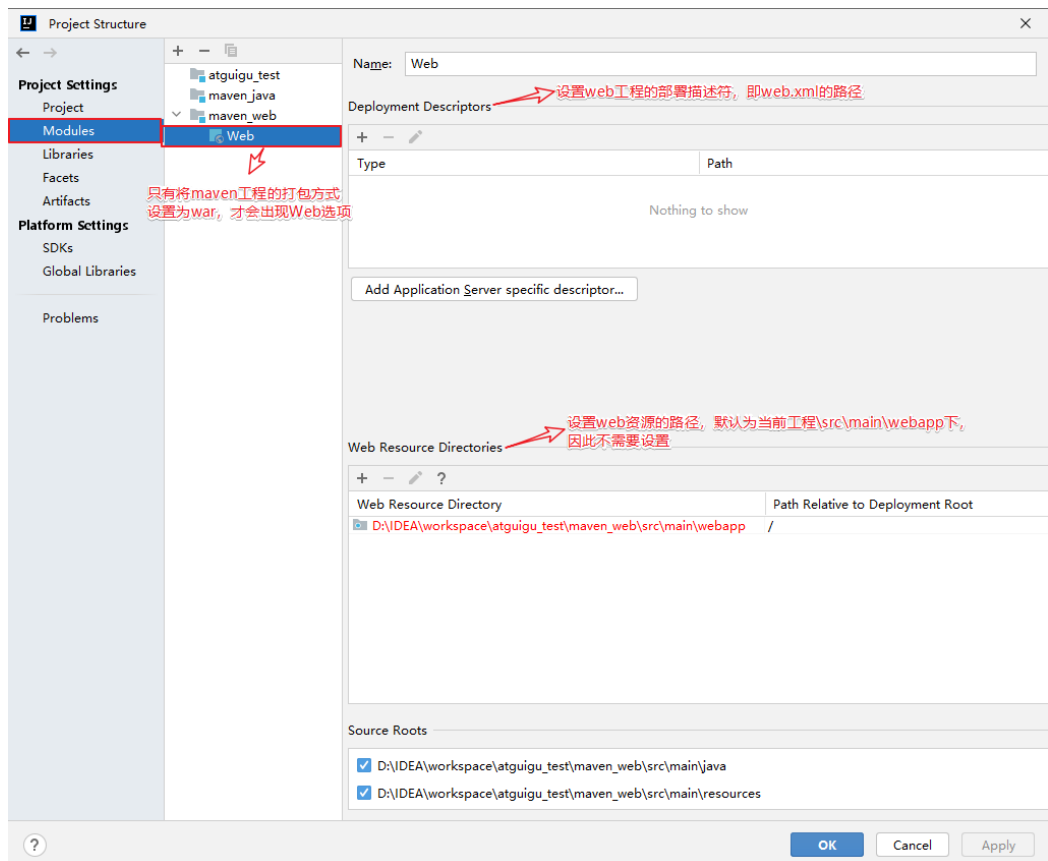
2. 修改pom.xml文件打包方式

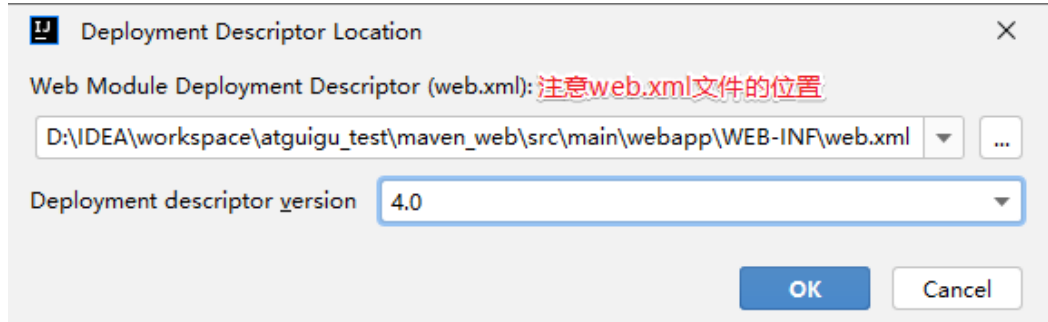
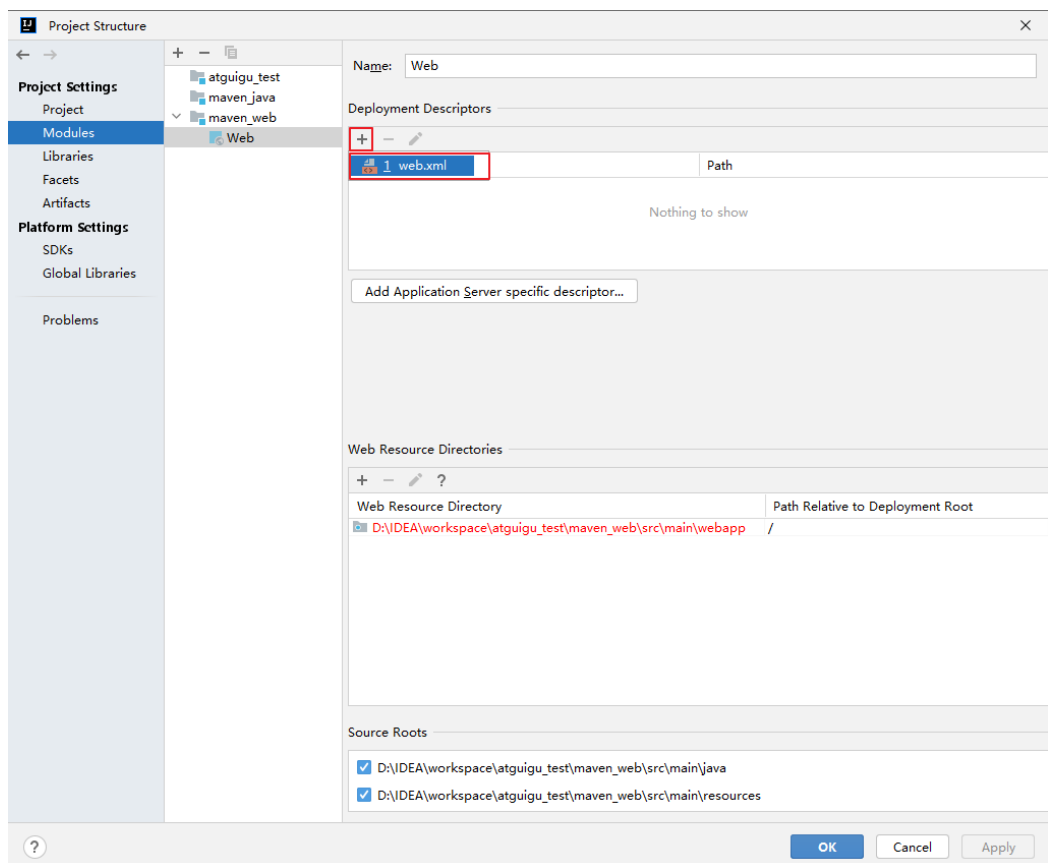
修改位置：项目下/pom.xml

```
1 <groupId>com.atguigu</groupId>
2 <artifactId>maven_web</artifactId>
3 <version>1.0-SNAPSHOT</version>
4 <!-- 新增一行打包方式packaging -->
5 <packaging>war</packaging>
```

