

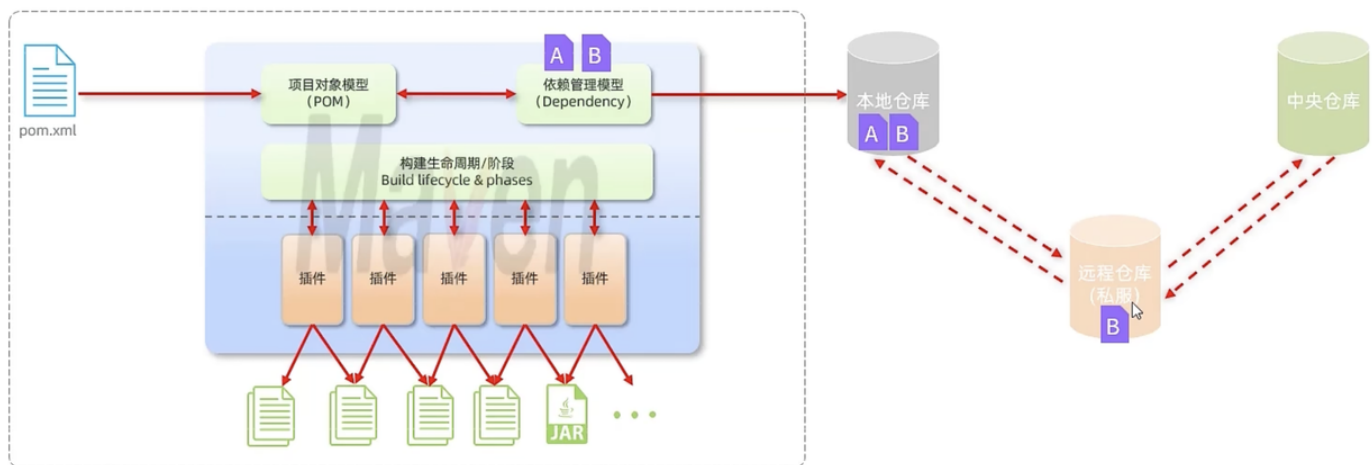
Maven详解

1. 工作原理（What）

基于Maven项目管理工具构建的项目叫做Maven项目

Maven究竟能做什么？

- 图解



1.1 构建项目生命周期

项目手动执行编译、打包、布署，是不是会过于麻烦？

- 生命周期这个词一般用于描述对象从产生到销毁的整个流程。Maven将整个项目作为一个对象进行管理，这个项目从编译到发布的整个过程也可以称之为该项目的生命周期。
- Maven基于插件帮助我们完成项目生命周期的构建（编译、测试等）
- 插件举例
 - maven-compiler-plugin 负责编译源码，对应compile阶段
 - maven-surefire-plugin 负责运行单元测试，对应test阶段

