Maven

1. 介绍

Maven是一个**项目管理工具**。主要是有以下的两个功能

- 项目构建

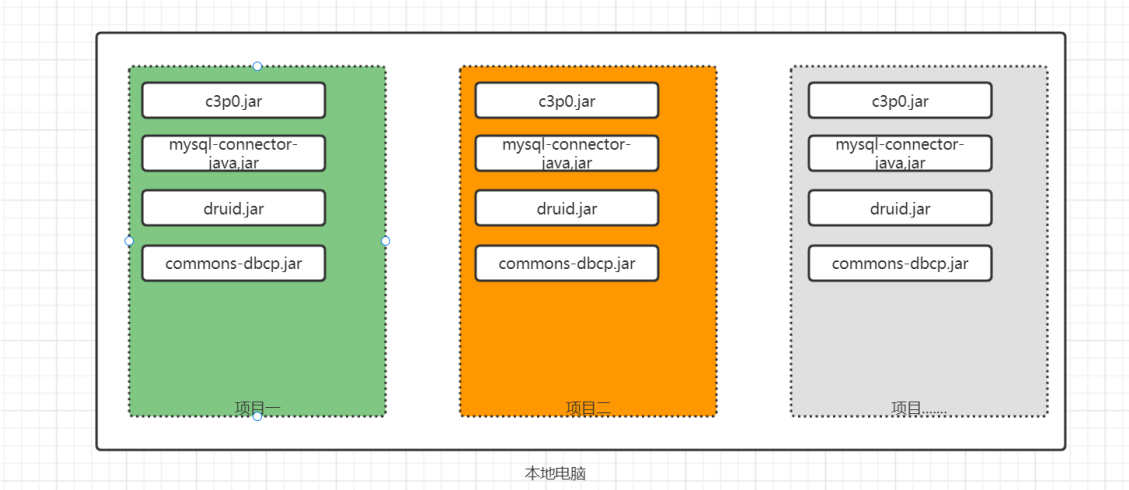
  > 项目构建是指把我们自己写的项目进行打包、编译、安装等等。

- 依赖管理

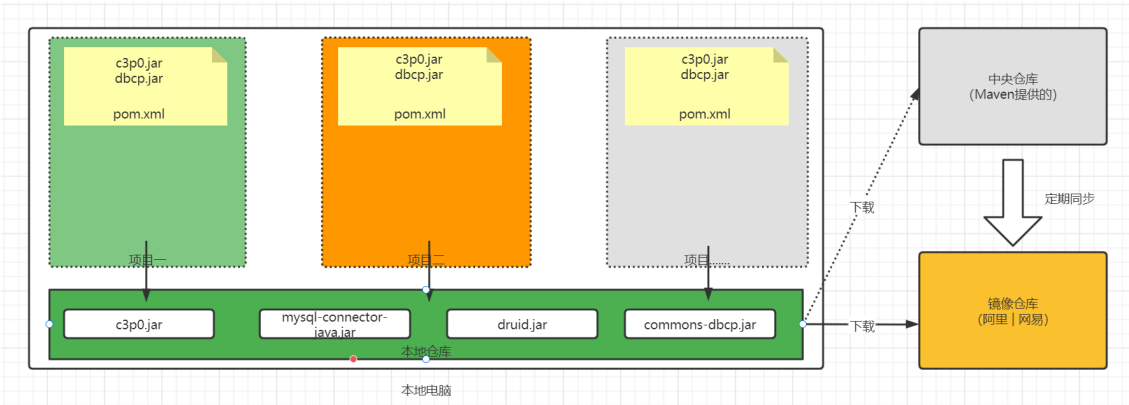
  > 依赖管理其实就是管理项目中的依赖，就是管理项目中导入的jar包。

2. 运行流程

- 传统Java项目导包方式



- maven项目工作流程



本地仓库：本质上就是本地的一个文件夹，这个文件夹中专门用来保存jar包

中央仓库：这个是Maven官方提供的一个仓库，服务器在国外，国内下载速度比较慢

镜像仓库：国内有些大公司或者是机构会对中央仓库进行定期同步，这种仓库叫做镜像仓库，也是国内的开发者大多数的选择

3. 安装

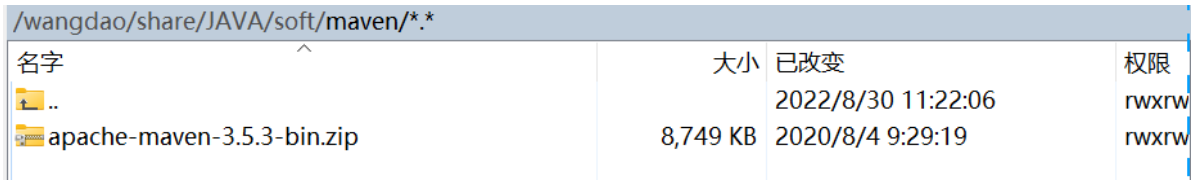
官网

maven是一个使用Java语言开发的工具，所以Maven的运行依赖与本地的jdk，所以maven的版本需要和JDK的版本保持一致。后续我们需要在IDEA中使用Maven，所以Maven的版本也尽量和IDEA的版本保持匹配。