3. 设置web资源路径和web.xml路径

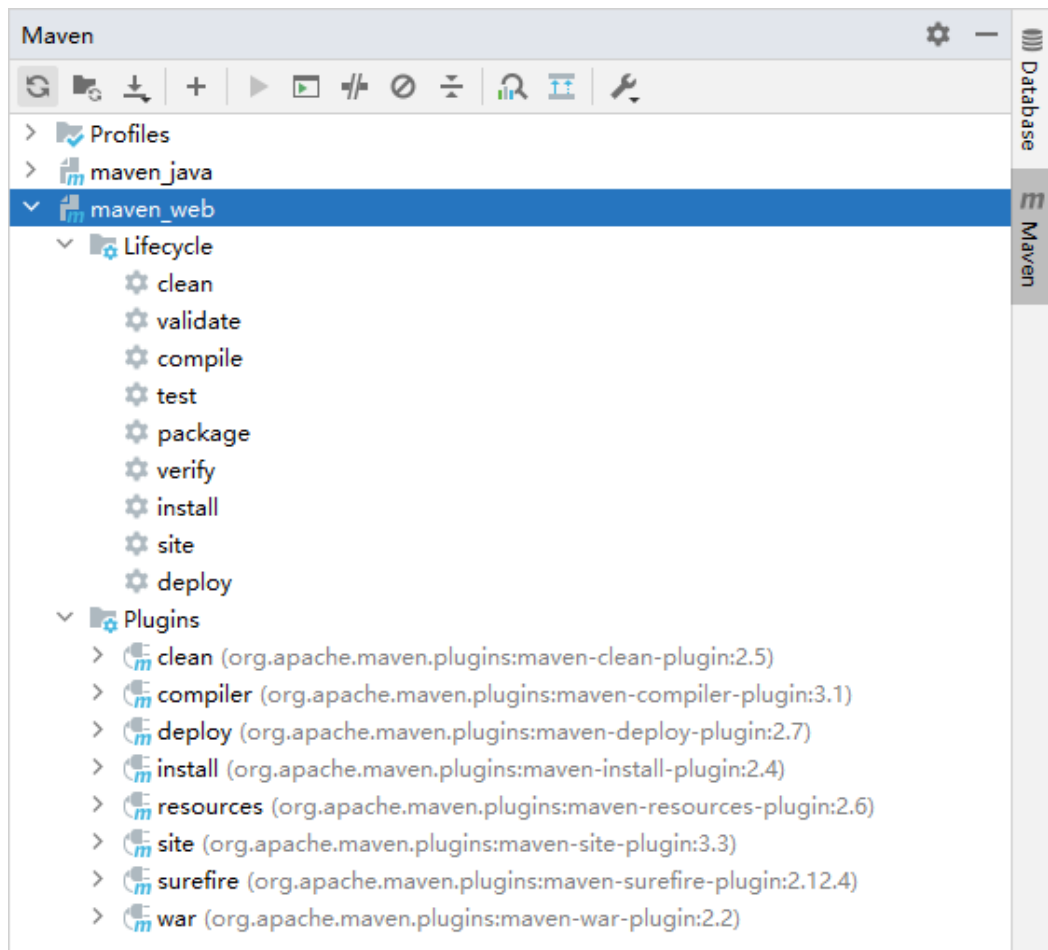
点击File-->Project Structure





4. 刷新和校验

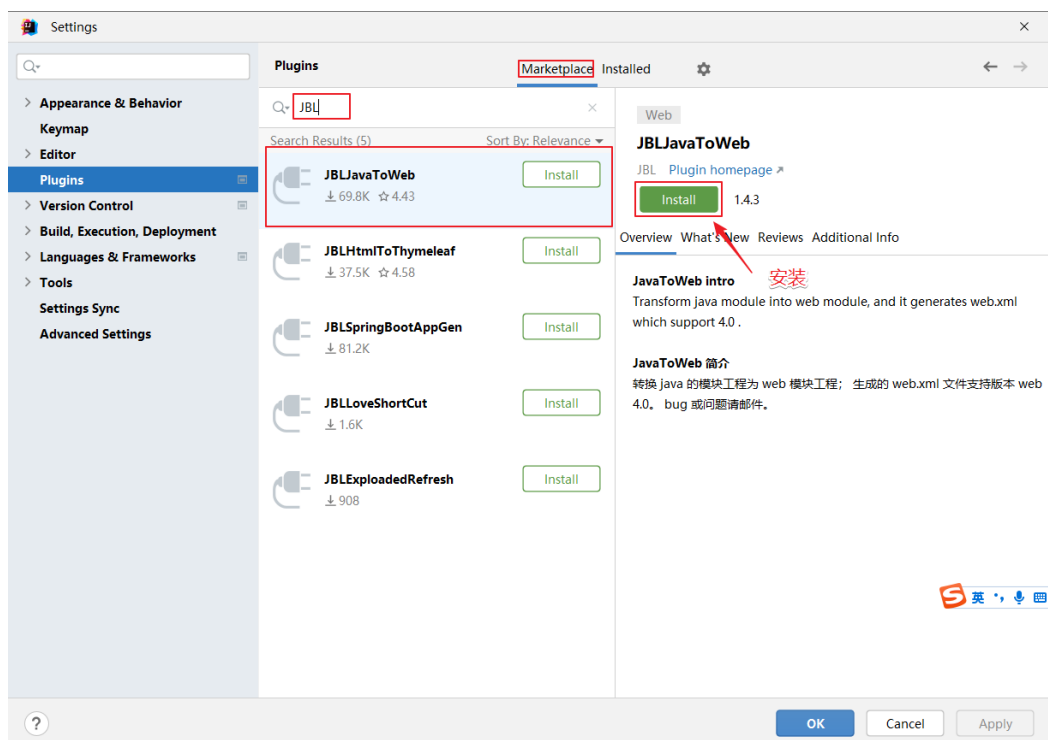




2. 插件创建

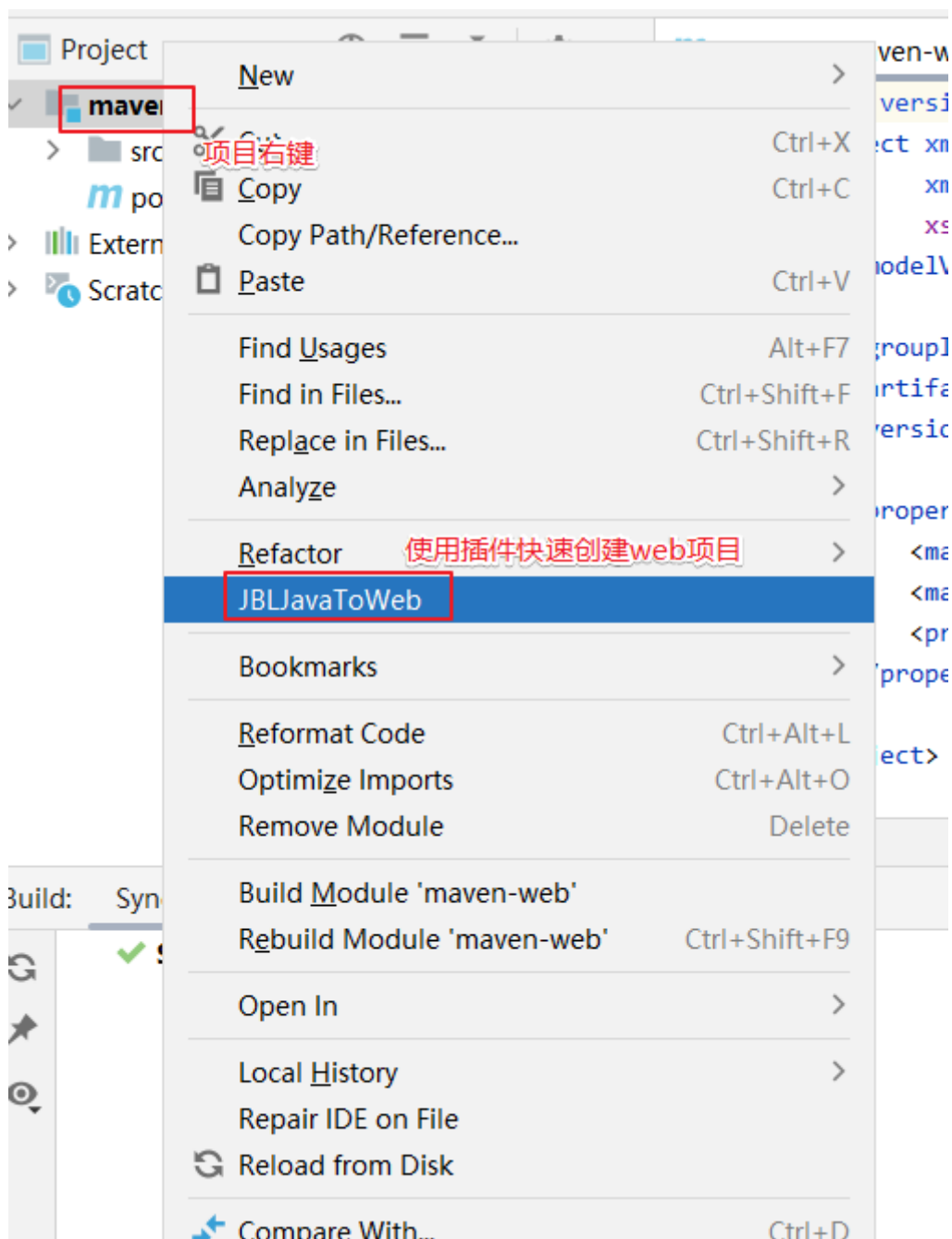
1. 安装插件JBLJavaToWeb

file / settings / plugins / marketplace



2. 创建一个javasemaven工程

3. 右键、使用插件快速补全web项目



4. Maven工程项目结构说明

Maven 是一个强大的构建工具，它提供一种标准化的项目结构，可以帮助开发者更容易地管理项目的依赖、构建、测试和发布等任务。以下是 Maven Web 程序的文件结构及每个文件的作用：

1	-- pom.xml	# Maven 项目管理文件
2	-- src	
3	-- main	# 项目主要代码
4	-- java	# Java 源代码目录
5	-- com/example/myapp	# 开发者代码主目录
6	-- controller	# 存放 Controller 层代码的目录
7	-- service	# 存放 Service 层代码的目录
8	-- dao	# 存放 DAO 层代码的目录
9	-- model	# 存放数据模型的目录
10	-- resources	# 资源目录，存放配置文件、静态资源等
11	-- log4j.properties	# 日志配置文件
12	-- spring-mybatis.xml	# Spring Mybatis 配置文件
13	-- static	# 存放静态资源的目录
14	-- css	# 存放 CSS 文件的目录

15				-- js	# 存放 JavaScript 文件的目录
16				-- images	# 存放图片资源的目录
17				-- webapp	# 存放 WEB 相关配置和资源
18				-- WEB-INF	# 存放 WEB 应用配置文件
19				-- web.xml	# Web 应用的部署描述文件
20				-- classes	# 存放编译后的 class 文件
21				-- index.html	# Web 应用入口页面
22				-- test	# 项目测试代码
23				-- java	# 单元测试目录
24				-- resources	# 测试资源目录

- pom.xml：Maven 项目管理文件，用于描述项目的依赖和构建配置等信息。
- src/main/java：存放项目的 Java 源代码。
- src/main/resources：存放项目的资源文件，如配置文件、静态资源等。
- src/main/webapp/WEB-INF：存放 Web 应用的配置文件。
- src/main/webapp/index.jsp：Web 应用的入口页面。
- src/test/java：存放项目的测试代码。
- src/test/resources：存放测试相关的资源文件，如测试配置文件等。

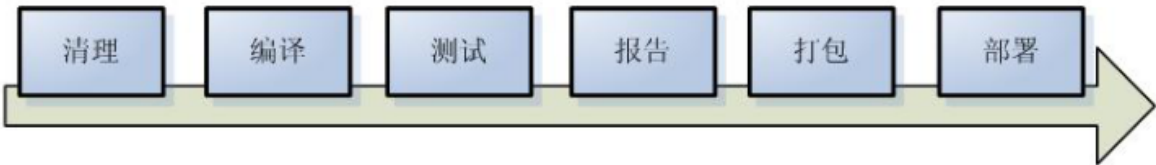
四、基于IDEA进行Maven工程构建

1. 构建概念和构建过程

项目构建是指将源代码、依赖库和资源文件等转换成可执行或可部署的应用程序的过程，在这个过程中包括编译源代码、链接依赖库、打包和部署等多个步骤。

项目构建是软件开发过程中至关重要的一部分，它能够大大提高软件开发效率，使得开发人员能够更加专注于应用程序的开发和维护，而不必关心应用程序的构建细节。

同时，项目构建还能够将多个开发人员的代码汇合到一起，并能够自动化项目的构建和部署，大大降低了项目的出错风险和提高开发效率。常见的构建工具包括 Maven、Gradle、Ant 等。



2. 命令方式项目构建

命令	描述
mvn compile	编译项目，生成target文件
mvn package	打包项目，生成jar或war文件
mvn clean	清理编译或打包后的项目结构
mvn install	打包后上传到maven本地仓库
mvn deploy	只打包，上传到maven私服仓库
mvn site	生成站点
mvn test	执行测试源码

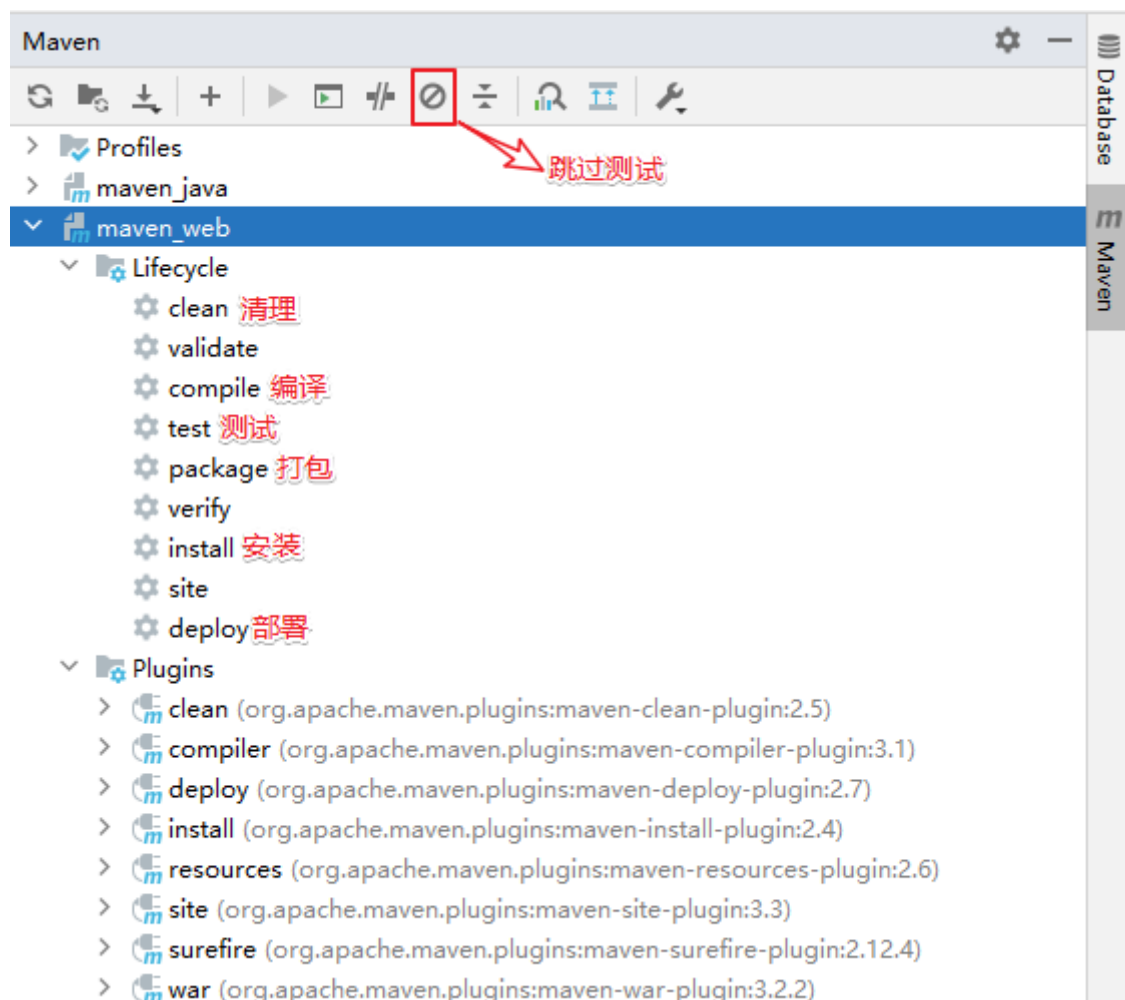
war包打包插件和jdk版本不匹配：pom.xml 添加以下代码即可

```
1 <build>
2     <!-- jdk17 和 war包版本插件不匹配 -->
3     <plugins>
4         <plugin>
5             <groupId>org.apache.maven.plugins</groupId>
6             <artifactId>maven-war-plugin</artifactId>
7             <version>3.2.2</version>
8         </plugin>
9     </plugins>
10 </build>
```

命令触发练习：

```
1 mvn 命令 命令
2
3 #清理
4 mvn clean
5 #清理，并重新打包
6 mvn clean package
7 #执行测试代码
8 mvn test
```

3. 可视化方式项目构建



注意：打包（package）和安装（install）的区别是什么

打包是将工程打成jar或war文件，保存在target目录下

安装是将当前工程所生成的jar或war文件，安装到本地仓库，会按照坐标保存到指定位置

4. 构建插件、命令、生命周期命令之间关系

• 构建生命周期

我们发现一个情况！当我们执行package命令也会自动执行compile命令！

```
1  [INFO] -----[ jar ]-----
2  [INFO]
3  [INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ mybatis-base-curd ---
4  [INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ mybatis-base-curd ---
5  [INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ mybatis-base-curd ---
6  [INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ mybatis-base-curd ---
7  [INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ mybatis-base-curd ---
8  [INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ mybatis-base-curd ---
9  [INFO] Building jar: D:\javaprojects\backend-engineering\part03-mybatis\mybatis-base-curd\target\mybatis-base-curd-1.0-SNAPSHOT.jar
10 [INFO] -----
11 [INFO] BUILD SUCCESS
12 [INFO] -----
13 [INFO] Total time: 5.013 s
14 [INFO] Finished at: 2023-06-05T10:03:47+08:00
15 [INFO] -----
```

这种行为就是因为构建生命周期产生的！构建生命周期可以理解成是一组固定构建命令的有序集合，触发周期后的命令，会自动触发周期前的命令！！

构建周期作用：会简化构建过程

例如：项目打包 mvn clean package即可。

主要两个构建生命周期：

- 清理周期：主要是对项目编译生成文件进行清理

包含命令：clean

- 默认周期：定义了真正构件时所需要执行的所有步骤，它是生命周期中最核心的部分

```
1  包含命令：compile - test - package - install - deploy
```

