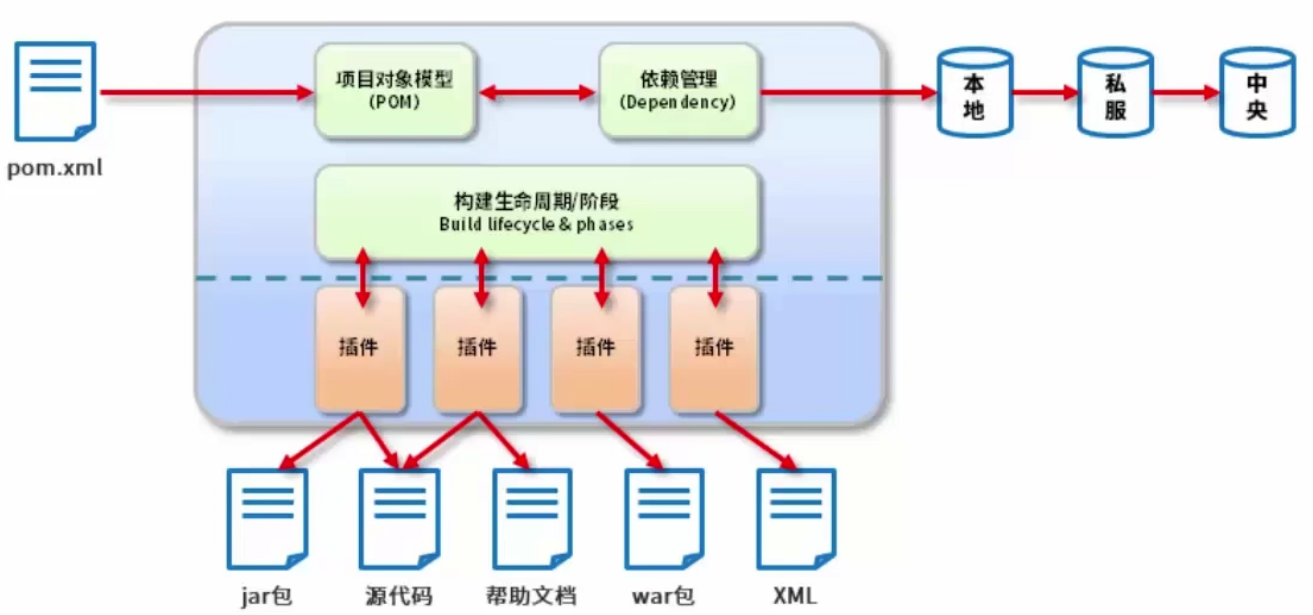
Maven 基础（V3.6.3）

第一章：Maven 简介

1.1 Maven 是什么?

- Maven 的本质是一个项目管理工具，将项目开发和管理过程抽象成一个项目对象模型（POM）。
- `POM（Project Object Model）：项目对象模型。`



1.2 Maven 的作用

- 项目构建：提供标准的、跨平台的自动化项目构建方式。
- 依赖管理：方便、快捷的管理项目依赖的资源（jar 包），避免资源间的版本冲突问题。
- 统一开发结构：提供标准的、统一的项目结构。