- IDEA：2018.3.6
- JDK：1.8
- MAVEN：3.5.3

3.1 下载

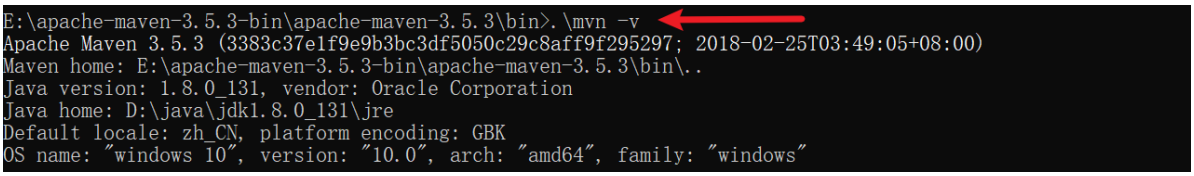
下载地址



3.2 解压

解压到一个不带有中文，不带有特殊字符的路径即可。

验证Maven解压出来的文件是否有问题：



> 如果出现以上的一些日志，就说明没有问题

3.3 配置

3.3.1 配置环境变量

3.3.2 配置Maven

其实以下的三个配置都是去修改Maven解压路径下的conf文件夹中的`settings.xml`

- 配置本地仓库

```
<settings>
    ...
    <localRepository>D:\maven\repo</localRepository>
    ...
</settings>
```

- 配置镜像仓库

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

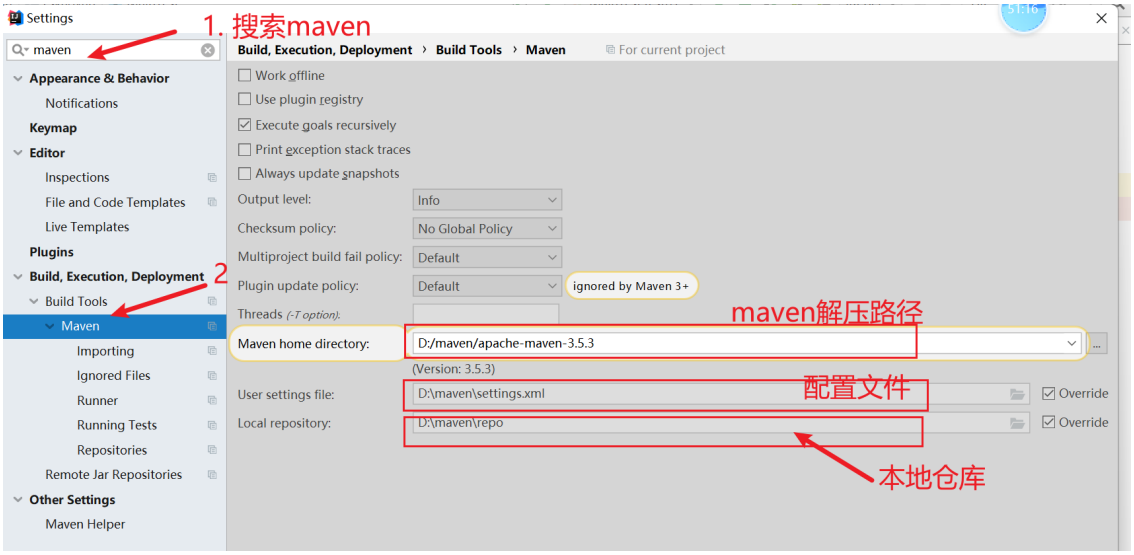
- 配置JDK的编译版本

```
<settings>
    ...
    <profiles>
        <profile>
            <id>jdk-1.8</id>
            <activation>
                <activeByDefault>true</activeByDefault>
                <jdk>1.8</jdk>
            </activation>
            <properties>
                <maven.compiler.source>1.8</maven.compiler.source>
                <maven.compiler.target>1.8</maven.compiler.target>

<maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>
            </properties>
        </profile>

    </profiles>
    ...

</settings>
```