• 插件、命令、周期三者关系（了解）

周期→包含若干命令→包含若干插件

使用周期命令构建，简化构建过程！

最终进行构建的是插件！

五、基于IDEA 进行Maven依赖管理

1. 依赖管理概念

Maven 依赖管理是 Maven 软件中最重要的功能之一。Maven 的依赖管理能够帮助开发人员自动解决软件包依赖问题，使得开发人员能够轻松地将其他开发人员开发的模块或第三方框架集成到自己的应用程序或模块中，避免出现版本冲突和依赖缺失等问题。

我们通过定义 POM 文件，Maven 能够自动解析项目的依赖关系，并通过 Maven **仓库自动**下载和管理依赖，从而避免了手动下载和管理依赖的繁琐工作和可能引发的版本冲突问题。

总之，Maven 的依赖管理是 Maven 软件的一个核心功能之一，使得软件包依赖的管理和使用更加智能和方便，简化了开发过程中的工作，并提高了软件质量和可维护性。

2. Maven工程核心信息配置和解读（GAVP）

位置：pom.xml

```
1  <!-- 模型版本 -->
2  <modelVersion>4.0.0</modelVersion>
3  <!-- 公司或者组织的唯一标志，并且配置时生成的路径也是由此生成，如com.companyname.project-group，
   maven会将该项目打成的jar包放本地路径：/com/companyname/project-group -->
4  <groupId>com.companyname.project-group</groupId>
5  <!-- 项目的唯一ID，一个groupId下面可能多个项目，就是靠artifactId来区分的 -->
6  <artifactId>project</artifactId>
7  <!-- 版本号 -->
8  <version>1.0.0</version>
9
10 <!--打包方式
11     默认：jar
12     jar指的是普通的java项目打包方式！项目打成jar包！
13     war指的是web项目打包方式！项目打成war包！
14     pom不会讲项目打包！这个项目作为父工程，被其他工程聚合或者继承！后面会讲解两个概念
15 -->
16 <packaging>jar/pom/war</packaging>
```

3. Maven工程依赖管理配置

位置：pom.xml

依赖管理和依赖添加

```
1  <!--
2      通过编写依赖jar包的gav必要属性，引入第三方依赖！
3      scope属性是可选的，可以指定依赖生效范围！
4      依赖信息查询方式：
5          1. maven仓库信息官网 https://mvnrepository.com/
6          2. mavensearch插件搜索
7  -->
8  <dependencies>
9      <!-- 引入具体的依赖包 -->
10     <dependency>
11         <groupId>log4j</groupId>
12         <artifactId>log4j</artifactId>
13         <version>1.2.17</version>
14         <!-- 依赖范围 -->
15         <scope>runtime</scope>
16     </dependency>
17
18 </dependencies>
```

依赖版本统一提取和维护

```
1  <!--声明版本-->
2  <properties>
3      <!--命名随便,内部制定版本号即可！-->
```

```
4      <junit.version>4.12</junit.version>
5      <!-- 也可以通过 maven规定的固定的key，配置maven的参数！如下配置编码格式！ -->
6      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
7      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
8  </properties>
9
10 <dependencies>
11     <dependency>
12         <groupId>junit</groupId>
13         <artifactId>junit</artifactId>
14         <!--引用properties声明版本 -->
15         <version>${junit.version}</version>
16     </dependency>
17 </dependencies>
```

4. 依赖范围

通过设置坐标的依赖范围(scope)，可以设置 对应jar包的作用范围：编译环境、测试环境、运行环境

依赖范围	描述
compile	编译依赖范围，scope 元素的缺省值。使用此依赖范围的 Maven 依赖，对于三种 classpath 均有效，即该 Maven 依赖在上述三种 classpath 均会被引入。例如，log4j 在编译、测试、运行过程都是必须的。
test	测试依赖范围。使用此依赖范围的 Maven 依赖，只对测试 classpath 有效。例如，Junit 依赖只有在测试阶段才需要。
provided	已提供依赖范围。使用此依赖范围的 Maven 依赖，只对编译 classpath 和测试 classpath 有效。例如，servlet-api 依赖对于编译、测试阶段而言是需要的，但是运行阶段，由于外部容器已经提供，故不需要 Maven 重复引入该依赖。
runtime	运行时依赖范围。使用此依赖范围的 Maven 依赖，只对测试 classpath、运行 classpath 有效。例如，JDBC 驱动实现依赖，其在编译时只需 JDK 提供的 JDBC 接口即可，只有测试、运行阶段才需要实现了 JDBC 接口的驱动。
system	系统依赖范围，其效果与 provided 的依赖范围一致。其用于添加非 Maven 仓库的本地依赖，通过依赖元素 dependency 中的 systemPath 元素指定本地依赖的路径。鉴于使用其会导致项目的可移植性降低，一般不推荐使用。
import	导入依赖范围，该依赖范围只能与 dependencyManagement 元素配合使用，其功能是将目标 pom.xml 文件中 dependencyManagement 的配置导入合并到当前 pom.xml 的 dependencyManagement 中。

5. Maven工程依赖下载失败错误解决（重点）

在使用 Maven 构建项目时，可能会发生依赖项下载错误的情况，主要原因有以下几种：

- 1. 下载依赖时出现网络故障或仓库服务器宕机等原因，导致无法连接至 Maven 仓库，从而无法下载依赖。
- 2. 依赖项的版本号或配置文件中的版本号错误，或者依赖项没有正确定义，导致 Maven 下载的依赖项与实际需要的不一致，从而引发错误。
- 3. 本地 Maven 仓库或缓存被污染或损坏，导致 Maven 无法正确地使用现有的依赖项。

解决方案：

1. 检查网络连接和 Maven 仓库服务器状态。
2. 确保依赖项的版本号与项目对应的版本号匹配，并检查 POM 文件中的依赖项是否正确。
3. 清除本地 Maven 仓库缓存（lastUpdated 文件），因为只要存在lastupdated缓存文件，刷新也不会重新下载。本地仓库中，根据依赖的gav属性依次向下查找文件夹，最终删除内部的文件，刷新重新下载即可！

例如：pom.xml依赖

```
1 <dependency>
2   <groupId>com.alibaba</groupId>
3   <artifactId>druid</artifactId>
4   <version>1.2.8</version>
5 </dependency>
```

文件：

电脑 > 本地磁盘 (D:) > repository > com > alibaba > druid > 1.2.8					gav查找本地依赖位置	
<input type="checkbox"/>	名称	修改日期	类型	大小		
<input type="checkbox"/>	_remote.repositories	2023/5/18 15:20	REPOSITORIES ...	1 KB		
<input type="checkbox"/>	druid-1.2.8.jar	2023/5/18 15:20	JAR 文件	3,616 KB		
<input type="checkbox"/>	druid-1.2.8.jar.sha1	2023/5/18 15:20	SHA1 文件	1 KB		
<input type="checkbox"/>	druid-1.2.8.pom	2023/5/18 15:20	POM 文件	15 KB		
<input type="checkbox"/>	druid-1.2.8.pom.sha1	2023/5/18 15:20	SHA1 文件	1 KB	发现内部包含 lastupdated 结尾的文件,删除即可!	

4. 或者可以将清除lastUpdated文件的操作写在一个脚本文件中，手动创建文件"clearLastUpdated.bat"，名字任意，但是后缀必须是bat，将以下内容复制到文件中

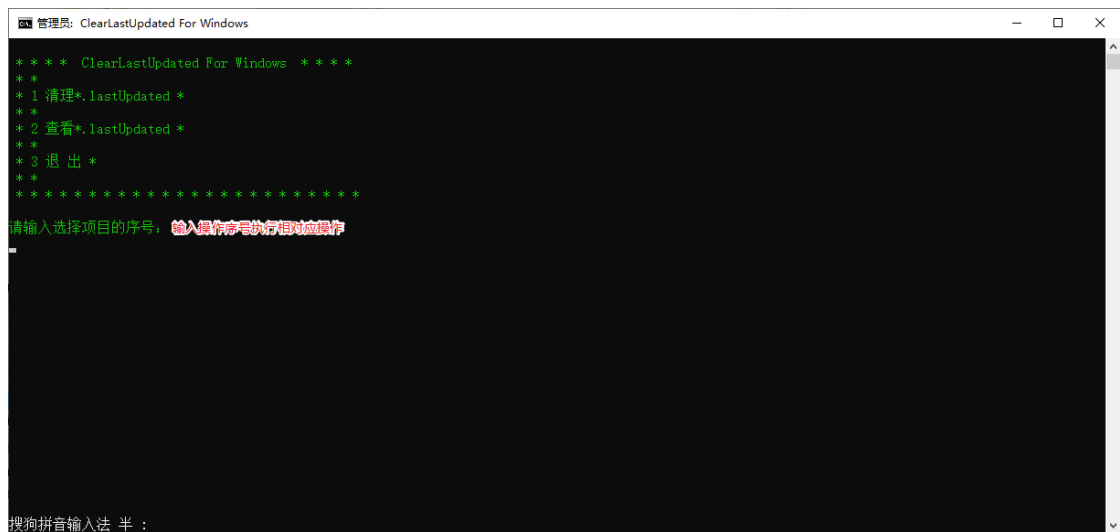
```
1  cls
2  @ECHO OFF
3  SET CLEAR_PATH=D:
4  SET CLEAR_DIR=D:\maven-repository(本地仓库路径)
5  color 0a
6  TITLE ClearLastUpdated For Windows
7  GOTO MENU
8  :MENU
9  CLS
10 ECHO.
11 ECHO. * * * * ClearLastUpdated For Windows * * * *
12 ECHO. * *
13 ECHO. * 1 清理*.lastUpdated *
14 ECHO. * *
15 ECHO. * 2 查看*.lastUpdated *
16 ECHO. * *
17 ECHO. * 3 退出 *
18 ECHO. * *
19 ECHO. * * * * *
20 ECHO.
21 ECHO. 请输入选择项目的序号:
22 set /p ID=
23 IF "%id%"=="1" GOTO cmd1
24 IF "%id%"=="2" GOTO cmd2
25 IF "%id%"=="3" EXIT
26 PAUSE
27 :cmd1
28 ECHO. 开始清理
```



```

29  %CLEAR_PATH%
30  cd %CLEAR_DIR%
31  for /r %%i in (*.lastUpdated) do del %%i
32  ECHO. OK
33  PAUSE
34  GOTO MENU
35  :cmd2
36  ECHO. 查看*.lastUpdated文件
37  %CLEAR_PATH%
38  cd %CLEAR_DIR%
39  for /r %%i in (*.lastUpdated) do echo %%i
40  ECHO. OK
41  PAUSE
42  GOTO MENU

```



6. Maven工程Build构建配置

项目构建是指将源代码、依赖库和资源文件等转换成可执行或可部署的应用程序的过程，在这个过程中包括编译源代码、链接依赖库、打包和部署等多个步骤。

默认情况下，构建不需要额外配置，都有对应的缺省配置。当然了，我们也可以在pom.xml定制一些配置，来修改默认构建的行为和产物！

例如：

1. 指定构建打包文件的名称，非默认名称
2. 制定构建打包时，指定包含文件格式和排除文件
3. 打包插件版本过低，配置更高版本插件

构建配置是在pom.xml / build标签中指定！

指定打包命名

```

1  <!-- 默认的打包名称：artifactId+version. 打包方式 -->
2  <build>
3      <finalName>定义打包名称</finalName>
4  </build>

```

指定打包文件

如果在java文件夹中添加java类，会自动打包编译到classes文件夹下！

但是在java文件夹中添加xml文件，默认不会被打包！

默认情况下，按照maven工程结构放置的文件会默认被编译和打包！

除此之外、我们可以使用resources标签，指定要打包资源的文件夹要把哪些静态资源打包到 classes根目录下！

应用场景：mybatis中有时会用于编写SQL语句的映射文件和mapper接口都写在src/main/java下的某个包中，此时映射文件就不会被打包，如何解决

```
1   <build>
2       <!--设置要打包的资源位置-->
3       <resources>
4           <resource>
5               <!--设置资源所在目录-->
6               <directory>src/main/java</directory>
7               <includes>
8                   <!--设置包含的资源类型-->
9                   <include>/**/*.xml</include>
10              </includes>
11          </resource>
12      </resources>
13  </build>
```

配置依赖插件

dependencies标签下引入开发需要的jar包！我们可以在build/plugins/plugin标签引入插件！

常用的插件：修改jdk版本、tomcat插件、mybatis分页插件、mybatis逆向工程插件等等！

```
1   <build>
2       <plugins>
3           <!-- java编译插件，配jdk的编译版本 -->
4           <plugin>
5               <groupId>org.apache.maven.plugins</groupId>
6               <artifactId>maven-compiler-plugin</artifactId>
7               <configuration>
8                   <source>1.8</source>
9                   <target>1.8</target>
10                  <encoding>UTF-8</encoding>
11              </configuration>
12          </plugin>
13          <!-- tomcat插件 -->
14          <plugin>
15              <groupId>org.apache.tomcat.maven</groupId>
16              <artifactId>tomcat7-maven-plugin</artifactId>
17              <version>2.2</version>
18              <configuration>
19                  <port>8090</port>
20                  <path>/</path>
21                  <uriEncoding>UTF-8</uriEncoding>
22                  <server>tomcat7</server>
23              </configuration>
24          </plugin>
25      </plugins>
26  </build>
```

六、Maven依赖传递和依赖冲突

1. Maven依赖传递特性

概念

假如有Maven项目A，项目B依赖A，项目C依赖B。那么我们可以说 C依赖A。也就是说，依赖的关系为：C → B → A，那么我们执行项目C时，会自动把B、A都下载导入到C项目的jar包文件夹中，这就是依赖的传递性。

作用

- 简化依赖导入过程
- 确保依赖版本正确

传递的原则

在 A 依赖 B，B 依赖 C 的前提下，C 是否能够传递到 A，取决于 B 依赖 C 时使用的依赖范围以及配置

- B 依赖 C 时使用 compile 范围：可以传递
- B 依赖 C 时使用 test 或 provided 范围：不能传递，所以需要这样的 jar 包时，就必须在需要的地方明确配置依赖才可以。
- B 依赖 C 时，若配置了以下标签，则不能传递

```
1 <dependency>
2     <groupId>com.alibaba</groupId>
3     <artifactId>druid</artifactId>
4     <version>1.2.15</version>
5     <optional>true</optional>
6 </dependency>
```

依赖传递终止

- 非compile范围进行依赖传递
- 使用optional配置终止传递
- 依赖冲突（传递的依赖已经存在）

案例：导入jackson依赖

分析：jackson需要三个依赖



1. Jackson Databind

com.fasterxml.jackson.core » [jackson-databind](#)

General data-binding functionality for Jackson: works on core streaming API

Last Release on Apr 24, 2023



2. Jackson Core

com.fasterxml.jackson.core » [jackson-core](#)

Core Jackson processing abstractions (aka Streaming API), implementation for JSON

Last Release on Apr 23, 2023



3. Jackson Annotations

com.fasterxml.jackson.core » [jackson-annotations](#)

Core annotations used for value types, used by Jackson data binding package.