第二章：下载和安装

2.1 下载

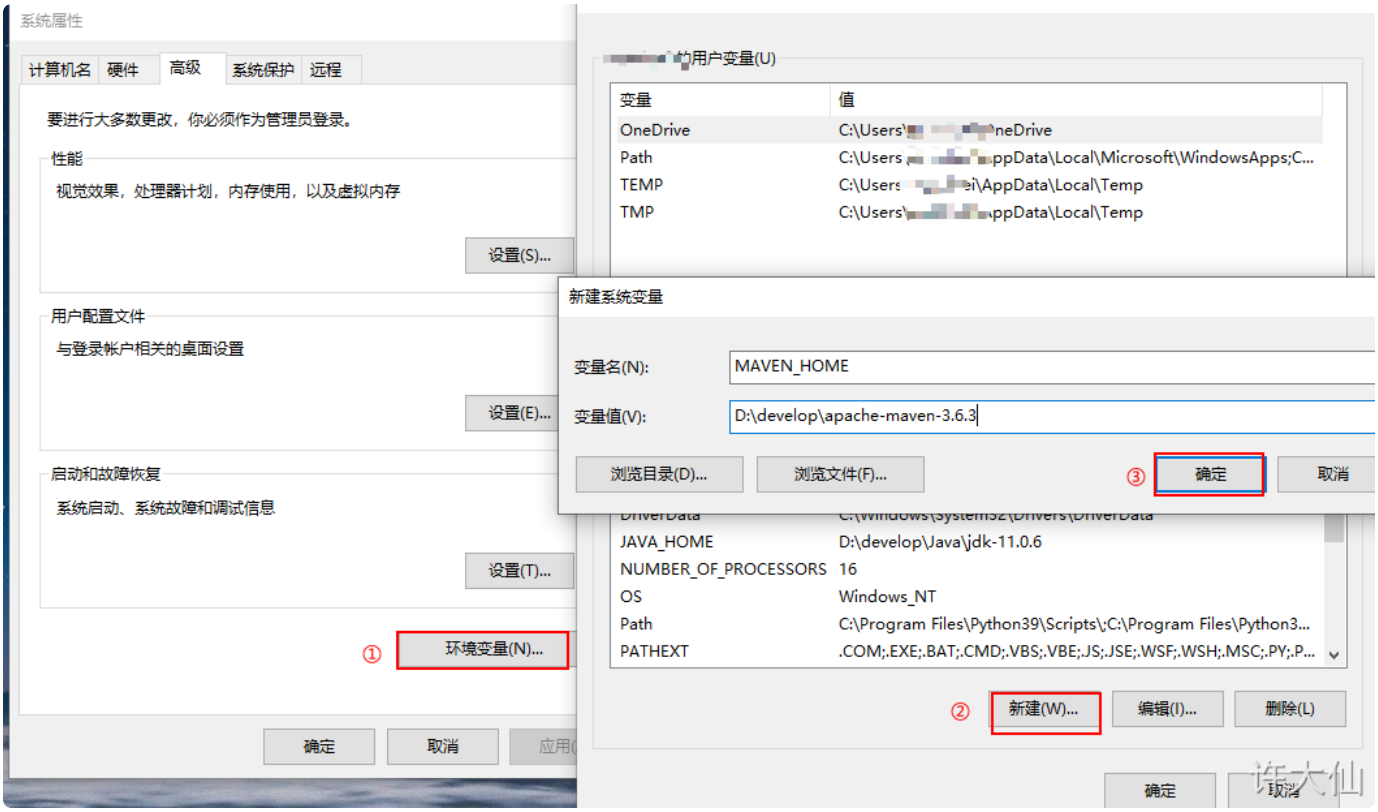
- 官网。

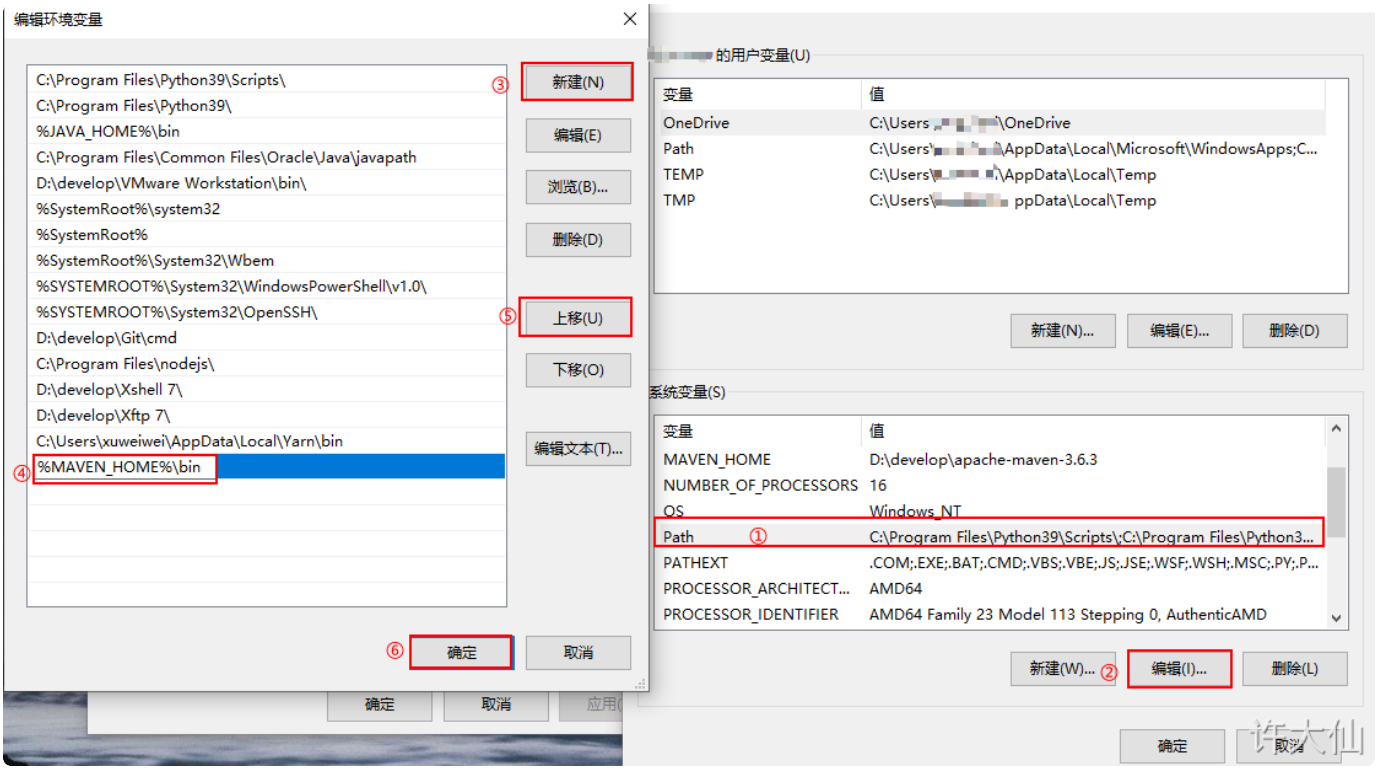
2.2 安装

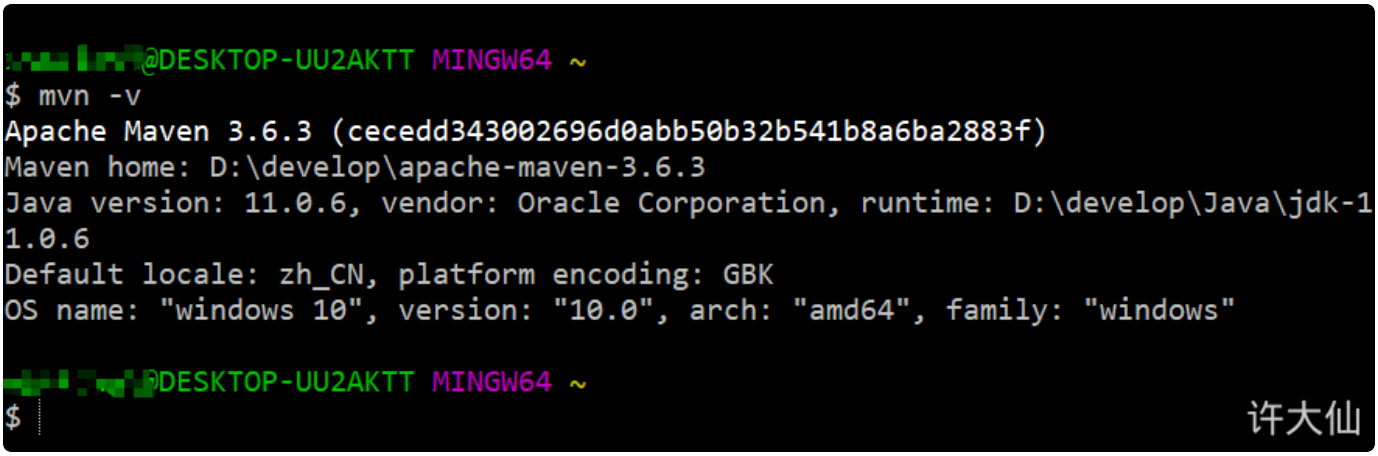
- Maven 属于绿色版软件，解压即安装。

2.3 Maven环境变量的配置

- 依赖 Java ，需要配置 JAVA_HOME（略）。
- 设置 Maven 自身的运行环境，需要配置 MAVEN_HOME 。


系统属性
计算机名 硬件 高级 系统保护 远程
要进行大多数更改，你必须作为管理员登录。
性能
视觉效果，处理器计划，内存使用，以及虚拟内存
设置(S)...
用户配置文件
与登录帐户相关的桌面设置
设置(E)...
启动和故障恢复
系统启动、系统故障和调试信息
设置(T)...
① 环境变量(N)...
确定 取消
的用户变量(U)
变量 值
OneDrive C:\Users\ neDrive
Path C:\Users ppData\Local\Microsoft\WindowsApps;C...
TEMP C:\Users i\AppData\Local\Temp
TMP C:\Users\ ppData\Local\Temp
新建系统变量
变量名(N): MAVEN_HOME
变量值(V): D:\develop\apache-maven-3.6.3
浏览目录(D)... 浏览文件(F)...
③ 确定 取消
JAVA_HOME D:\develop\Java\jdk-11.0.6
NUMBER_OF_PROCESSORS 16
OS Windows_NT
Path C:\Program Files\Python39\Scripts\;C:\Program Files\Python3...
PATHEXT .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.P...
② 新建(W)... 编辑(I)... 删除(L)
确定 取消

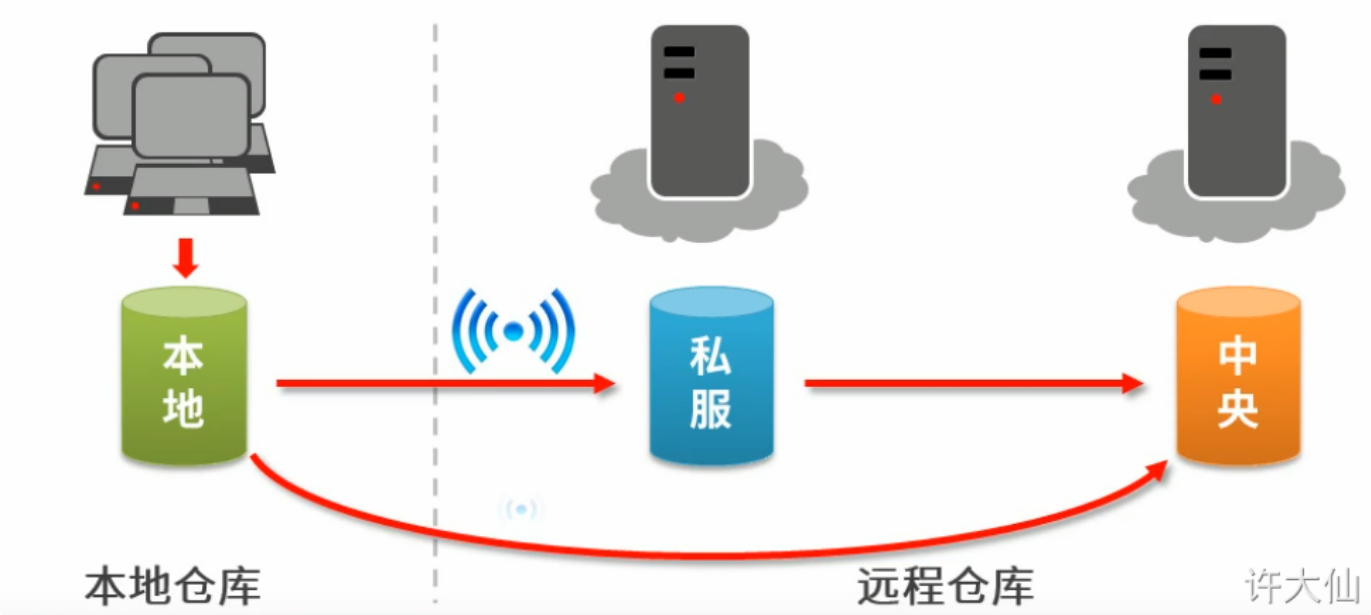

编辑环境变量
C:\Program Files\Python39\Scripts\
C:\Program Files\Python39\
%JAVA_HOME%\bin
C:\Program Files\Common Files\Oracle\Java\javapath
D:\develop\VMware Workstation\bin\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
D:\develop\Git\cmd
C:\Program Files\nodejs\
D:\develop\Xshell 7\
D:\develop\Xftp 7\
C:\Users\xuweiwei\AppData\Local\Yarn\bin
④ %MAVEN_HOME%\bin
③ 新建(N)
编辑(E)
浏览(B)...
删除(D)
⑤ 上移(U)
下移(O)
编辑文本(T)...
⑥ 确定 取消
的用户变量(U)
变量 值
OneDrive C:\Users \OneDrive
Path C:\Users\ \AppData\Local\Microsoft\WindowsApps;C...
TEMP C:\Users\ \AppData\Local\Temp
TMP C:\Users\ ppData\Local\Temp
新建(N)... 编辑(E)... 删除(D)
系统变量(S)
变量 值
MAVEN_HOME D:\develop\apache-maven-3.6.3
NUMBER_OF_PROCESSORS 16
OS Windows_NT
Path ① C:\Program Files\Python39\Scripts\;C:\Program Files\Python3...
PATHEXT .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.PY;.P...
PROCESSOR_ARCHITECT... AMD64
PROCESSOR_IDENTIFIER AMD64 Family 23 Model 113 Stepping 0, AuthenticAMD
新建(W)... ② 编辑(I)... 删除(L)
确定 取消