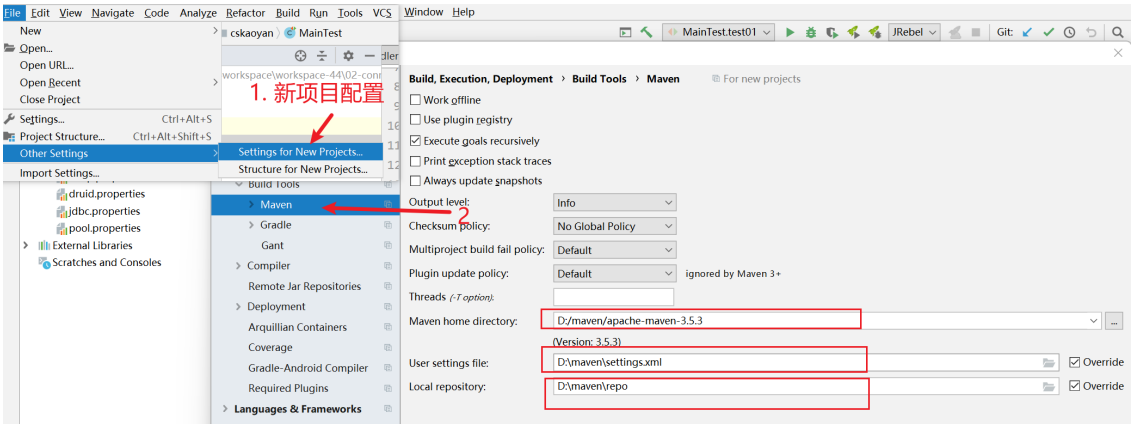
3.3.3 配置IDEA

因为后续需要在IDEA中使用Maven，所以在IDEA也需要指定maven的设置。

- 配置当前项目



- 配置新项目



4. 初识Maven

4.1 maven工程结构

- 项目名字
  - src
    - main
      - java

        > 这个路径是源代码路径，这个文件夹下面都是放的`.Java`文件以及他们的包

      - resources

        > 配置文件存放的路径

    - test
      - java

        > 测试代码存放的路径

      - resources

        > 测试代码中需要用到的配置文件存放的路径

  - pom.xml

    maven项目的管理文件，在这个文件中来管理这个项目所导入的依赖