Last Release on Apr 23, 2023

依赖传递关系：data-bind中，依赖其他两个依赖

Compile Dependencies (2)

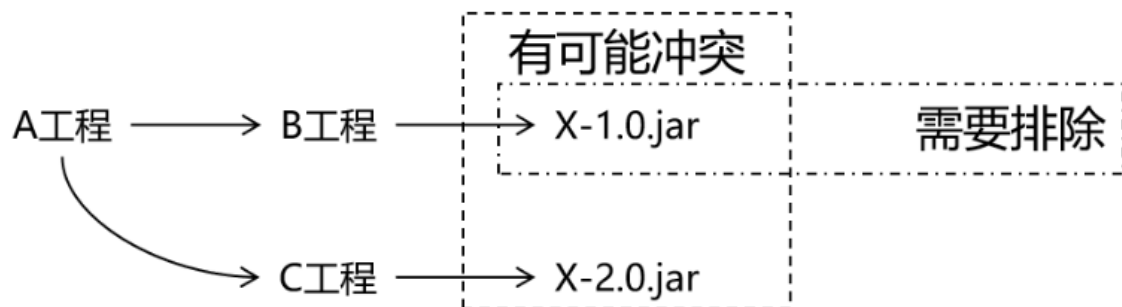
Category/License	Group / Artifact	Version	Updates
 Annotation Lib Apache 2.0	 com.fasterxml.jackson.core » jackson-annotations	2.10.0	2.15.0
 JSON Lib Apache 2.0	 com.fasterxml.jackson.core » jackson-core	2.10.0	2.15.0

最佳导入：直接可以导入data-bind，自动依赖传递需要的依赖

```
1 <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
2 <dependency>
3     <groupId>com.fasterxml.jackson.core</groupId>
4     <artifactId>jackson-databind</artifactId>
5     <version>2.10.0</version>
6 </dependency>
7
```

2. Maven依赖冲突特性

当直接引用或者间接引用出现了相同的jar包! 这时呢，一个项目就会出现相同的重复jar包，这就算作冲突! 依赖冲突避免出现重复依赖，并且终止依赖传递!



maven自动解决依赖冲突问题能力，会按照自己的原则，进行重复依赖选择。同时也提供了手动解决的冲突的方式，不过不推荐!

解决依赖冲突（如何选择重复依赖）方式：

1. 自动选择原则

- 短路优先原则（第一原则）

A→B→C→D→E→X(version 0.0.1)

A→F→X(version 0.0.2)

则A依赖于X(version 0.0.2)。

- 依赖路径长度相同情况下，则“先声明优先”（第二原则）

A→E→X(version 0.0.1)

A→F→X(version 0.0.2)

在<dependencies></dependencies>中，先声明的，路径相同，会优先选择!

2. 手动排除

```
1 <dependency>
2     <groupId>com.atguigu.maven</groupId>
3     <artifactId>pro01-maven-java</artifactId>
4     <version>1.0-SNAPSHOT</version>
5     <scope>compile</scope>
6     <!-- 使用excludes标签配置依赖的排除 -->
7     <exclusions>
```

```
8      <!-- 在exclude标签中配置一个具体的排除 -->
9      <exclusion>
10         <!-- 指定要排除的依赖的坐标（不需要写version） -->
11         <groupId>commons-logging</groupId>
12         <artifactId>commons-logging</artifactId>
13     </exclusion>
14 </exclusions>
15 </dependency>
```

3. 小案例

伪代码如下：

```
1  前提：
2      A 1.1 -> B 1.1 -> C 1.1
3      F 2.2 -> B 2.2
4
5  pom声明：
6      F 2.2
7      A 1.1
```

请问最终会导入哪些依赖和对应版本？

七、Maven工程继承和聚合关系

1. Maven工程继承关系

1. 继承概念

Maven 继承是指在 Maven 的项目中，让一个项目从另一个项目中继承配置信息的机制。继承可以让我们在多个项目中共享同一配置信息，简化项目的管理和维护工作。

2. 继承作用

在父工程中统一管理项目中的依赖信息。

它的背景是：

- 对一个比较大型的项目进行了模块拆分。
- 一个 project 下面，创建了很多个 module。
- 每一个 module 都需要配置自己的依赖信息。

它背后的需求是：

- 在每一个 module 中各自维护各自的依赖信息很容易发生出入，不易统一管理。
- 使用同一个框架内的不同 jar 包，它们应该是同一个版本，所以整个项目中使用的框架版本需要统一。
- 使用框架时所需要的 jar 包组合（或者说依赖信息组合）需要经过长期摸索和反复调试，最终确定一个可用组合。这个耗费很大精力总结出来的方案不应该在新的项目中重新摸索。通过在父工程中为整个项目维护依赖信息的组合既保证了整个项目使用规范、准确的 jar 包；又能够将以往的经验沉淀下来，节约时间和精力。

3. 继承语法

- 父工程

```

1      <groupId>com.atguigu.maven</groupId>
2      <artifactId>pro03-maven-parent</artifactId>
3      <version>1.0-SNAPSHOT</version>
4      <!-- 当前工程作为父工程，它要去管理子工程，所以打包方式必须是 pom -->
5      <packaging>pom</packaging>
6

```

- 子工程

```

1      <!-- 使用parent标签指定当前工程的父工程 -->
2      <parent>
3          <!-- 父工程的坐标 -->
4          <groupId>com.atguigu.maven</groupId>
5          <artifactId>pro03-maven-parent</artifactId>
6          <version>1.0-SNAPSHOT</version>
7      </parent>
8
9      <!-- 子工程的坐标 -->
10     <!-- 如果子工程坐标中的groupId和version与父工程一致，那么可以省略 -->
11     <!-- <groupId>com.atguigu.maven</groupId> -->
12     <artifactId>pro04-maven-module</artifactId>
13     <!-- <version>1.0-SNAPSHOT</version> -->

```

4. 父工程依赖统一管理

- 父工程声明版本

```

1      <!-- 使用dependencyManagement标签配置对依赖的管理 -->
2      <!-- 被管理的依赖并没有真正被引入到工程 -->
3      <dependencyManagement>
4          <dependencies>
5              <dependency>
6                  <groupId>org.springframework</groupId>
7                  <artifactId>spring-core</artifactId>
8                  <version>6.0.10</version>
9              </dependency>
10             <dependency>
11                 <groupId>org.springframework</groupId>
12                 <artifactId>spring-beans</artifactId>
13                 <version>6.0.10</version>
14             </dependency>
15             <dependency>
16                 <groupId>org.springframework</groupId>
17                 <artifactId>spring-context</artifactId>
18                 <version>6.0.10</version>
19             </dependency>
20             <dependency>
21                 <groupId>org.springframework</groupId>
22                 <artifactId>spring-expression</artifactId>
23                 <version>6.0.10</version>
24             </dependency>
25             <dependency>
26                 <groupId>org.springframework</groupId>
27                 <artifactId>spring-aop</artifactId>
28                 <version>6.0.10</version>
29             </dependency>
30         </dependencies>
31     </dependencyManagement>

```

- 子工程引用版本

```
1  <!-- 子工程引用父工程中的依赖信息时，可以把版本号去掉。 -->
2  <!-- 把版本号去掉就表示子工程中这个依赖的版本由父工程决定。 -->
3  <!-- 具体来说是由父工程的dependencyManagement来决定。 -->
4  <dependencies>
5      <dependency>
6          <groupId>org.springframework</groupId>
7          <artifactId>spring-core</artifactId>
8      </dependency>
9      <dependency>
10         <groupId>org.springframework</groupId>
11         <artifactId>spring-beans</artifactId>
12     </dependency>
13     <dependency>
14         <groupId>org.springframework</groupId>
15         <artifactId>spring-context</artifactId>
16     </dependency>
17     <dependency>
18         <groupId>org.springframework</groupId>
19         <artifactId>spring-expression</artifactId>
20     </dependency>
21     <dependency>
22         <groupId>org.springframework</groupId>
23         <artifactId>spring-aop</artifactId>
24     </dependency>
25 </dependencies>
```

2. Maven工程聚合关系

1. 聚合概念

Maven 聚合是指将多个项目组织到一个父级项目中，以便一起构建和管理的机制。聚合可以帮助我们更好地管理一组相关的子项目，同时简化它们的构建和部署过程。

2. 聚合作用

1. 管理多个子项目：通过聚合，可以将多个子项目组织在一起，方便管理和维护。
2. 构建和发布一组相关的项目：通过聚合，可以在一个命令中构建和发布多个相关的项目，简化了部署和维护工作。
3. 优化构建顺序：通过聚合，可以对多个项目进行顺序控制，避免出现构建依赖混乱导致构建失败的情况。
4. 统一管理依赖项：通过聚合，可以在父项目中管理公共依赖项和插件，避免重复定义。

3. 聚合语法

父项目中包含的子项目列表。

```
1  <project>
2      <groupId>com.example</groupId>
3      <artifactId>parent-project</artifactId>
4      <packaging>pom</packaging>
5      <version>1.0.0</version>
6      <modules>
7          <module>child-project1</module>
8          <module>child-project2</module>
9      </modules>
10 </project>
```

4. 聚合演示

通过触发父工程构建命令、引发所有子模块构建！产生反应堆！

八、Maven私服

1. Maven私服简介

①私服简介

Maven 私服是一种特殊的Maven远程仓库，它是架设在局域网内的仓库服务，用来代理位于外部的远程仓库（中央仓库、其他远程公共仓库）。

当然也并不是说私服只能建立在局域网，也有很多公司会直接把私服部署到公网，具体还是得看公司业务性质是否是保密的等等，因为局域网的话只能在公司用，部署到公网的话员工在家里也可以办公使用。

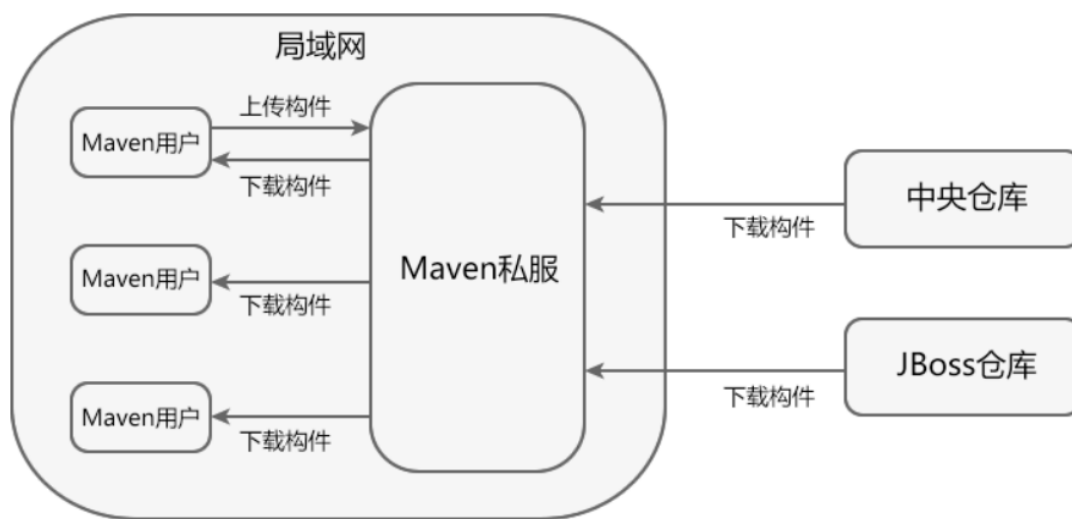
建立了 Maven 私服后，当局域网内的用户需要某个构件时，会按照如下顺序进行请求和下载。