1.2 项目对象模型（POM）

- 用groupId、artifactId、version三个“向量”组成该项目的唯一坐标（Coordinates）/标识
- 在xml文件中

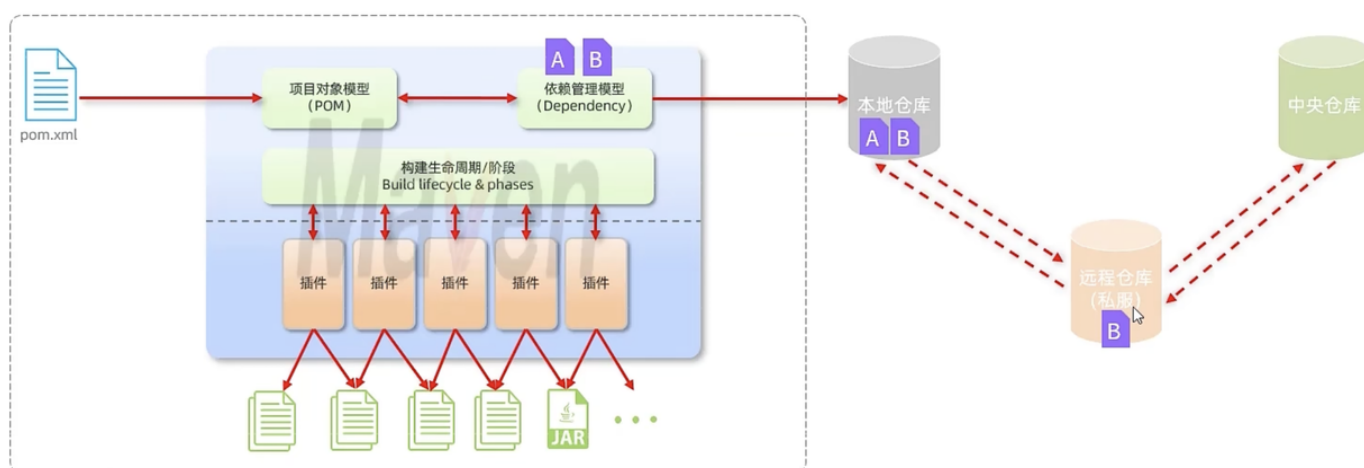
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.werun</groupId>
    <artifactId>shike-songshuo</artifactId>
    <version>1.0-SNAPSHOT</version>
```

1.3 依赖管理模型（Dependency）

项目需要引用这么多jar包，如何下载、版本怎么管理？