在每一个pom.xml文件中都会声明这个工程的坐标

- target

  这个是编译生成的target文件夹，这个文件夹中放的是 字节码文件以及配置文件

4.2 Maven工程的坐标

每一个Maven工程的坐标都是独一无二的。

- groupId

  公司名或者是组织名（一般是域名反转）

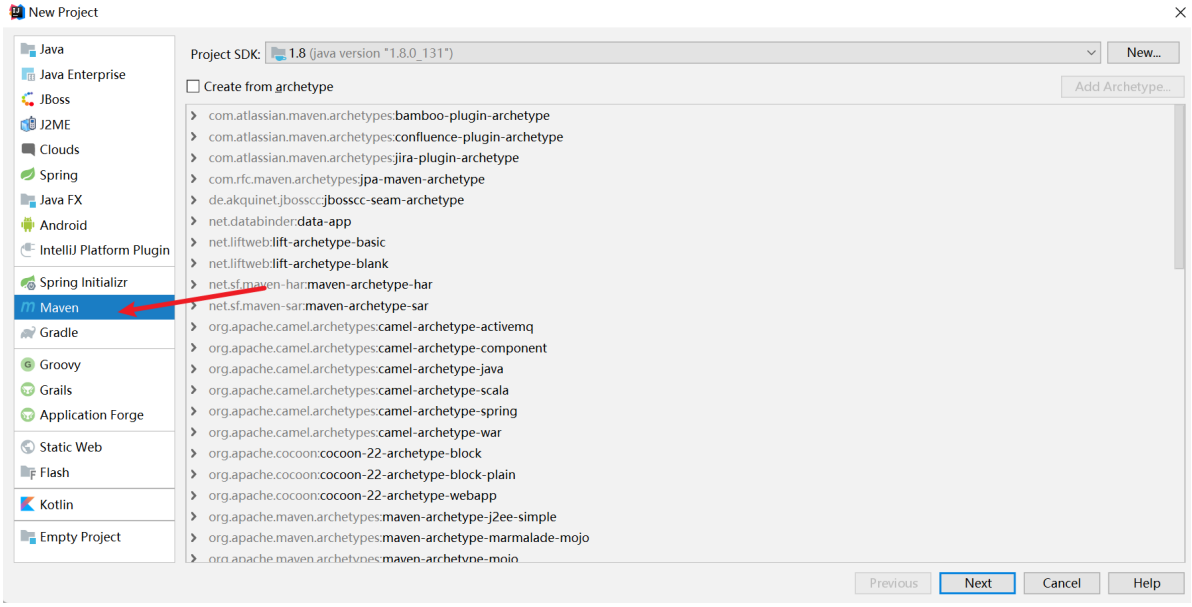
- artifactId

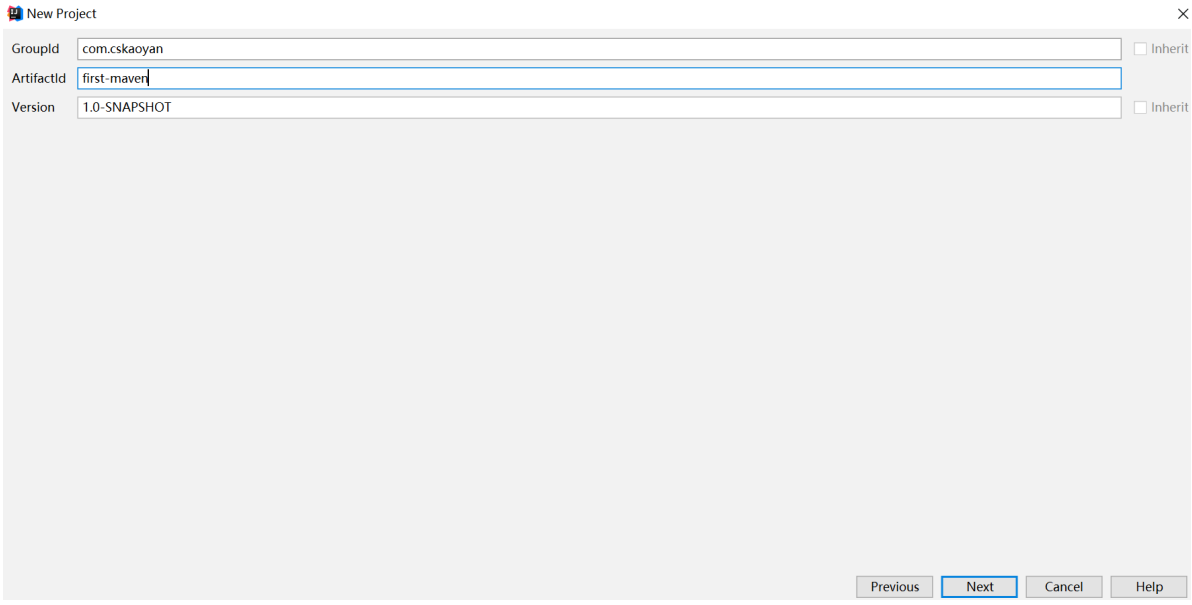
  工程名

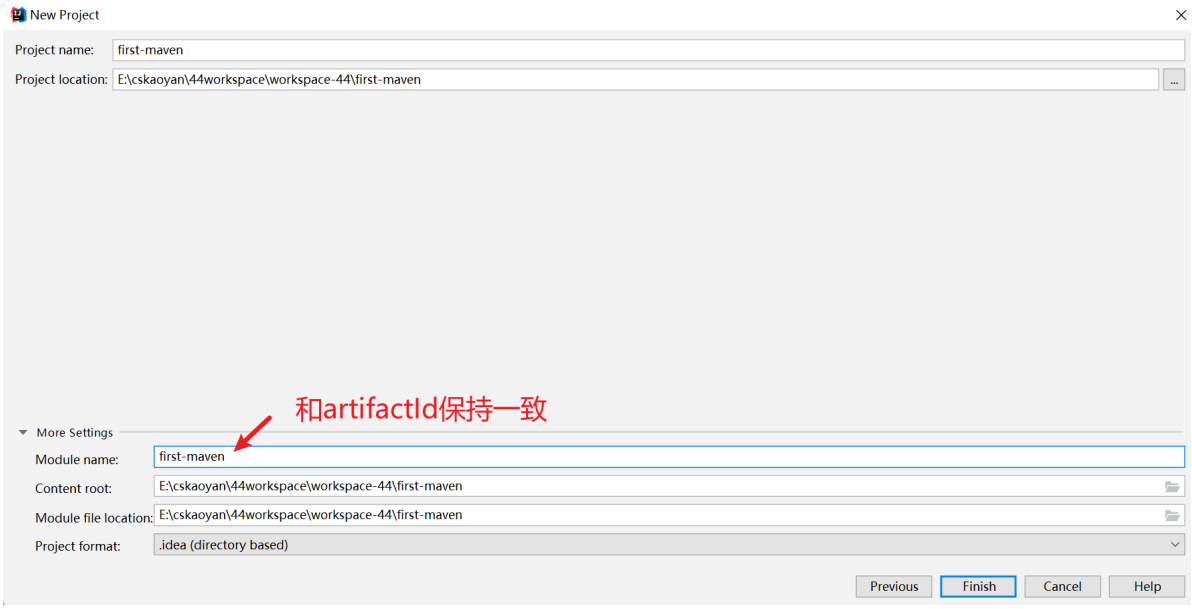
- version

  版本号（例如 1.0.0 | 1.0 | 1.0.0-RELEASE(发行版，表示是正式的版本，稳定) | 0.1.15-SNAPSHOT(快照版，一般认为不稳定)）