请求本地仓库，若本地仓库不存在所需构件，则跳转到第 2 步；

请求 Maven 私服，将所需构件下载到本地仓库，若私服中不存在所需构件，则跳转到第 3 步。

请求外部的远程仓库，将所需构件下载并缓存到 Maven 私服，若外部远程仓库不存在所需构件，则 Maven 直接报错。

此外，一些无法从外部仓库下载到的构件，也能从本地上传到私服供其他人使用。



②Maven私服的优势

1. 节省外网带宽

消除对外部远程仓库的大量重复请求（会消耗很大的带宽），降低外网带宽压力。

2. 下载速度更快

Maven私服位于局域网内，从私服下载构件更快更稳定。

3. 便于部署第三方构件

有些构件无法从任何一个远程仓库中获得（如：公司或组织内部的私有构件、Oracle的JDBC驱动等），建立私服之后，就可以将这些构件部署到私服中，供内部Maven项目使用。

4. 提高项目的稳定性，增强对项目的控制

如果不建立私服，那么Maven项目的构件就高度依赖外部的远程仓库，若外部网络不稳定，则项目的构建过程也会变得不稳定。建立私服后，即使外部网络状况不佳甚至中断，只要私服中已经缓存了所需的构件，Maven也能够正常运行。私服软件（如：Nexus）提供了很多控制功能（如：权限管理、RELEASE/SNAPSHOT版本控制等），可以对仓库进行一些更加高级的控制。

5. 降低中央仓库得负荷压力

由于私服会缓存中央仓库得构件，避免了很多对中央仓库的重复下载，降低了中央仓库的负荷。

③常见的Maven私服产品

1. Apache的Archiva
2. JFrog的Artifactory
3. Sonatype的Nexus ([ˈneksəs]) （当前最流行、使用最广泛）

2. Nexus下载安装

下载地址：<https://help.sonatype.com/repomanager3/product-information/download>

解压，以管理员身份打开CMD，进入bin目录下，执行./nexus /run命令启动

访问 Nexus 首页

首页地址：<http://localhost:8081/>，8081为默认端口号



3. 初始设置

点这里登录

Sonatype Nexus Repository Manager
OSS 3.37.0-01

Sign in

Browse
Welcome
Search
Browse

Browse
Browse assets and components

	Name	Type	Format	Status	URL ↑	Health c...	
	maven-central	proxy	maven2	Online - R...	copy		➤
	maven-public	group	maven2	Online	copy		➤
	maven-releases	hosted	maven2	Online	copy		➤
	maven-snapshots	hosted	maven2	Online	copy		➤
	nuget-group	group	nuget	Online	copy		➤
	nuget-hosted	hosted	nuget	Online	copy		➤
	nuget.org-proxy	proxy	nuget	Online - R...	copy		➤

Sign In
✕

Your **admin** user password is located in
E:\Server\nexus-3.61.0-02-win64\sonatype-work\nexus3\admin.password on the server.

Sign in
Cancel

这里参考提示：

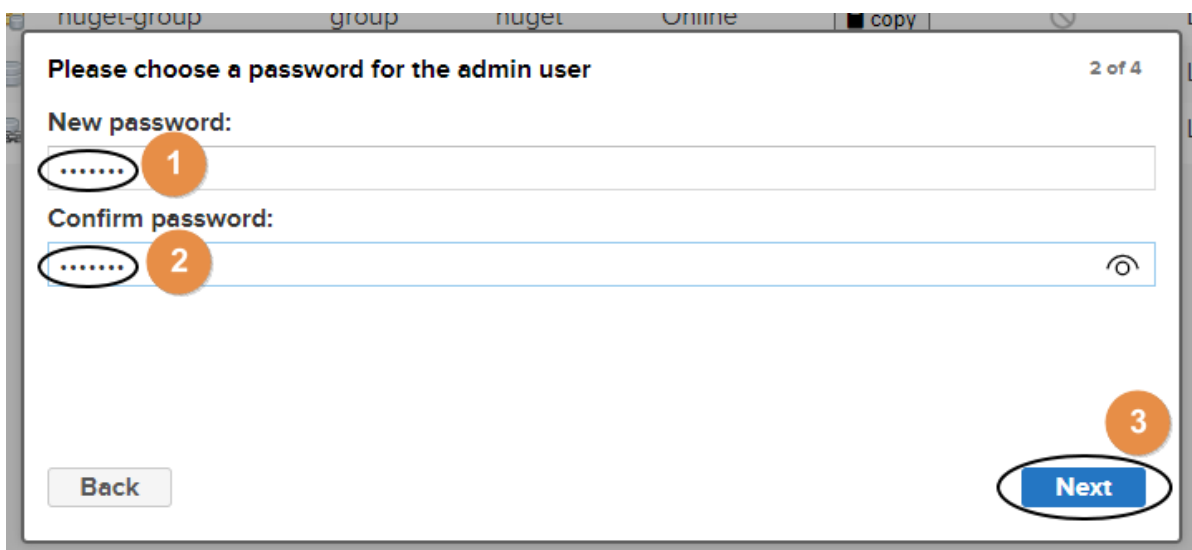
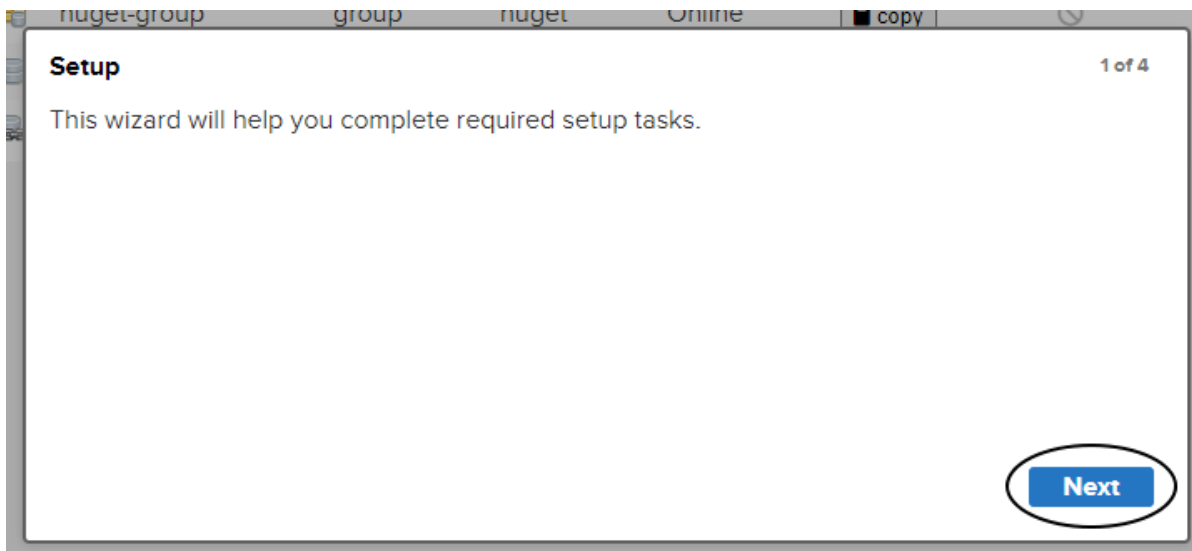
- 用户名：admin
- 密码：查看 **E:\Server\nexus-3.61.0-02-win64\sonatype-work\nexus3\admin.password** 文件

Sign In
✕

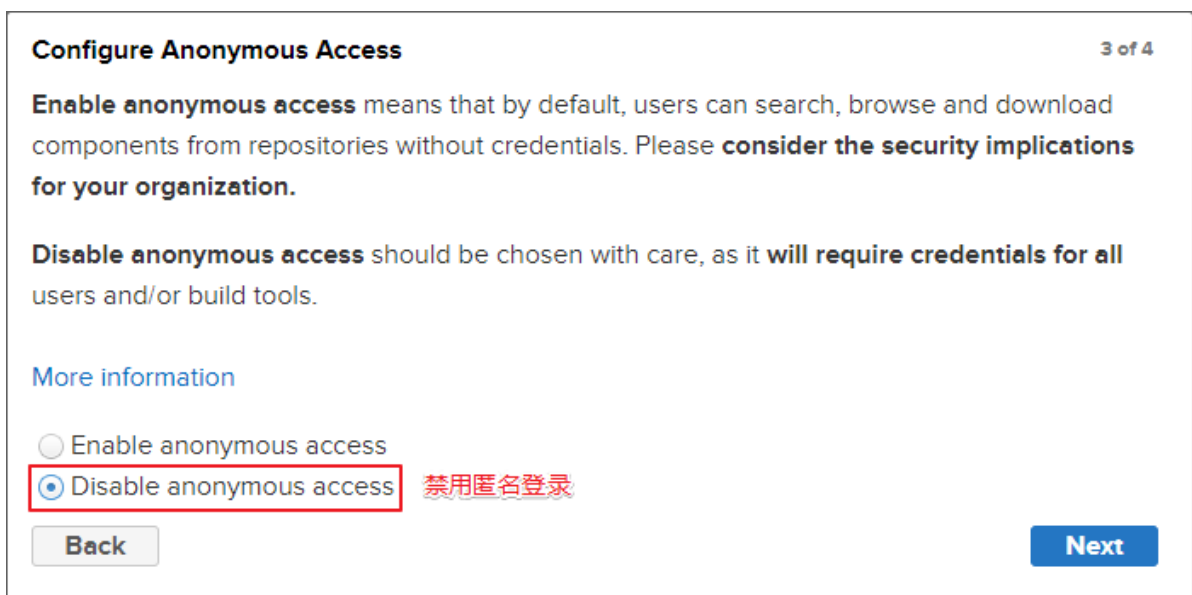
Your **admin** user password is located in
E:\Server\nexus-3.61.0-02-win64\sonatype-work\nexus3\admin.password on the server.

Sign in
Cancel

继续执行初始化：



匿名登录，启用还是禁用？由于启用匿名登录后，后续操作比较简单，这里我们演示禁用匿名登录的操作：

















初始化完毕：

The setup tasks have been completed, enjoy using Nexus Repository Manager!

[Finish](#)

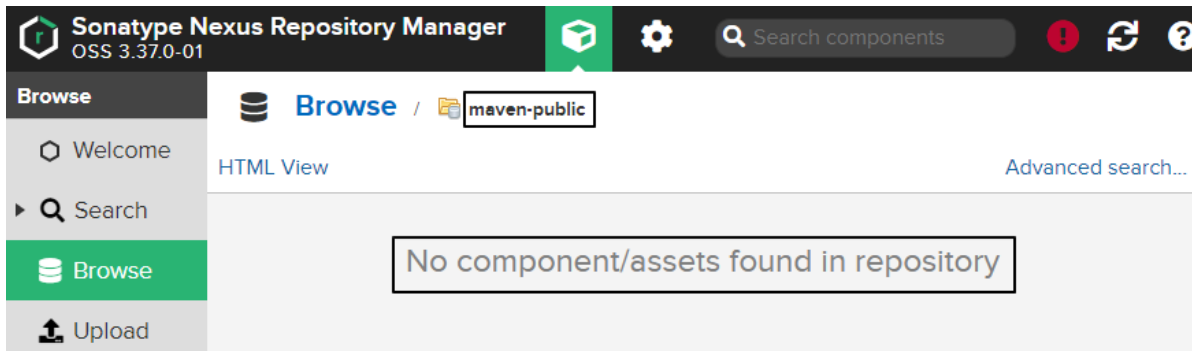
4. Nexus上的各种仓库

	Name	Type	Format	Status	URL ↑
	maven-central	proxy	maven2	Online - R...	 copy
	maven-public	group	maven2	Online	 copy
	maven-releases	hosted	maven2	Online	 copy
	maven-snapshots	hosted	maven2	Online	 copy
	nuget-group	group	nuget	Online	 copy
	nuget-hosted	hosted	nuget	Online	 copy
	nuget.org-proxy	proxy	nuget	Online - R...	 copy

仓库类型	说明
proxy	某个远程仓库的代理
group	存放：通过 Nexus 获取的第三方 jar 包
hosted	存放：本团队其他开发人员部署到 Nexus 的 jar 包

仓库名称	说明
maven-central	Nexus 对 Maven 中央仓库的代理
maven-public	Nexus 默认创建，供开发人员下载使用的组仓库
maven-releases	Nexus 默认创建，供开发人员部署自己 jar 包的宿主仓库 要求 releases 版本
maven-snapshots	Nexus 默认创建，供开发人员部署自己 jar 包的宿主仓库 要求 snapshots 版本

初始状态下，这几个仓库都没有内容：



5. 通过 Nexus 下载 jar 包

修改本地maven的核心配置文件settings.xml，设置新的本地仓库地址

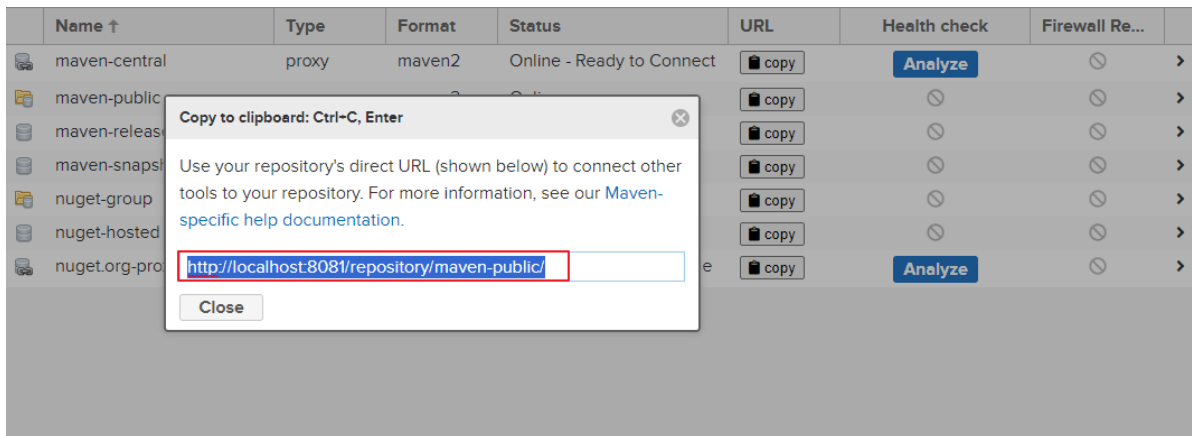
```
1 <!-- 配置一个新的 Maven 本地仓库 -->
2 <localRepository>D:/maven-repository-new</localRepository>
```