第三章：Maven 基础概念

3.1 仓库

- 仓库：用于存储资源，包含各种jar包。
- 仓库分类：
  - 本地仓库：自己电脑上存储资源的仓库，连接远程仓库获取资源。
  - 远程仓库：非本机电脑上的仓库，为本地仓库提供资源。
    - 中央仓库：Maven 团队维护，存储所有资源的仓库。
    - 私服：部门/公司范围内存储资源的仓库，从中央仓库获取资源。
- 私服的作用：
  - 保存具有版权的资源，包含购买或自主研发的 jar（中央仓库中的 jar 都是开源的，不能存储具有版权的资源）。
  - 一定范围内共享资源，仅对内部开放，不对外共享。



3.2 坐标

3.2.1 什么是坐标?

- Maven 中的坐标用于描述仓库中资源的位置。
- https://repo1.maven.org/maven2/。

3.2.2 Maven 坐标主要组成

- `groupId`：定义当前 Maven 项目隶属组织名称（通常是域名反写，例如：org.mybatis ）。
- `artifactId`：定义当前 Maven 项目名称（通常是模块名称，例如：CRM、SMS ）。
- `version`：定义当前项目版本号。
- `packaging`：定义该项目的打包方式。
- 示例:

XML

```xml
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

3.2.3 Maven 坐标的作用

- 使用唯一标识，唯一性定位资源位置，通过该标识可以将资源的识别和下载工作，交由机器完成。

3.3 仓库配置

3.3.1 本地仓库配置

- Maven 启动后，会自动保存下载的资源到本地仓库。
- 默认位置（当前目录位置为登录用户名所在目录下的 .m2 文件夹中）：

```xml
<localRepository>${user.home}/.m2/repository</localRepository>
```

- 自定义位置（当前目录位置为 D:\develop\apache-maven-3.6.3\repository 文件夹中）：

```xml
<localRepository>D:\develop\apache-maven-3.6.3\repository</localRepository>
```

3.3.2 远程仓库配置

- Maven 默认连接的仓库位置：

```xml
<repositories>
  <repository>
    <id>central</id>
    <name>Central Repository</name>
    <url>https://repo.maven.apache.org/maven2</url>
    <layout>default</layout>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

3.3.3 镜像仓库配置

- 在 settings.xml 文件中配置阿里云镜像仓库：

```xml
<mirrors>
    <!-- 配置具体的仓库的下载镜像 -->
    <mirror>
        <!-- 此镜像的唯一标识符，用来区分不同的mirror元素 -->
        <id>nexus-aliyun</id>
        <!-- 对那种仓库进行镜像，简单说就是替代哪个仓库 -->
        <mirrorOf>central</mirrorOf>
        <!-- 镜像名称 -->
        <name>Nexus aliyun</name>
        <!-- 镜像URL -->
        <url>http://maven.aliyun.com/nexus/content/groups/public</url>
    </mirror>
</mirrors>
```

3.3.4 全局 setting 和用户 setting 的区别

- 全局 setting 定义了当前计算机中 Maven 的公共配置。
- 用户 setting 定义了当前用户的配置。

第四章：第一个 Maven 项目（手工制作）

4.1 Maven 工程目录结构

- maven 工程名：
  - src：
    - main：
      - java
      - resources
    - test：
      - java
      - resources
  - pom.xml

4.2 pom.xml

- pom.xml

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

    <groupId>com.xudaxian</groupId>
    <artifactId>maven</artifactId>
    <version>1.0</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.20</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

</project>
```

4.3 Maven 项目构建命令

- Maven 构建命令使用 mvn 开头，后面添加功能参数，可以一次执行多个命令，使用空格分隔。

```
mvn compile          #编译
mvn clean            #清理
mvn test             #测试
mvn package          #打包
mvn install          #安装到本地仓库
```

4.4 插件创建工程

- 创建工程：

```shell
mvn archetype:generate -DgroupId={project-packaging} -DartifactId={project-name} -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

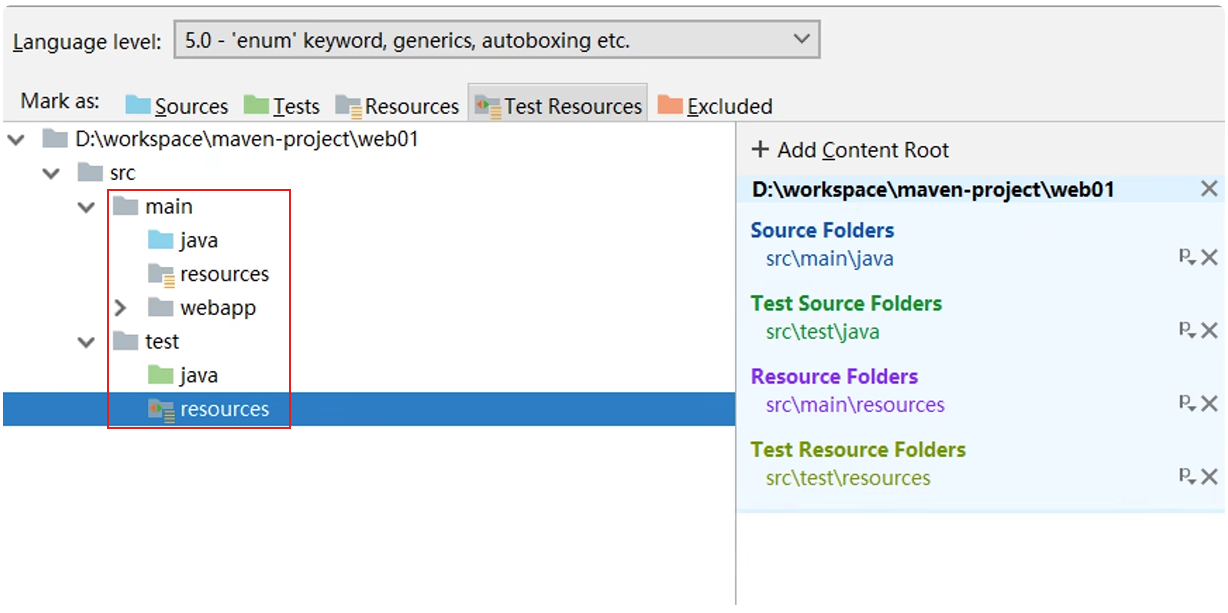
- 创建Java工程：

```shell
mvn archetype:generate -DgroupId={project-packaging} -DartifactId={project-name} -Dversion={project-version} -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

- 创建web工程：

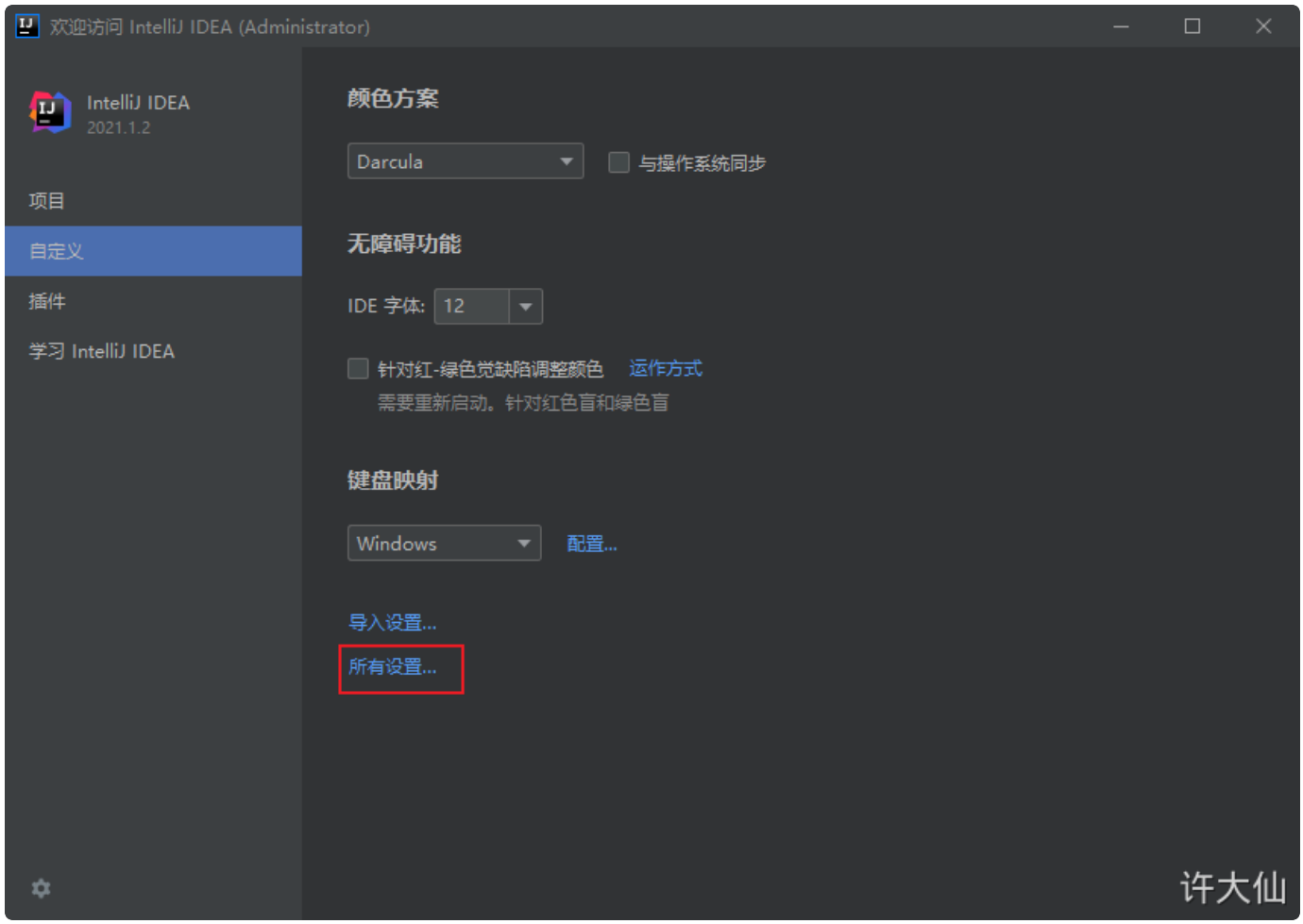
```shell
mvn archetype:generate -DgroupId={project-packaging} -DartifactId={project-name} -Dversion={project-version} -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