- Maven可以帮助我们自动引入当前项目所需要的jar包
- 按照本地——远程——中央仓库的次序依次查找jar包
- jar包：出于盈利或者非盈利的目的，将自己的通用工具或者框架抽取出来，打成jar包供他人调用
- 依赖
 - 动词含义：某个项目依赖于某个jar包实现功能
 - 名词含义：当前项目所需要的jar包，一个项目可以引入多个依赖

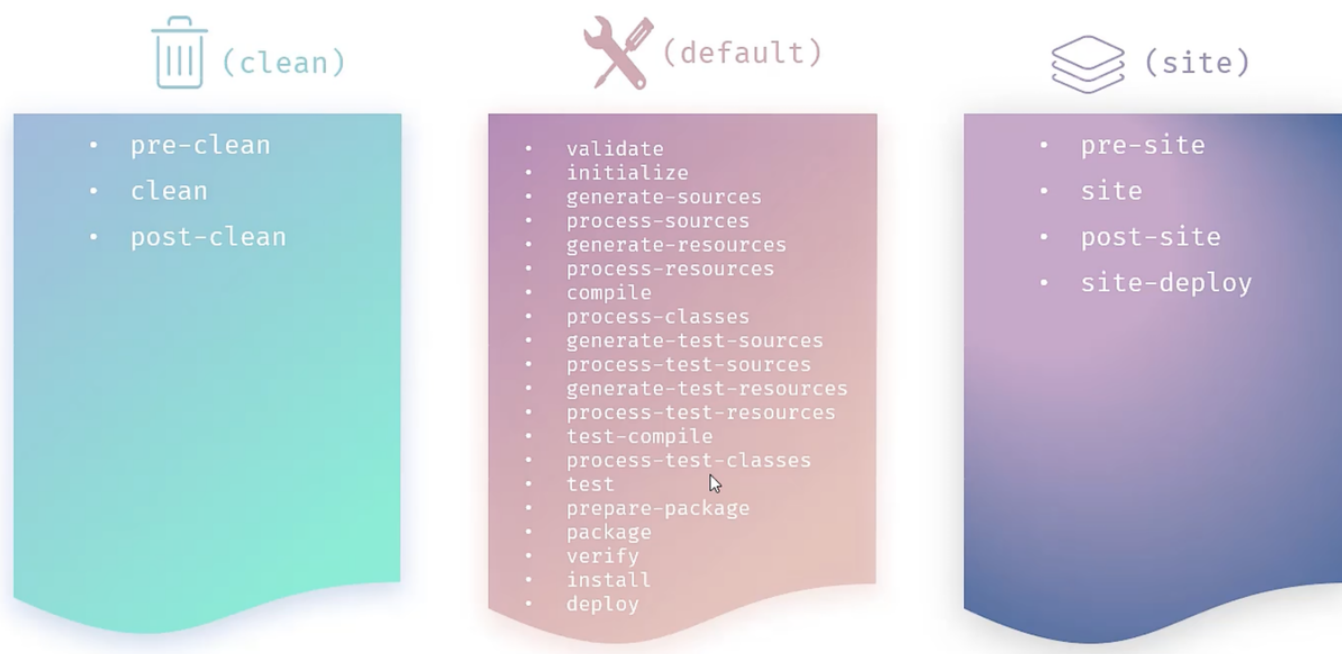


2. 具体实现（How）

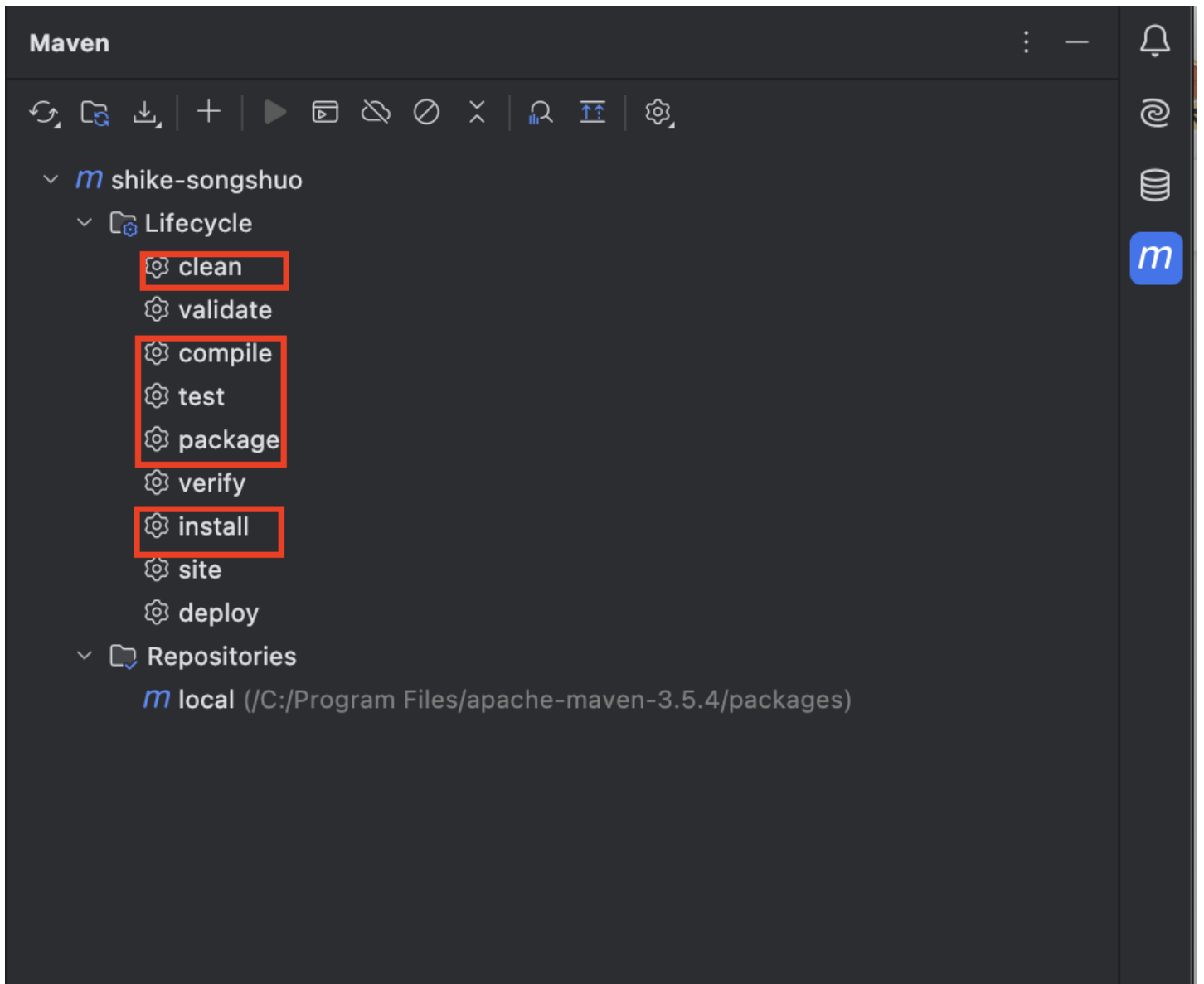
Maven怎么做到的？

2.1 提供标准化的构建流程

- Maven中有三套相互独立的生命周期
 - clean：清理工作