把我们原来配置阿里云仓库地址的 mirror 标签改成下面这样：

```
1 <mirror>
2   <id>nexus-mine</id>
3   <mirrorOf>central</mirrorOf>
4   <name>Nexus mine</name>
5   <url>http://localhost:8081/repository/maven-public/</url>
6 </mirror>
```

这里的 url 标签是这么来的：

	Name	Type	Format	Status	URL	Health check
	maven-central	proxy	maven2	Online - R...	copy	Analyze
	maven-public	group	maven2	Online	copy	
	maven-releases	hosted	maven2	Online	copy	
	maven-snapshots	hosted	maven2	Online	copy	
	nuget-group	group	nuget	Online	copy	
	nuget-hosted	hosted	nuget	Online	copy	
	nuget.org-proxy	proxy	nuget	Online - R...	copy	Analyze



把上图中看到的地址复制出来即可。如果我们在前面允许了匿名访问，到这里就够了。但如果我们禁用了匿名访问，那么接下来我们还要继续配置 settings.xml：

```
1 <server>
2   <id>nexus-mine</id>
3   <username>admin</username>
4   <password>atguigu</password>
5 </server>
```

这里需要**格外注意**：server 标签内的 id 标签值必须和 mirror 标签中的 id 值一样。

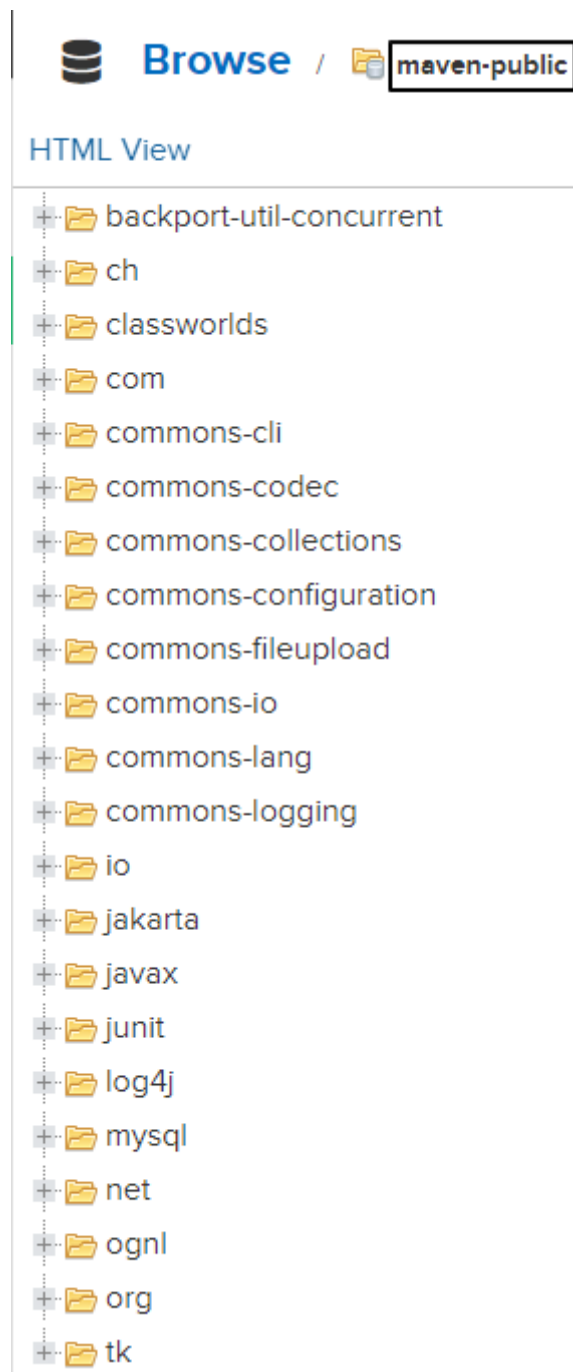
找一个用到框架的 Maven 工程，执行命令：

```
1 mvn clean compile
```

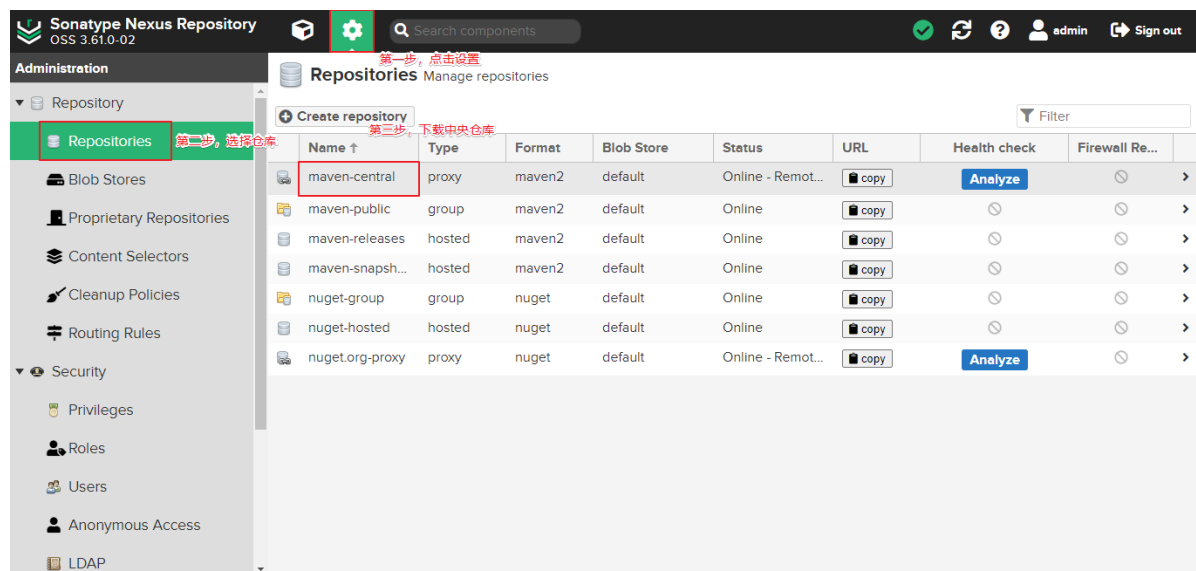
下载过程日志：

```
Downloading from nexus-mine: http://localhost:8081/repository/maven-public/com/jayway/jsonpath/json-path/2.4.0/json-path-2.4.0.pom
Downloaded from nexus-mine: http://localhost:8081/repository/maven-public/com/jayway/jsonpath/json-path/2.4.0/json-path-2.4.0.pom (2.6 kB at 110 kB/s)
Downloading from nexus-mine: http://localhost:8081/repository/maven-public/net/minidev/json-smart/2.3/json-smart-2.3.pom
Downloaded from nexus-mine: http://localhost:8081/repository/maven-public/net/minidev/json-smart/2.3/json-smart-2.3.pom (9.0 kB at 376 kB/s)
Downloading from nexus-mine: http://localhost:8081/repository/maven-public/net/minidev/minidev-parent/2.3/minidev-parent-2.3.pom
Downloaded from nexus-mine: http://localhost:8081/repository/maven-public/net/minidev/minidev-parent/2.3/minidev-parent-2.3.pom (8.5 kB at 404 kB/s)
Downloading from nexus-mine: http://localhost:8081/repository/maven-public/net/minidev/accessors-smart/1.2/accessors-smart-1.2.pom
Downloaded from nexus-mine: http://localhost:8081/repository/maven-public/net/minidev/accessors-smart/1.2/accessors-smart-1.2.pom (12 kB at 463 kB/s)
```

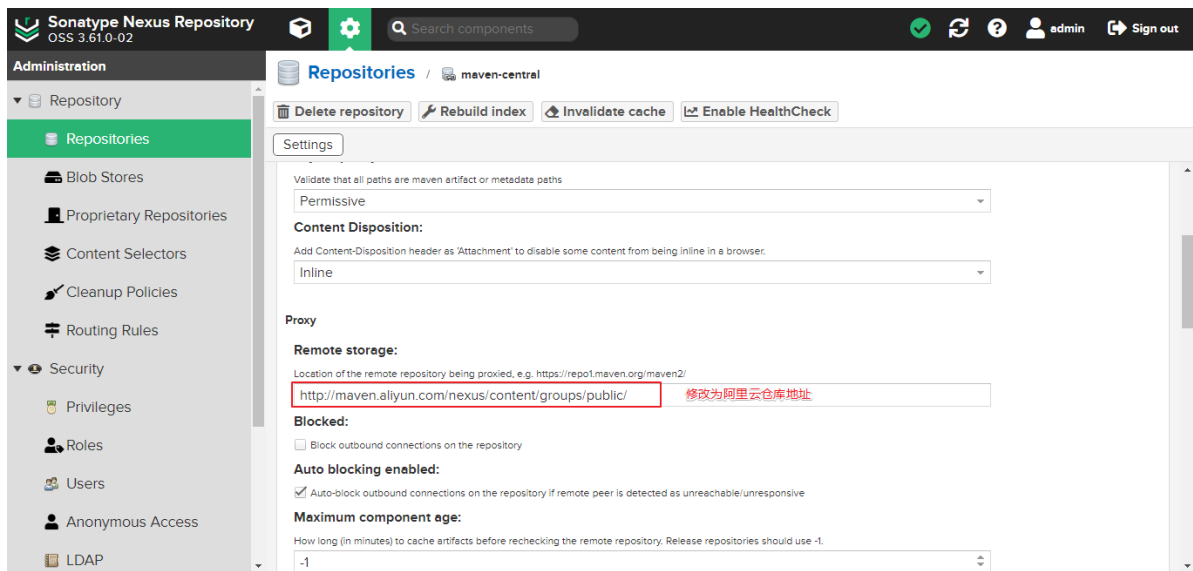
下载后，Nexus 服务器上就有了 jar 包：



若下载速度太慢，可以设置私服中中央仓库的地址为阿里云仓库地址



修改为: <http://maven.aliyun.com/nexus/content/groups/public/>



6. 将jar包部署到 Nexus

maven工程中配置：

```
1 <distributionManagement>
2   <snapshotRepository>
3     <id>nexus-mine</id>
4     <name>Nexus Snapshot</name>
5     <url>http://localhost:8081/repository/maven-snapshots/</url>
6   </snapshotRepository>
7 </distributionManagement>
```

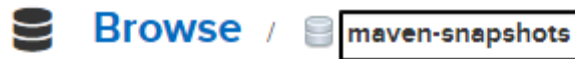
注意：这里 snapshotRepository 的 id 标签必须和 settings.xml 中指定的 mirror 标签的 id 属性一致。

执行部署命令：

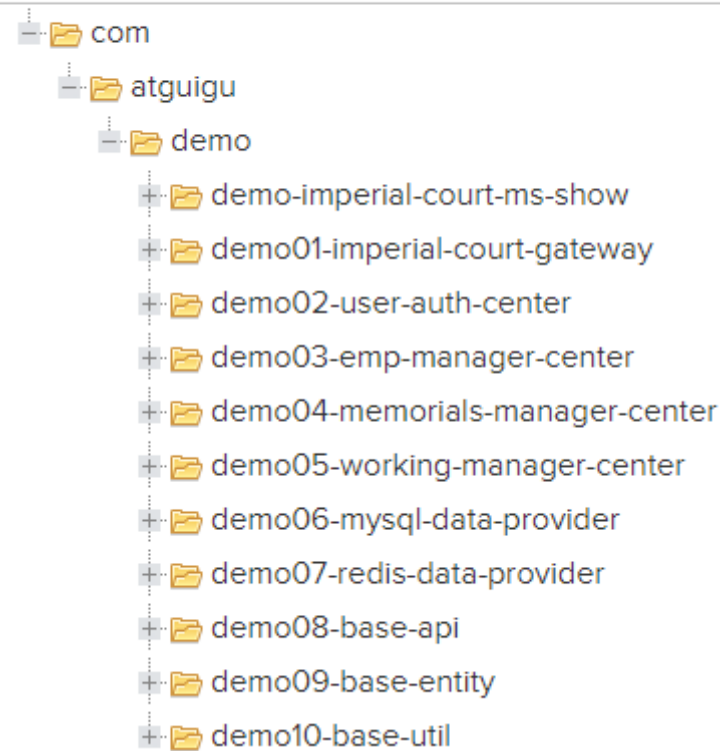
```
1 mvn deploy
```

```
Uploading to nexus-mine: http://localhost:8081/repository/maven-snapshots/com/atguigu/demo/d
emo07-redis-data-provider/1.0-SNAPSHOT/maven-metadata.xml
Uploaded to nexus-mine: http://localhost:8081/repository/maven-snapshots/com/atguigu/demo/de
mo07-redis-data-provider/1.0-SNAPSHOT/maven-metadata.xml (786 B at 19 kB/s)
Uploading to nexus-mine: http://localhost:8081/repository/maven-snapshots/com/atguigu/demo/d
emo07-redis-data-provider/maven-metadata.xml
Uploaded to nexus-mine: http://localhost:8081/repository/maven-snapshots/com/atguigu/demo/de
mo07-redis-data-provider/maven-metadata.xml (300 B at 6.5 kB/s)
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] demo-imperial-court-ms-show 1.0-SNAPSHOT ..... SUCCESS [ 1.875 s]
[INFO] demo09-base-entity ..... SUCCESS [ 21.883 s]
[INFO] demo10-base-util ..... SUCCESS [ 0.324 s]
[INFO] demo08-base-api ..... SUCCESS [ 1.171 s]
[INFO] demo01-imperial-court-gateway ..... SUCCESS [ 0.403 s]
[INFO] demo02-user-auth-center ..... SUCCESS [ 2.932 s]
[INFO] demo03-emp-manager-center ..... SUCCESS [ 0.312 s]
[INFO] demo04-memorials-manager-center ..... SUCCESS [ 0.362 s]
[INFO] demo05-working-manager-center ..... SUCCESS [ 0.371 s]
```

```
[INFO] demo06-mysql-data-provider ..... SUCCESS [ 6.779 s]
[INFO] demo07-redis-data-provider 1.0-SNAPSHOT ..... SUCCESS [ 0.273 s]
```