4.3 新建Maven工程





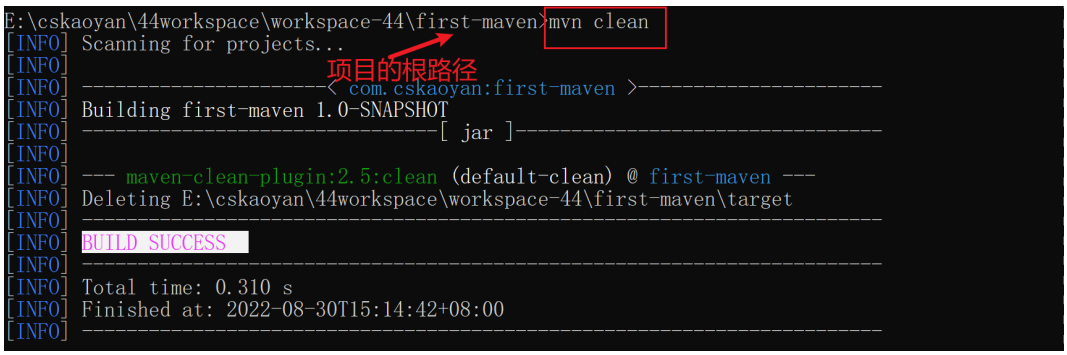


5. 项目构建

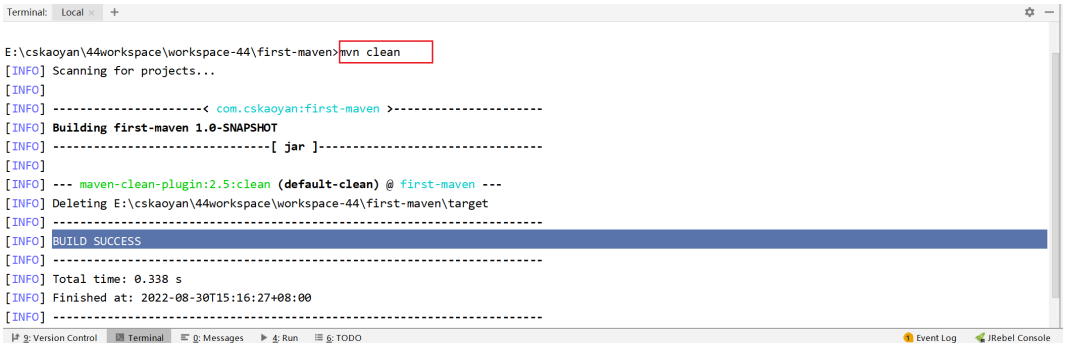
5.1命令的执行方式

在Maven中，每一个命令都有三种运行的方式：

1. 在cmd中运行

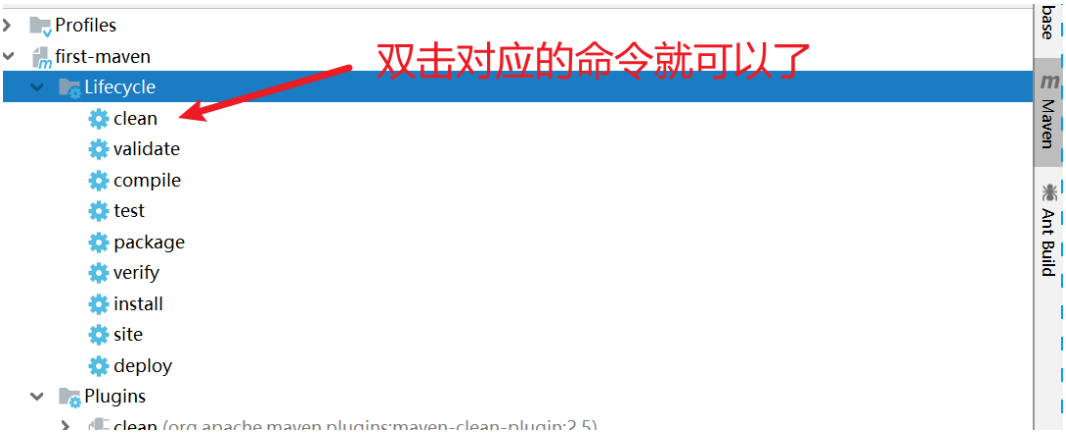


2. 在idea中提供的命令行来运行



假如被告知找不到mvn命令，那么需要把idea使用管理员权限打开

3. 使用idea的maven插件来运行

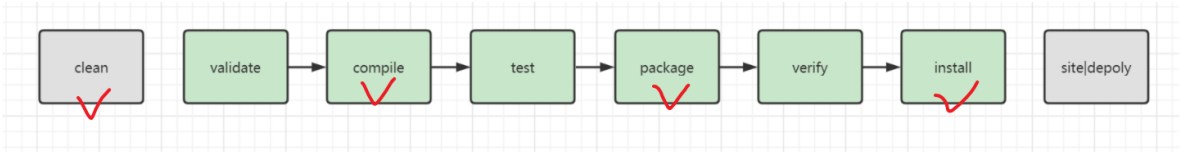


5.2 命令的工作

项目构建依赖Maven给我们提供的一系列命令，而命令的运行依赖于maven的一些插件，这些插件其实就是一个一个的jar包。

5.3 命令的执行

项目构建是指Maven可以帮助我们编译、测试、打包、安装项目



- clean

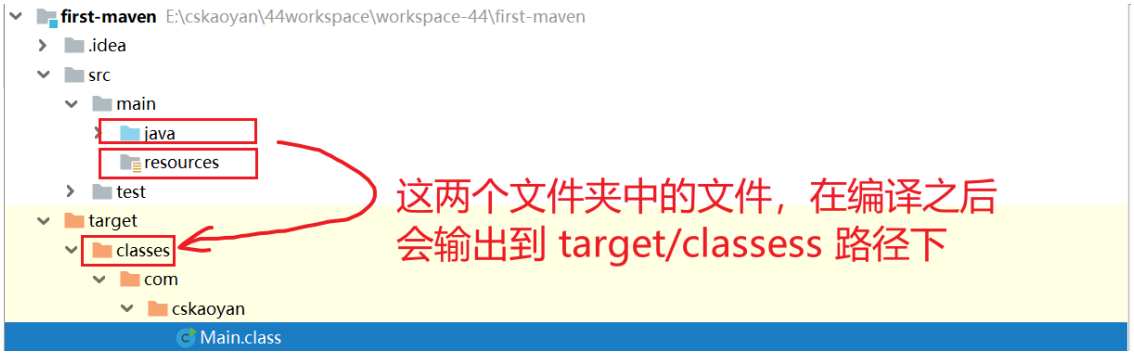
  删除编译生成的target文件夹

- validate

  在编译之前验证项目中的文件是否有损坏、权限等等，做编译的准备工作

- compile

  编译项目，生成target文件夹



- test

  测试项目，其实就是运行这个项目中所有的测试类以及测试方法

  > 说明：只会运行src/test/java路径下的所有的测试方法

- package

  实际上是对我们的项目进行打包，把项目打包成一个jar包或者是一个war(Tomcat)包。


```
<!-- 当前这个Maven工程的坐标 -->
<groupId>com.cskaoyan</groupId>
<artifactId>03-first-maven</artifactId>
<version>1.0-SNAPSHOT</version>

<!--jar 默认值 ，war是打war包的时候用到的，pom目前不要使用-->
<!-- jar | war | pom-->
<packaging>jar</packaging>


<!-- 声明需要导入jar包的坐标-->
<dependencies>

    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
    <dependency>
```