 - default：核心工作：编译、测试、打包、安装到本地仓库、部署等
 - site：生成报告，发布站点等



- 我们关注的核心：（在IDEA中常用的指令）
 - clean： 清除上一次构建好的Target目录下的所有文件
 - compile： 编译，将源码转为字节码文件（.class）
 - test： 测试代码编译后的字节码文件（用单元测试）
 - package： 打包成jar/war文件
 - install： 安装项目到本地仓库



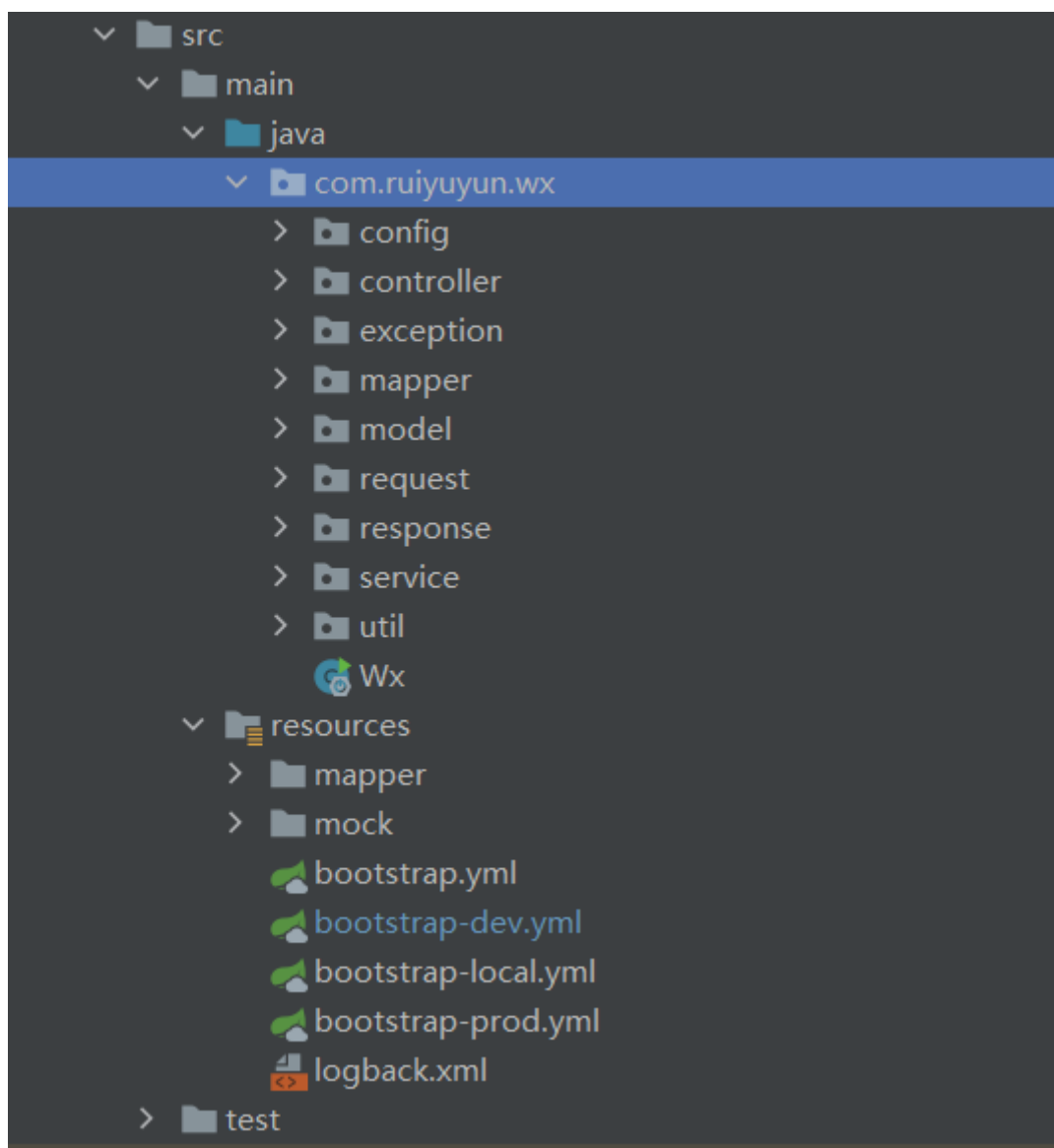
2.2 提供标准化的项目结构

- 模型化思想：将所有Maven项目抽象为一个标准化的模型，所有项目都基于这个模型。

类似 类与对象的关系

- 根目录：工程名