HTML View



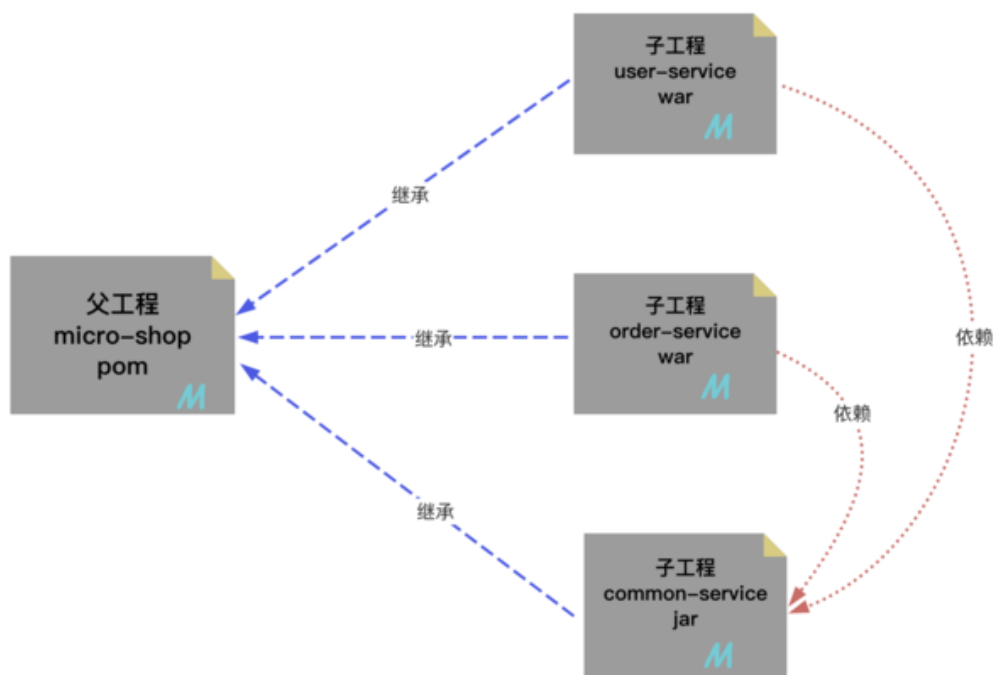
7. 引用别人部署的 jar 包

maven工程中配置：

```
1  <repositories>
2    <repository>
3      <id>nexus-mine</id>
4      <name>nexus-mine</name>
5      <url>http://localhost:8081/repository/maven-snapshots/</url>
6      <snapshots>
7        <enabled>true</enabled>
8      </snapshots>
9      <releases>
10       <enabled>true</enabled>
11     </releases>
12   </repository>
13 </repositories>
```

九、Maven综合案例

1. 项目需求和结构分析



需求案例：搭建一个电商平台项目，该平台包括用户服务、订单服务、通用工具模块等。

项目架构：

1. 用户服务：负责处理用户相关的逻辑，例如用户信息的管理、用户注册、登录等。
 - spring-context 6.0.6
 - spring-core 6.0.6
 - spring-beans 6.0.6
 - common-service
2. 订单服务：负责处理订单相关的逻辑，例如订单的创建、订单支付、退货、订单查看等。
 - spring-context 6.0.6
 - spring-core 6.0.6
 - spring-beans 6.0.6
 - spring-security 6.0.6
 - common-service
3. 通用模块：负责存储其他服务需要通用工具类，其他服务依赖此模块。
 - commons-io 2.11.0
 - junit 5.9.2

2. 项目搭建和统一构建

①父模块 (micro-shop)

创建工程：

New Module

New Module

Generators

Maven Archetype

Jakarta EE

Spring Initializr

JavaFX

Quarkus

Micronaut

Ktor

Compose Multiplatform

HTML

React

Express

Angular CLI

IDE Plugin

Android

Name:

micro-shop

Location:

D:\IDEA\workspace\atguigu_test

Module will be created in: D:\IDEA\workspace\atguigu_test\micro-shop

Language:

Java

Groovy

JavaScript

+

Build system:

IntelliJ

Maven

Gradle

JDK:

Project SDK 17

Parent:

<None>

☐ Add sample code

Advanced Settings

下面输入maven工程坐标

GroupId:

com.atguigu

ArtifactId:

micro-shop

Create

Cancel

pom.xml配置:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6
7      <groupId>com.atguigu</groupId>
8      <artifactId>micro-shop</artifactId>
9      <version>1.0-SNAPSHOT</version>
10     <!--知识点：父工程的打包方式为pom-->
11     <packaging>pom</packaging>
12
13     <properties>
14         <spring.version>6.0.6</spring.version>
15         <jackson.version>2.15.0</jackson.version>
16         <commons.version>2.11.0</commons.version>
17         <junit.version>5.9.2</junit.version>
18         <maven.compiler.source>17</maven.compiler.source>
19         <maven.compiler.target>17</maven.compiler.target>
20         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
21     </properties>
22
23     <!-- 依赖管理 -->
24     <dependencyManagement>
25         <dependencies>
26             <!-- spring-context会依赖传递core/beans -->
27             <dependency>
28                 <groupId>org.springframework</groupId>

```

```

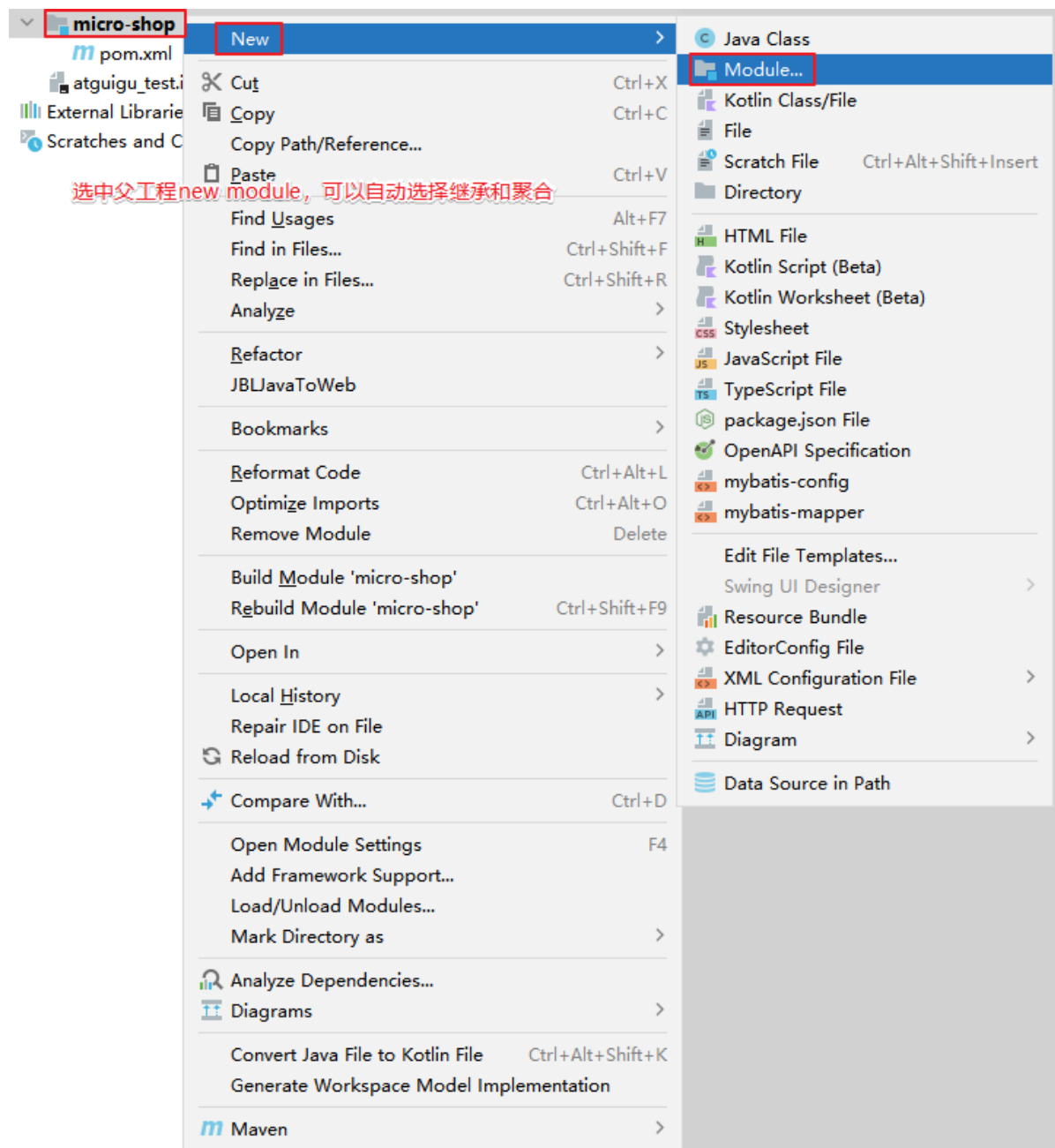
29         <artifactId>spring-context</artifactId>
30         <version>${spring.version}</version>
31     </dependency>
32
33     <!-- jackson-databind会依赖传递core/annotations -->
34     <dependency>
35         <groupId>com.fasterxml.jackson.core</groupId>
36         <artifactId>jackson-databind</artifactId>
37         <version>${jackson.version}</version>
38     </dependency>
39
40     <!-- commons-io -->
41     <dependency>
42         <groupId>commons-io</groupId>
43         <artifactId>commons-io</artifactId>
44         <version>${commons.version}</version>
45     </dependency>
46
47     <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
48     <dependency>
49         <groupId>org.junit.jupiter</groupId>
50         <artifactId>junit-jupiter-api</artifactId>
51         <version>${junit.version}</version>
52         <scope>test</scope>
53     </dependency>
54
55     </dependencies>
56 </dependencyManagement>
57
58 <!-- 统一更新子工程打包插件-->
59 <build>
60     <!-- jdk17 和 war包版本插件不匹配 -->
61     <plugins>
62         <plugin>
63             <groupId>org.apache.maven.plugins</groupId>
64             <artifactId>maven-war-plugin</artifactId>
65             <version>3.2.2</version>
66         </plugin>
67     </plugins>
68 </build>
69
70 </project>

```

可选操作：删除src目录

②通用模块 (common-service)

创建工程：



New Module

New Module

Generators

Maven Archetype

Jakarta EE

Spring Initializr

JavaFX

Quarkus

Micronaut

Ktor

Compose Multiplatform

HTML

React

Express

Angular CLI

IDE Plugin

Android

Name:

common-service

Location:

D:\IDEA\workspace\atguigu_test\micro-shop

Module will be created in: D:\IDEA\workspace\atguigu_test\micro-shop\common

Language:

Java

Groovy

JavaScript

+

Build system:

IntelliJ

Maven

Gradle

JDK:

Project SDK 17

Parent:

micro-shop

☐ Add sample code

Advanced Settings

GroupId:

com.atguigu 和父工程保持一致

ArtifactId:

common-service

?

Create

Cancel

pom.xml配置:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <parent>
6          <artifactId>micro-shop</artifactId>
7          <groupId>com.atguigu</groupId>
8          <version>1.0-SNAPSHOT</version>
9      </parent>
10     <modelVersion>4.0.0</modelVersion>
11
12     <artifactId>common-service</artifactId>
13     <!--知识点: 打包方式默认就是jar, 因此可以省略-->
14     <packaging>jar</packaging>
15
16     <properties>
17         <maven.compiler.source>17</maven.compiler.source>
18         <maven.compiler.target>17</maven.compiler.target>
19         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
20     </properties>
21
22     <dependencies>
23         <!-- 配置spring-context, 继承父工程版本, 自动传递 core / beans -->
24         <dependency>
25             <groupId>org.springframework</groupId>
26             <artifactId>spring-context</artifactId>
27         </dependency>
28         <!-- 配置jackson-databind, 继承父工程版本, 自动传递 core / annotations -->