- verify

  验证通过package命令打出的jar包或者是war包

- install

  install就是把package生成的jar包复制到本地仓库中

  > **如何从本地仓库中找到对应的jar包呢?**
  >
  > **通过坐标去本地仓库中找，一级一级的找**

- site/deploy

  这两个命令是和部署相关的，一般在公司中，不使用maven来部署项目，一般使用Jenkins

6. 依赖管理

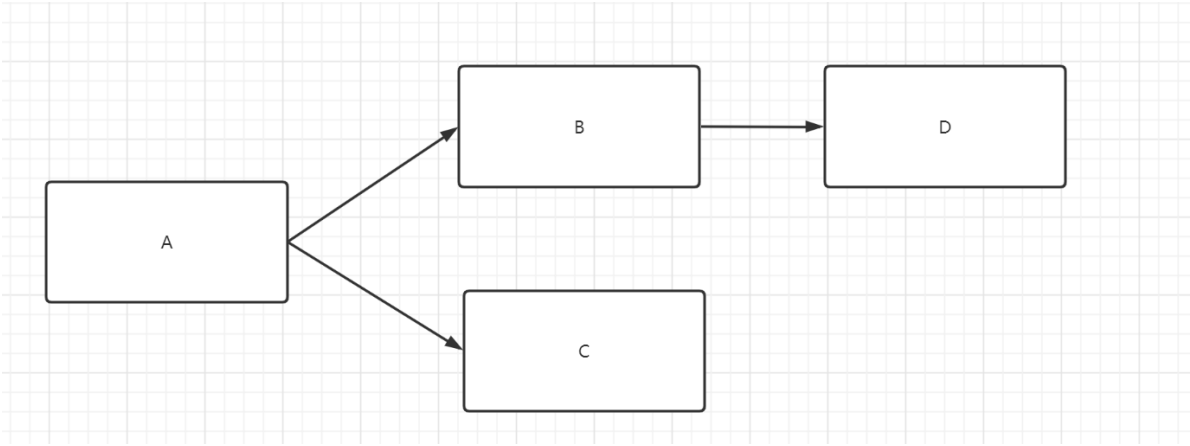
maven可以帮助我们管理项目的依赖，其实就是管理项目导入的jar包。

6.1 如何导包

> 1. 去Maven的官方网站搜索对应的依赖包，并且复制依赖包的坐标
> 2. 把对应的坐标粘贴到pom.xml中的 标签下就可以了

6.2 依赖传递

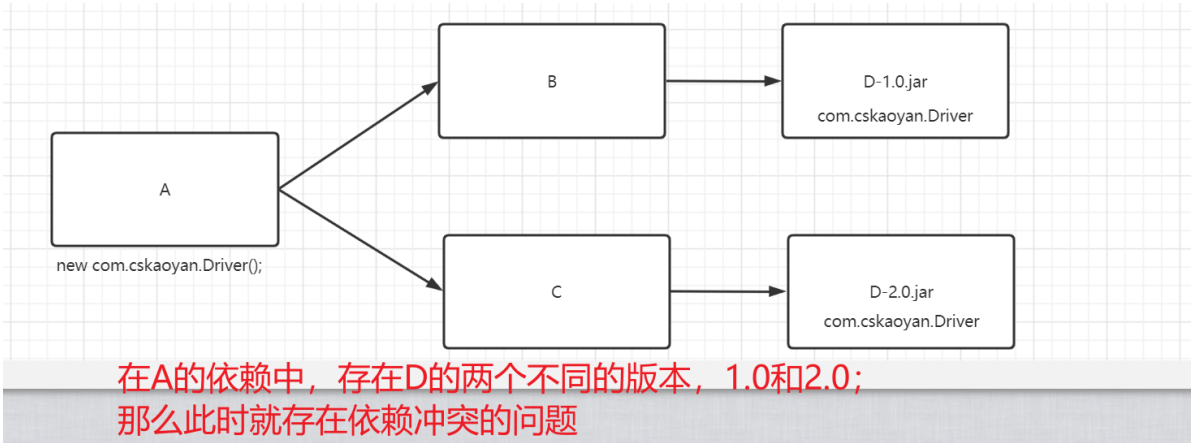
依赖具有传递性。



> 根据依赖的传递性，A的依赖中有D这个依赖

6.3 依赖冲突

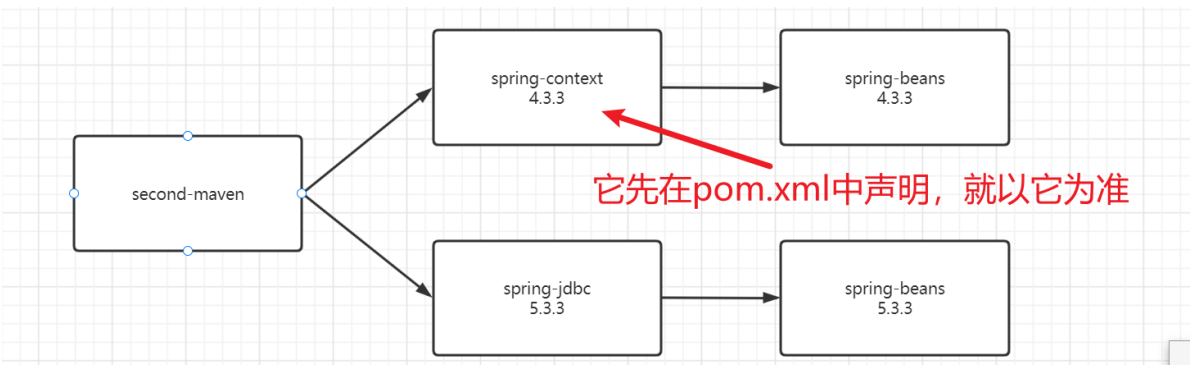
依赖冲突是指在同一个项目中，导入了同一个jar包的不同版本，就会存在冲突的问题。