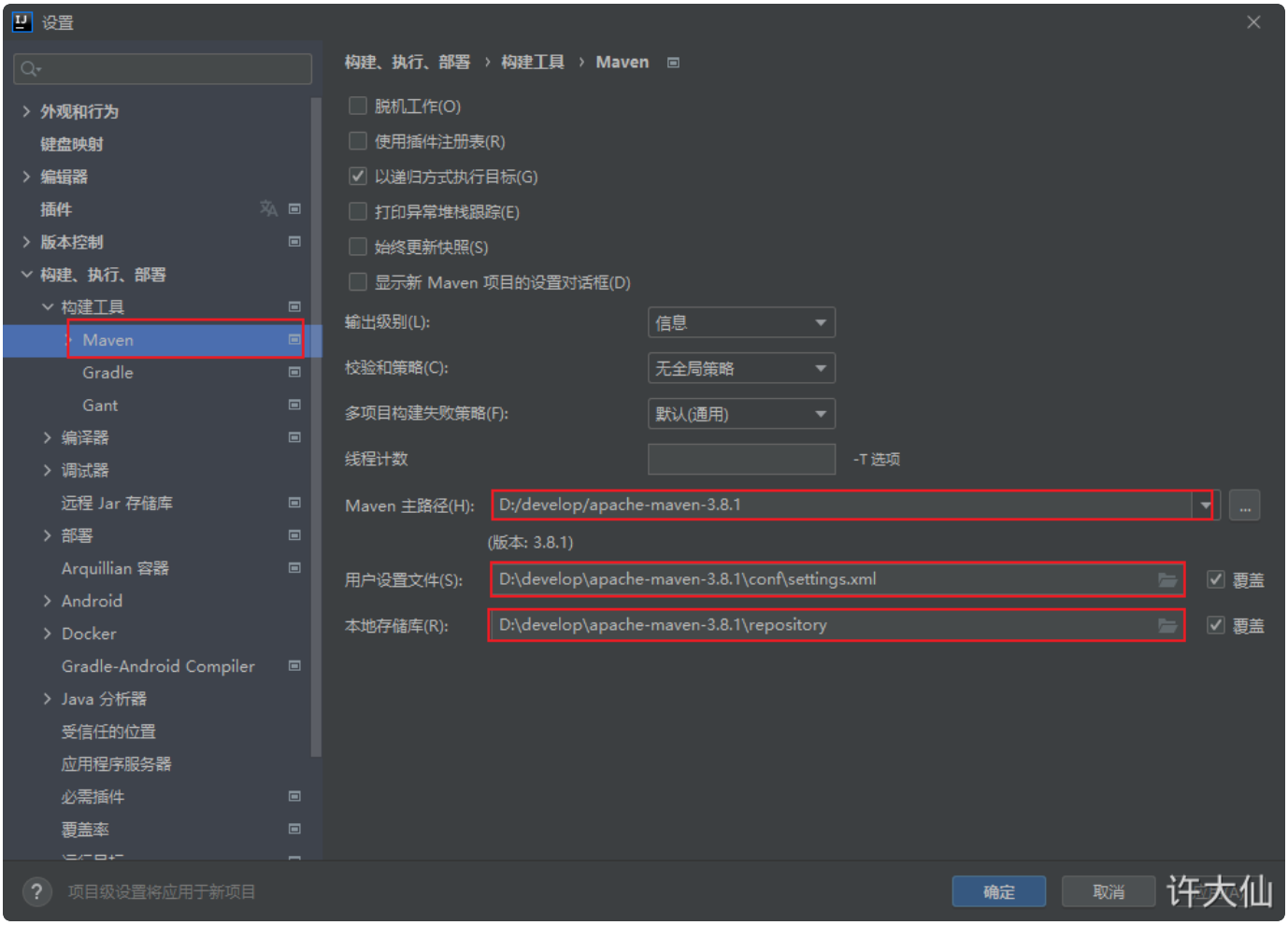
Maven创建的标准项目结构目录



第五章：第一个 Maven 项目（IDEA 生成）

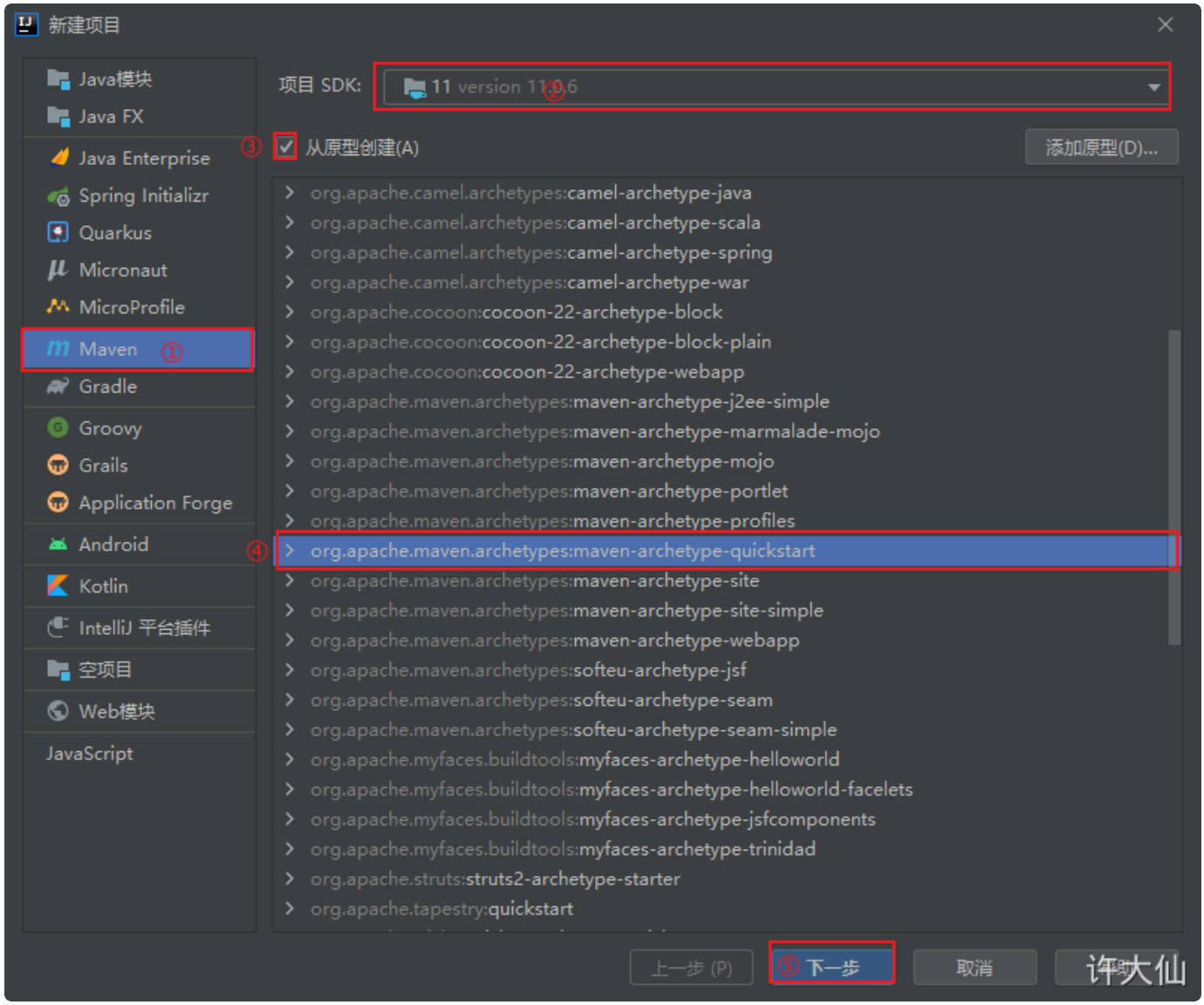
5.1 配置 Maven

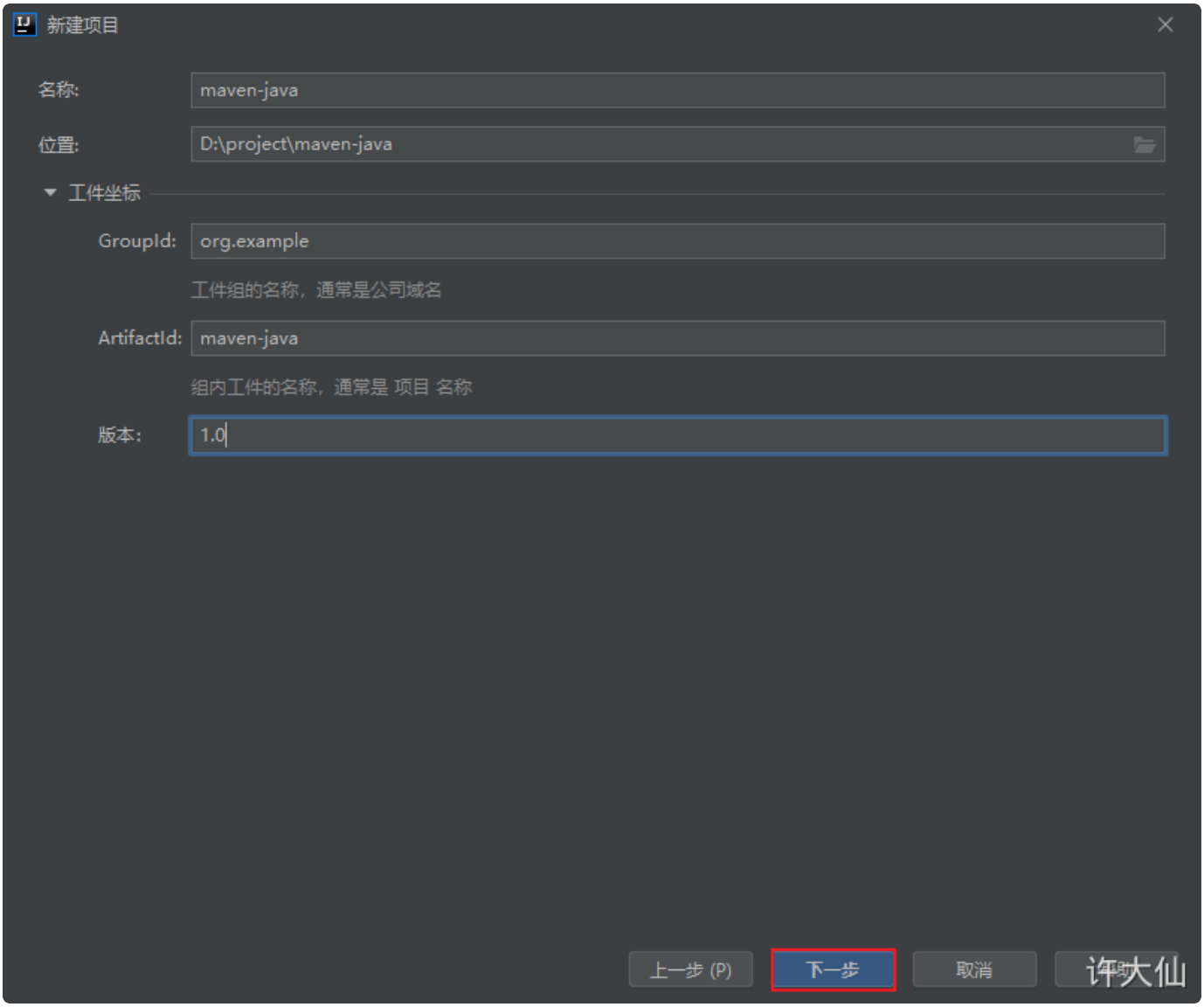




5.2 原型创建 Java 项目