```



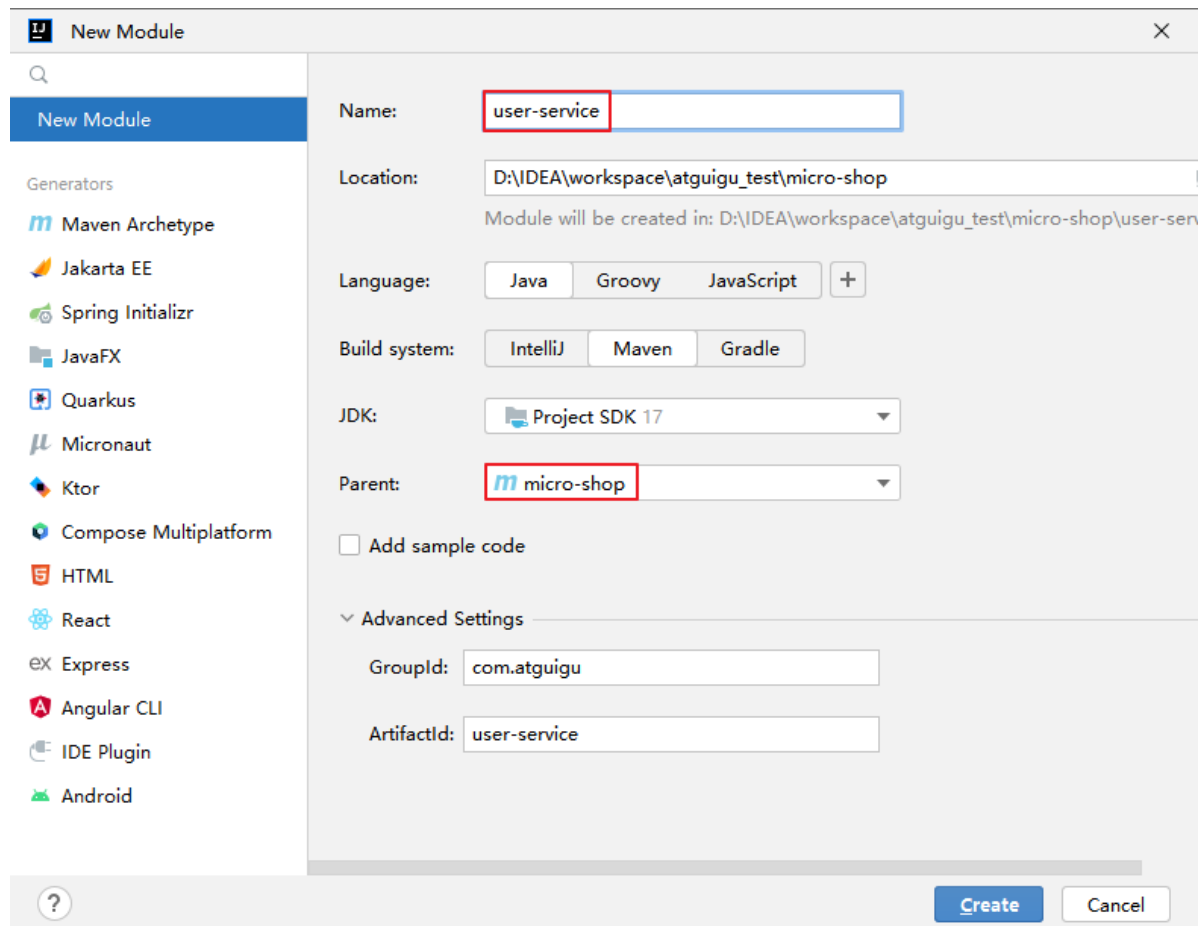
```

29         <dependency>
30             <groupId>com.fasterxml.jackson.core</groupId>
31             <artifactId>jackson-databind</artifactId>
32         </dependency>
33         <!-- 配置commons-io, 继承父工程版本 -->
34         <dependency>
35             <groupId>commons-io</groupId>
36             <artifactId>commons-io</artifactId>
37         </dependency>
38         <!-- 配置junit, 继承父工程版本 -->
39         <dependency>
40             <groupId>org.junit.jupiter</groupId>
41             <artifactId>junit-jupiter-api</artifactId>
42             <scope>test</scope>
43         </dependency>
44     </dependencies>
45
46 </project>

```

③用户模块 (user-service)

创建工程：



New Module

Search:

New Module

Generators

- Maven Archetype
- Jakarta EE
- Spring Initializr
- JavaFX
- Quarkus
- Micronaut
- Ktor
- Compose Multiplatform
- HTML
- React
- Express
- Angular CLI
- IDE Plugin
- Android

Name:

Location:

Module will be created in: D:\IDEA\workspace\atguigu_test\micro-shop\user-service

Language:

Build system:

JDK:

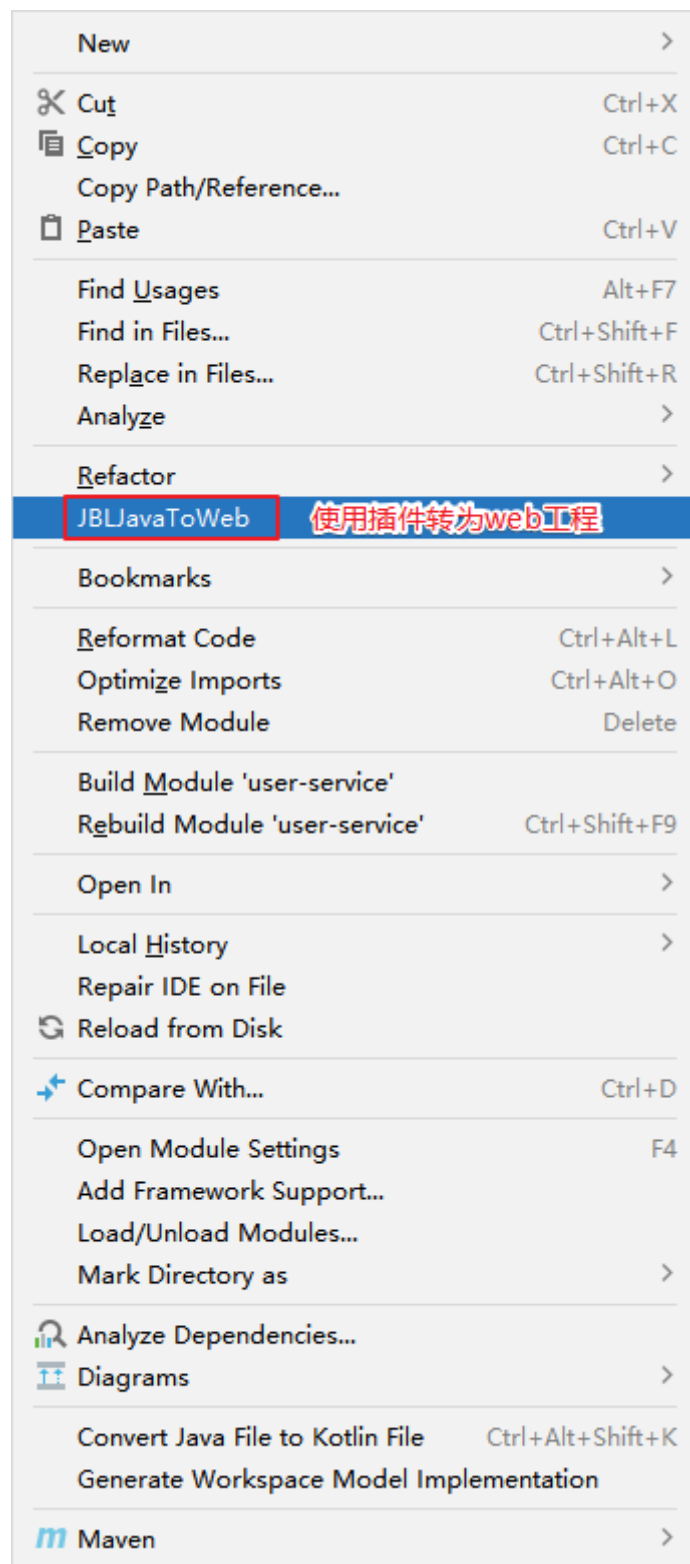
Parent:

☐ Add sample code

Advanced Settings

GroupId:

ArtifactId:



pom.xml配置:

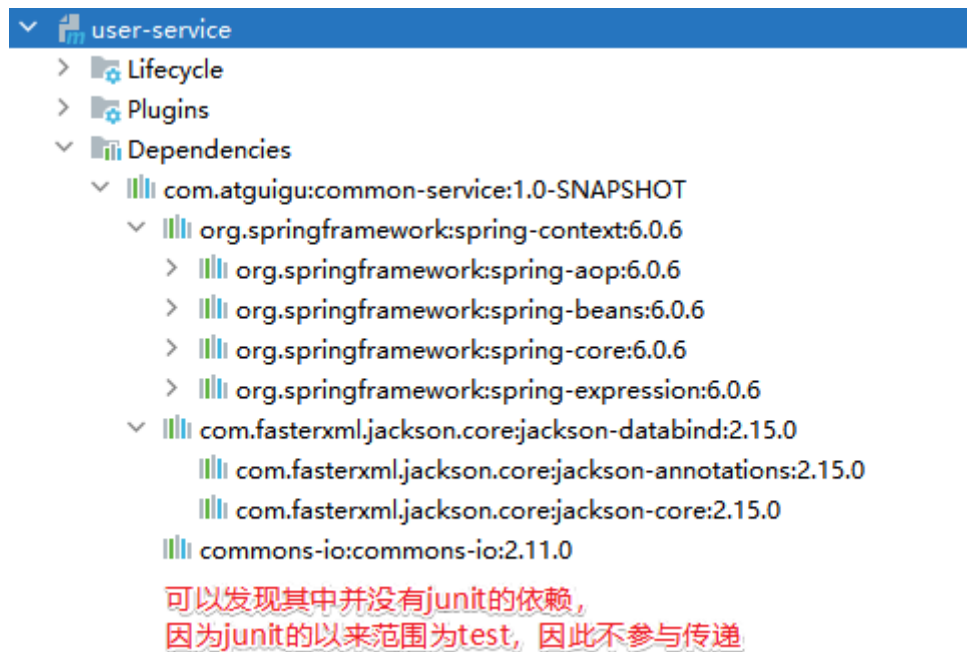
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5         http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <parent>
7         <artifactId>micro-shop</artifactId>
8         <groupId>com.atguigu</groupId>
9         <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
```

```

12     <artifactId>user-service</artifactId>
13     <!-- web工程打包方式为war -->
14     <packaging>war</packaging>
15
16     <properties>
17         <maven.compiler.source>17</maven.compiler.source>
18         <maven.compiler.target>17</maven.compiler.target>
19         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
20     </properties>
21
22     <dependencies>
23         <!-- 配置common-service，所需依赖会传递到当前工程（仅限compile范围） -->
24         <dependency>
25             <groupId>com.atguigu</groupId>
26             <artifactId>common-service</artifactId>
27             <version>1.0-SNAPSHOT</version>
28         </dependency>
29     </dependencies>
30
31 </project>

```

依赖传递结果：



④订单模块 (order-service)

创建工程，并使用插件转为web工程：

New Module

New Module

Generators

Maven Archetype

Jakarta EE

Spring Initializr

JavaFX

Quarkus

Micronaut

Ktor

Compose Multiplatform

HTML

React

Express

Angular CLI

IDE Plugin

Android

Name:order-service

Location:D:\IDEA\workspace\atguigu_test\micro-shop

Module will be created in: D:\IDEA\workspace\atguigu_test\micro-shop\order-se

Language:JavaGroovyJavaScript+

Build system:IntelliJ MavenGradle

JDK:Project SDK 17

Parent:micro-shop

☐ Add sample code

Advanced Settings

GroupId:com.atguigu

ArtifactId:order-service

?

Create

Cancel

pom.xml配置:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <parent>
6          <artifactId>micro-shop</artifactId>
7          <groupId>com.atguigu</groupId>
8          <version>1.0-SNAPSHOT</version>
9      </parent>
10     <modelVersion>4.0.0</modelVersion>
11
12     <artifactId>order-service</artifactId>
13     <!-- web工程打包方式为war -->
14     <packaging>war</packaging>
15
16     <properties>
17         <maven.compiler.source>17</maven.compiler.source>
18         <maven.compiler.target>17</maven.compiler.target>
19         <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
20     </properties>
21
22     <dependencies>
23         <!-- 配置common-service，所需依赖会传递到当前工程（仅限compile范围） -->
24         <dependency>
25             <groupId>com.atguigu</groupId>
26             <artifactId>common-service</artifactId>
27             <version>1.0-SNAPSHOT</version>
28         </dependency>
```

```
29         </dependencies>
30
31     </project>
```

此时，查看父工程的pom.xml，会发现其中已经自动聚合了子工程：

```
1     <modules>
2         <module>common-service</module>
3         <module>user-service</module>
4         <module>order-service</module>
5     </modules>
```