声明优先原则

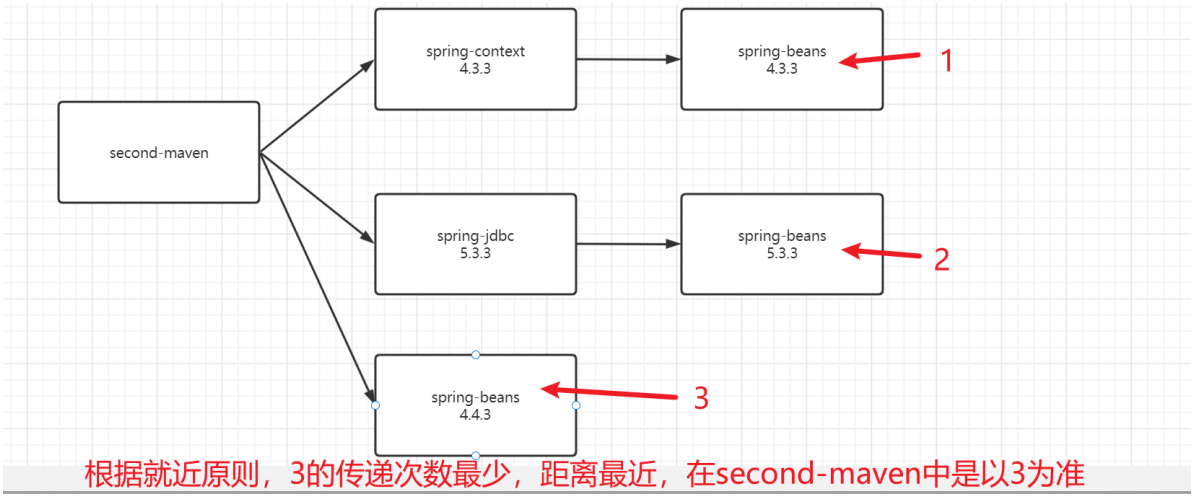
谁先声明，以谁为准。

谁先在pom.xml中声明，就以谁为准。



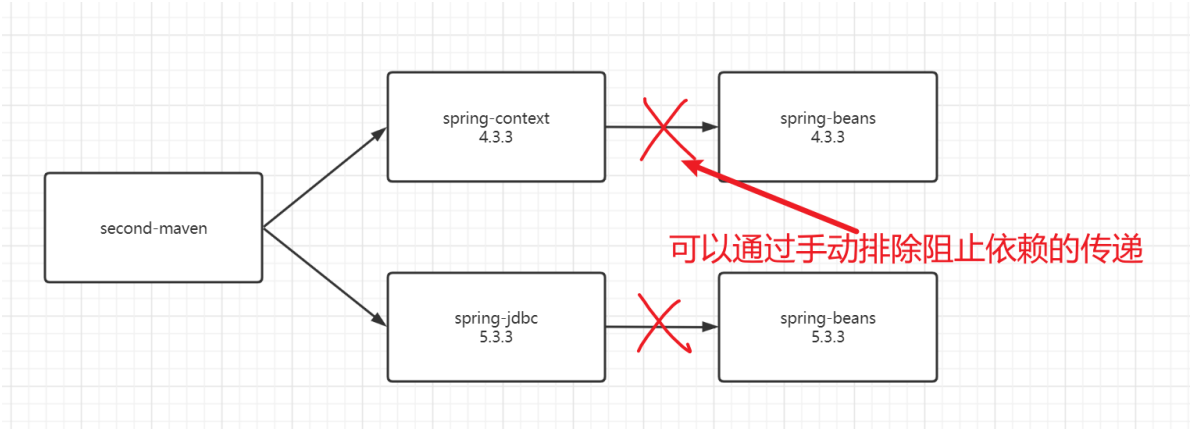
就近原则

就近原则是指谁传递的次数比较少，以谁为准。（就近原则的优先级要高于声明优先原则）

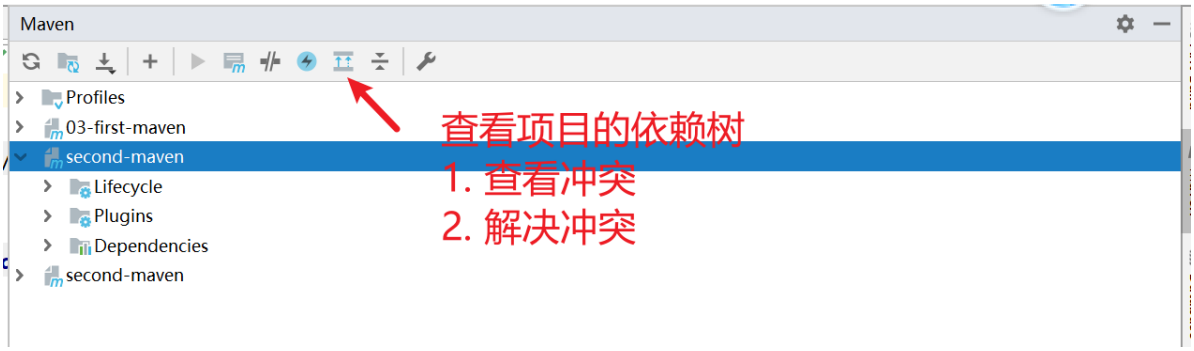


手动排除

阻止依赖的传递。把传递过来的依赖手动排除掉。




```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.3.3</version>

    <exclusions>
        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-beans</artifactId>
        </exclusion>

        <exclusion>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
        </exclusion>

    </exclusions>

</dependency>
```



提取常量

提取常量是指我们可以把一些jar包的版本提取出来，统一管理。

（是一个预防冲突的手段）


```
<properties>
    <spring.version>5.3.3</spring.version>
    <mysql.version>5.1.47</mysql.version>
    <junit.version>4.12</junit.version>
    <c3p0.version>0.9.5.5</c3p0.version>
</properties>

<dependencies>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>${spring.version}</version>

        <exclusions>
```

6.4 scope作用域