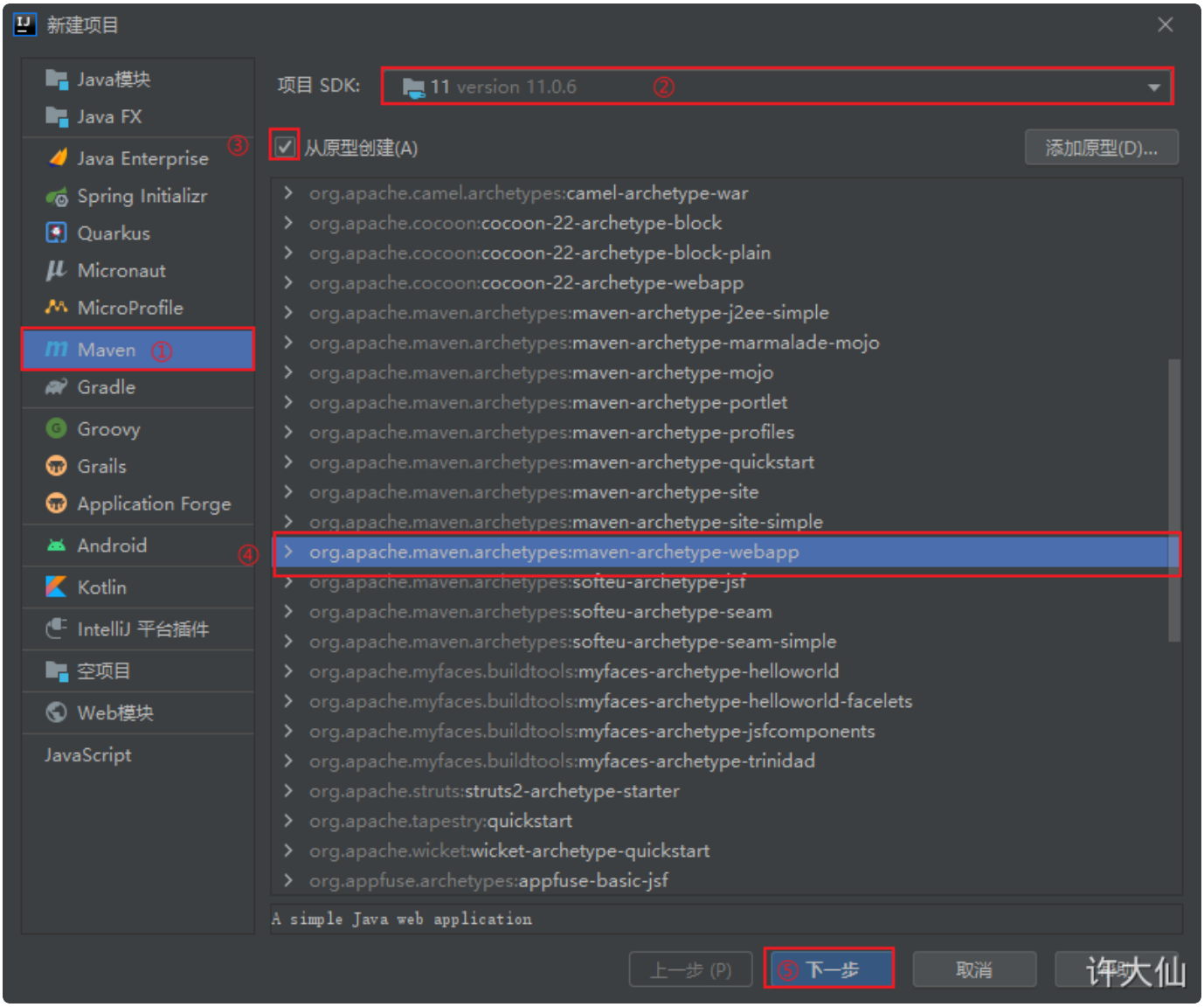

欢迎访问 IntelliJ IDEA (Administrator)
IntelliJ IDEA
2021.1.2
项目
自定义
插件
学习 IntelliJ IDEA
欢迎访问 IntelliJ IDEA (Administrator)
从头创建新项目。
从磁盘或版本控制中打开现有项目。
新建项目
打开
从 VCS 获取
许大仙