项目中的每一个依赖都有自己的作用域。

- test

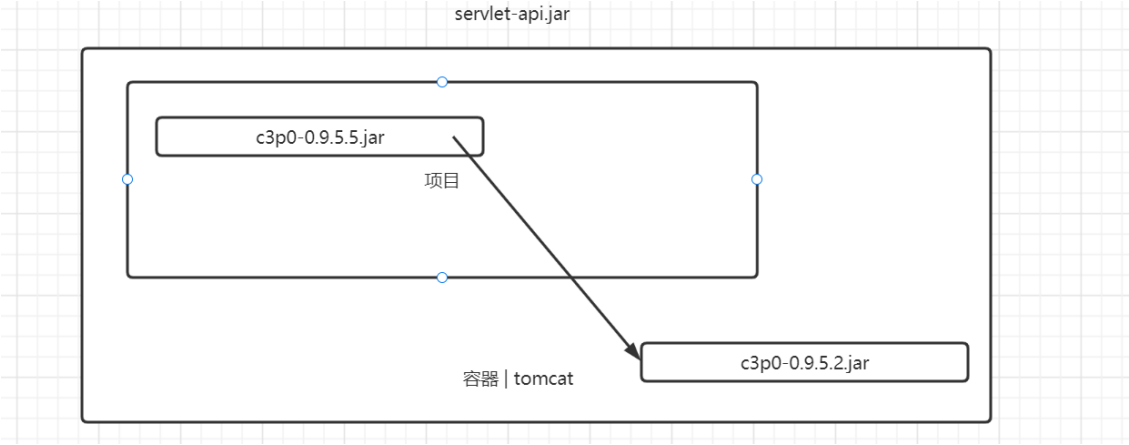
  > 这个依赖（jar包）只在src/test/java路径下生效，在src/mian/java路径下不生效

- runtime

  > 这个依赖（jar包）在运行的时候才生效，但是在编译的时候不生效

- provided

  > 这个作用域修饰的依赖在编译的时候生效，但是一运行起来就失效了



- compile

> 不管是在编译的时候，还是在运行的时候，在src/mian/java和src/test/java路径下都有效
>
> 默认的作用域

7.使用Maven开发

DBCP

```
static {

        try {

            // 创建DBCP的数据库连接池
            Properties properties = new Properties();
//            properties.load(new
FileInputStream("target/classes/dbcp.properties"));

            // 获取类加载器
            ClassLoader classLoader = DBCPUtils.class.getClassLoader();
            // 注意：这种方式是找的 target/classes/dbcp.properties，不是找的
src/main/resources下面的配置文件
            InputStream inputStream =
classLoader.getResourceAsStream("dbcp.properties");
            properties.load(inputStream);

            dataSource = BasicDataSourceFactory.createDataSource(properties);

        }catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```