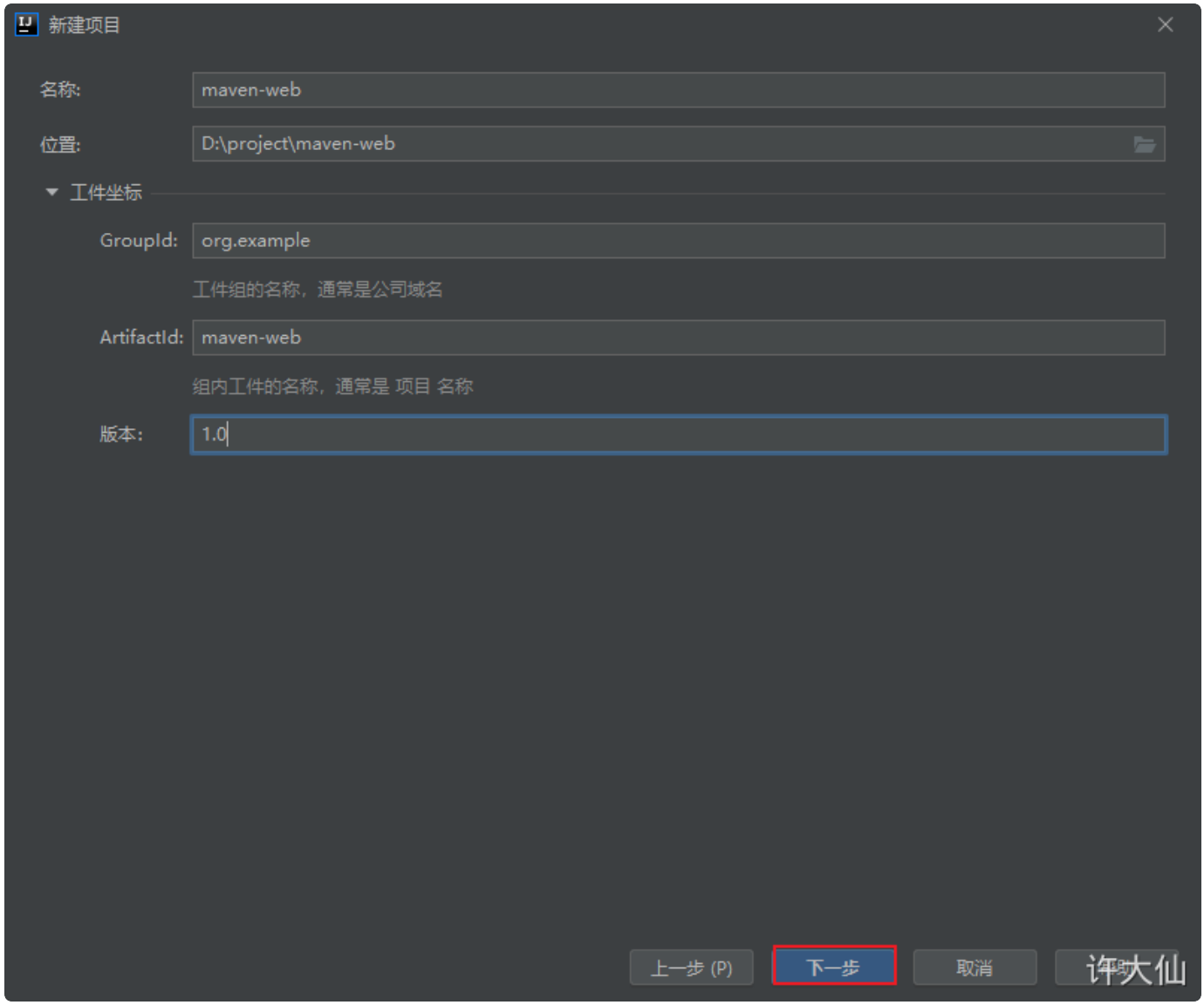

新建项目
Java模块
Java FX
Java Enterprise
Spring Initializr
Quarkus
Micronaut
MicroProfile
Maven
Gradle
Groovy
Grails
Application Forge
Android
Kotlin
IntelliJ 平台插件
空项目
Web模块
JavaScript
项目 SDK:
11 version 11.0.6
从原型创建(A)
添加原型(D)...
org.apache.camel.archetypes:camel-archetype-java
org.apache.camel.archetypes:camel-archetype-scala
org.apache.camel.archetypes:camel-archetype-spring
org.apache.camel.archetypes:camel-archetype-war
org.apache.cocoon:cocoon-22-archetype-block
org.apache.cocoon:cocoon-22-archetype-block-plain
org.apache.cocoon:cocoon-22-archetype-webapp
org.apache.maven.archetypes:maven-archetype-j2ee-simple
org.apache.maven.archetypes:maven-archetype-marmalade-mojo
org.apache.maven.archetypes:maven-archetype-mojo
org.apache.maven.archetypes:maven-archetype-portlet
org.apache.maven.archetypes:maven-archetype-profiles
org.apache.maven.archetypes:maven-archetype-quickstart
org.apache.maven.archetypes:maven-archetype-site
org.apache.maven.archetypes:maven-archetype-site-simple
org.apache.maven.archetypes:maven-archetype-webapp
org.apache.maven.archetypes:softeu-archetype-jsf
org.apache.maven.archetypes:softeu-archetype-seam
org.apache.maven.archetypes:softeu-archetype-seam-simple
org.apache.myfaces.buildtools:myfaces-archetype-helloworld
org.apache.myfaces.buildtools:myfaces-archetype-helloworld-facelets
org.apache.myfaces.buildtools:myfaces-archetype-jsfcomponents
org.apache.myfaces.buildtools:myfaces-archetype-trinidad
org.apache.struts:struts2-archetype-starter
org.apache.tapestry:quickstart
① ② ③ ④ ⑤
上一步 (P)
下一步
取消
许大仙



5.3 原型创建 web 项目


欢迎访问 IntelliJ IDEA (Administrator)
IntelliJ IDEA
2021.1.2
项目
自定义
插件
学习 IntelliJ IDEA
欢迎访问 IntelliJ IDEA (Administrator)
从头创建新项目。
从磁盘或版本控制中打开现有项目。
新建项目
打开
从 VCS 获取
许大仙


新建项目
Java模块
Java FX
Java Enterprise
Spring Initializr
Quarkus
Micronaut
MicroProfile
Maven
Gradle
Groovy
Grails
Application Forge
Android
Kotlin
IntelliJ 平台插件
空项目
Web模块
JavaScript
项目 SDK:
11 version 11.0.6
从原型创建(A)
添加原型(D)...
org.apache.camel.archetypes:camel-archetype-war
org.apache.cocoon:cocoon-22-archetype-block
org.apache.cocoon:cocoon-22-archetype-block-plain
org.apache.cocoon:cocoon-22-archetype-webapp
org.apache.maven.archetypes:maven-archetype-j2ee-simple
org.apache.maven.archetypes:maven-archetype-marmalade-mojo
org.apache.maven.archetypes:maven-archetype-mojo
org.apache.maven.archetypes:maven-archetype-portlet
org.apache.maven.archetypes:maven-archetype-profiles
org.apache.maven.archetypes:maven-archetype-quickstart
org.apache.maven.archetypes:maven-archetype-site
org.apache.maven.archetypes:maven-archetype-site-simple
org.apache.maven.archetypes:maven-archetype-webapp
org.apache.maven.archetypes:softeu-archetype-jsf
org.apache.maven.archetypes:softeu-archetype-seam
org.apache.maven.archetypes:softeu-archetype-seam-simple
org.apache.myfaces.buildtools:myfaces-archetype-helloworld
org.apache.myfaces.buildtools:myfaces-archetype-helloworld-facelets
org.apache.myfaces.buildtools:myfaces-archetype-jsfcomponents
org.apache.myfaces.buildtools:myfaces-archetype-trinidad
org.apache.struts:struts2-archetype-starter
org.apache.tapestry:quickstart
org.apache.wicket:wicket-archetype-quickstart
org.appfuse.archetypes:appfuse-basic-jsf
A simple Java web application
上一步 (P)
下一步
取消
①
②
③
④
⑤

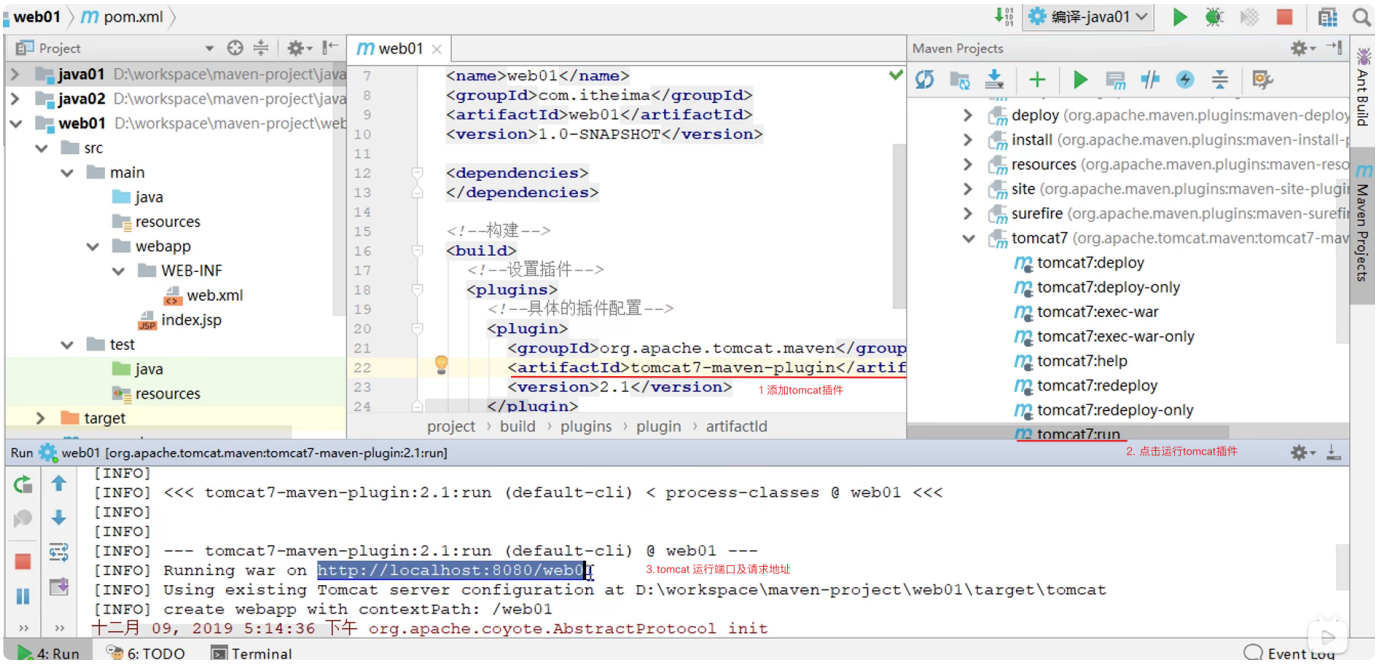


5.4 插件

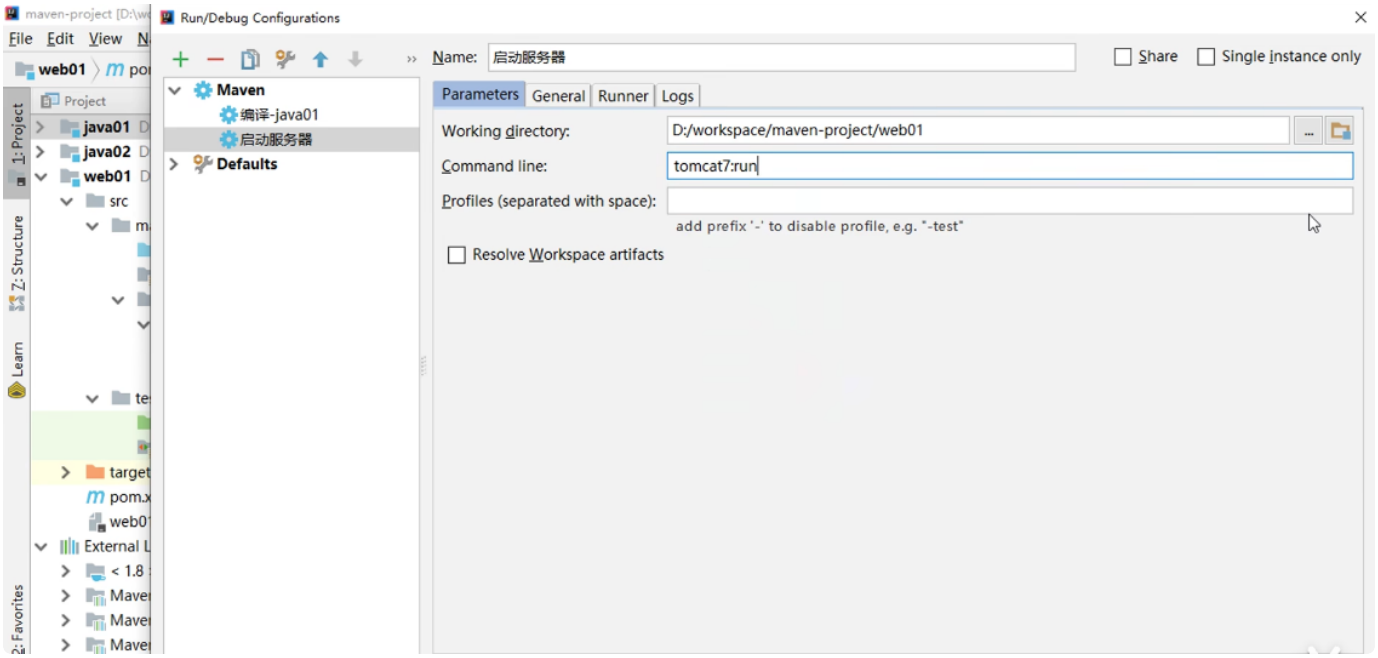
- Tomcat 7 插件：

```xml
<!-- 构建-->
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.tomcat.maven</groupId>
      <artifactId>tomcat7-maven-plugin</artifactId>
      <version>2.1</version>
      <configuration>
        <port>80</port>
        <path>/</path>
      </configuration>
    </plugin>
  </plugins>
</build>
```

启动运行tomcat项目



也可以使用启动项配置进行tomcat应用启动



第六章：依赖管理

6.1 依赖配置

- 依赖是指当前项目运行所需要的 jar 包，一个项目可以设置多个依赖。
- 格式：

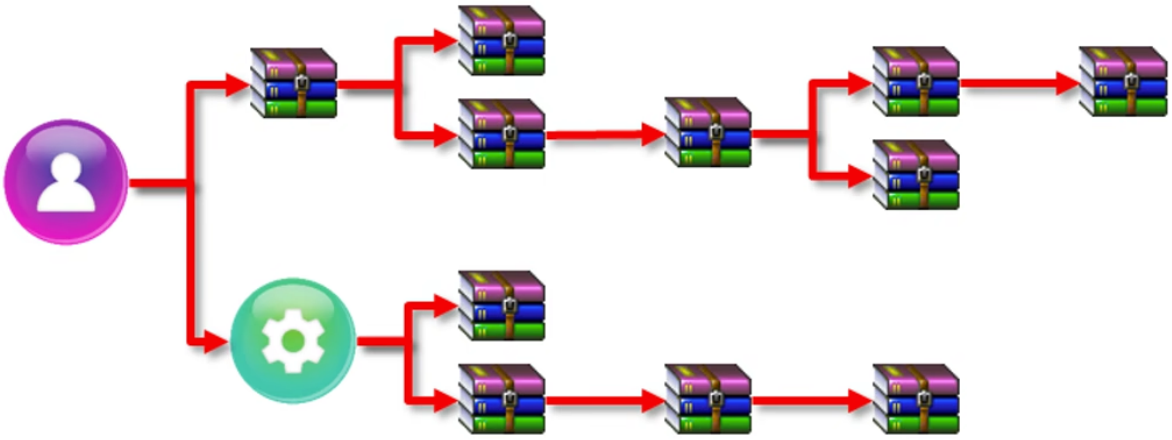
```xml
<!-- 设置当前项目所依赖的所有jar包 -->
<dependencies>
    <!-- 设置具体的依赖 -->
    <dependency>
        <!-- 依赖的组织id -->
        <groupId>xxx</groupId>
        <!-- 依赖的所属项目id -->
        <artifactId>xxx</artifactId>
        <!-- 依赖的版本号 -->
        <version>xxx</version>
    </dependency>
</dependencies>
```

- 示例：

```xml
<dependencies>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.20</version>
        <scope>provided</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

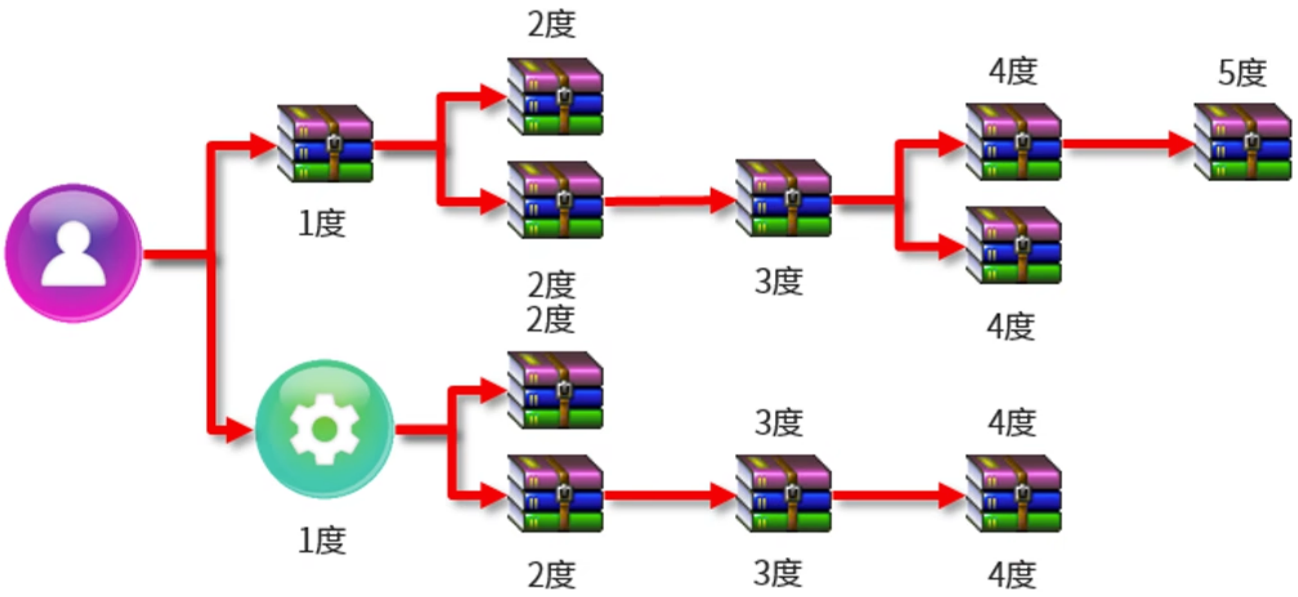
6.2 依赖传递

- 依赖具有传递性：
  - 直接依赖：在当前项目中通过依赖配置建立的资源关系。
  - 间接依赖：被依赖的资源如果依赖其他资源，那么当前项目会间接依赖其他资源。



6.3 依赖传递冲突问题

- 路径优先：当依赖中出现相同的资源时，层级越深，优先级越低，层级越浅，优先级越高。
- 声明优先：当资源在相同层级被依赖的时候，配置顺序靠前的覆盖配置顺序靠后的。
- 特殊优先：当同级配置了相同资源的不同版本，后配置的覆盖先配置的。



6.4 可选依赖（不透明）

- 可选依赖指的是 `对外` 隐藏当前所依赖的资源。

```xml
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <optional>true</optional>
</dependency>
```

6.5 排除依赖（不需要）

- 排除依赖指的是主动断开依赖的资源，被排除的资源无需指定版本。

```xml
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <exclusions>
        <exclusion>
            <groupId>org.hamcrest</groupId>
            <artifactId>hamcrest-core</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

6.6 依赖范围

- **所依赖的 jar 默认情况下可以在任何地方使用，可以通过 scope 标签设定其作用范围。**
- 作用范围：
  - 主程序范围有效（main 文件夹范围内）。
  - 测试程序范围有效（test 文件夹范围内）。
  - 是否参与打包（package 指令范围内）。

scope	主代码	测试代码	打包	范例
compile（默认）	Y	Y	Y	log4j
test		Y		junit
provided	Y	Y		servlet-api
runtime			Y	jdbc

6.7 依赖范围的传递性

- 带有依赖范围的资源在进行传递的时候，作用范围将受到影响。

	compile	test	provided	runtime
compile	compile	test	provided	runtime
test				

provided				
runtime	runtime	test	provided	runtime

第七章：生命周期和插件

7.1 构建生命周期

7.1.1 项目构建生命周期

- Maven 对项目构建的生命周期划分为3套：
  - clean：清理工作。
  - default：核心工作，例如：编译、测试、打包、部署等。
  - site：产生报告，发布站点等。

7.1.2 clean 生命周期

- pre-clean：执行一些需要在 clean 之前完成的工作。
- clean：移除所有上一次构建生成的文件。
- post-clean：执行一些需要在 clean 之后立刻完成的工作。

7.13 default 生命周期

- validate （校验）：校验项目是否正确并且所有必要的信息可以完成项目的构建过程。
- initialize （初始化）：初始化构建状态，比如设置属性值。
- generate sources （生成源代码）：生成包含在编译阶段中的任何源代码。
- process-sources（处理源代码）：处理源代码，比如说，过滤任意值。
- generate-resources（生成资源文件）：生成将会包含在项目包中的资源文件。
- process-resources （处理资源文件）：复制和处理资源到目标目录，为打包阶段最好准备。
- compile（编译）：编译项目的源代码。
- process-classes（处理类文件）：处理编译生成的文件，比如说对 Java class 文件做字节码改善优化。
- generate-test-sources（生成测试源代码）：生成包含在编译阶段中的任何测试源代码。

- process-test-sources（处理测试源代码）：处理测试源代码，比如说，过滤任意值。
- generate-test-resources（生成测试资源文件）：为测试创建资源文件。
- process-test-resources（处理测试资源文件）：复制和处理测试资源到目标目录。
- test-compile（编译测试源码）：编译测试源代码到测试目标目录。
- process-test-classes（处理测试类文件）：处理测试源码编译生成的文件。
- test（测试）：使用合适的单元测试框架运行测试（Juint 是其中之一）。
- prepare-package（准备打包）：在实际打包之前，执行任何的必要的操作为打包做准备。
- package（打包）：将编译后的代码打包成可分发格式的文件，比如 JAR、WAR 或者 EAR 文件。
- pre-integration-test（集成测试前）：在执行集成测试前进行必要的动作。比如说，搭建需要的环境。
- integration-test（集成测试）：处理和部署项目到可以运行集成测试环境中。
- post-integration-test（集成测试后）：在执行集成测试完成后进行必要的动作。比如说，清理集成测试环境。
- verify （验证）：运行任意的检查来验证项目包有效且达到质量标准。
- install（安装）：安装项目包到本地仓库，这样项目包可以用作其他本地项目的依赖。
- deploy（部署）：将最终的项目包复制到远程仓库中与其他开发者和项目共享。

7.1.4 site 构建生命周期

- pre-site：执行一些需要在生成站点文档之前完成的工作。
- site：生成项目的站点文档。
- post-site：执行一些需要在生成站点文档之后完成的工作，并且为部署做准备。
- site-deploy：将生成的站点文档部署到特定的服务器上。

7.2 插件

maven官网插件：https://maven.apache.org/plugins/index.html

- 插件和生命周期内的阶段绑定，在执行到对应生命周期时执行对应的插件功能。
- 默认 maven 在各个生命周期上绑定有预设的功能。
- 通过插件可以自定义其他功能。

XML

```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-source-plugin</artifactId>
            <version>2.2.1</version>
            <executions>
                <execution>
                    <!-- 执行目标 -->
                    <goals>
                        <goal>jar</goal>
                    </goals>
                    <!-- 构建生命周期的阶段 -->
                    <phase>generate-test-resources</phase>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```