```
|---src:           //源码
|---|---main:      //存放主程序
|---|---|---java:  //源码文件
|---|---|---resources: //存放框架的资源文件
|---|---test:      //存放测试程序
|---|---|---java:  //java测试文件
|---|---|---resources: //java测试的资源文件
|---pom.xml:       //maven的核心配置文件
```



2.3 提供一套依赖管理机制

- 引入依赖：
 1. 在 pom.xml 中编写 <dependencies> 标签
 2. 在 <dependencies> 标签中使用 <dependency> 引入坐标
 3. 定义坐标的 groupId, artifactId, version
 4. 点击刷新按钮，引入最新加入的坐标

```
1  <dependencies>
2
3      <!-- SpringBoot框架 -->
4      <dependency>
5          <groupId>org.springframework.boot</groupId>
6          <artifactId>spring-boot-starter-web</artifactId>
7      </dependency>
8      <dependency>
9          <groupId>org.springframework.boot</groupId>
```

```
10         <artifactId>spring-boot-starter-test</artifactId>
11         <scope>test</scope>
12     </dependency>
```

3. 详解POM.XML文件

3.1 认识POM

- pom.xml 文件是 Maven 项目的“大脑”
- 通过标准化配置解决了依赖管理、构建流程、多模块协作等问题。
- 掌握其核心标签（如 <dependencies>、<build>、<parent>）和概念（坐标、作用域、生命周期）是高效使用 Maven 的关键

3.2 POM 的核心作用

3.2.1 项目描述

定义项目的基本信息（如版本与规则、名称、开发者等）

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

  <groupId>org.werun</groupId>
  <artifactId>shike-songshuo</artifactId>
  <version>1.0-SNAPSHOT</version>
```

```
1  <?xml version="1.0" encoding="UTF-8"?>
```

- 声明xml版本信息1.0，文件字符编码方式为UTF-8
- 根标签<project>：定义xml命名空间与规则
- xmlns(XML NameSpace):表明所有未加前缀的标签（例如<groupId>）都来自Maven POM 4.0.0规范

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
```

- xsi (XML Schema Instance) 前缀对应的命名空间为XML Schema

```
1 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

- IDEA或Maven会根据这个Schema文件进行标签是否正确的校验

```
1 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
2 http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

- 标签版本

```
1 <modelVersion>4.0.0</modelVersion>
```

- 项目坐标

```
1 <groupId>org.werun</groupId>  
2 <artifactId>shike-songshuo</artifactId>  
3 <version>1.0-SNAPSHOT</version>
```

3.2.2 依赖管理

声明项目依赖的第三方仓库（自动从仓库下载）。

- <properties>

用于管理依赖的版本信息（本质是定义一些在pom文件中可以直接使用的常量，）

用\${}符号使用

```

1  <properties> <!--声明相应的版本信息-->
2  <!--统一字符集编码-->
3      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
4
5      <!-- 数据库驱动 -->
6      <mysql-connector-java.version>8.0.27</mysql-connector-java.version>
7      <redis.version>2.7.18</redis.version>

```

- <dependencies>

各种依赖管理：

```

1  <dependencies>
2
3      <!-- SpringBoot框架 -->
4      <dependency>
5          <groupId>org.springframework.boot</groupId>
6          <artifactId>spring-boot-starter-web</artifactId>
7      </dependency>
8      <dependency>
9          <groupId>org.springframework.boot</groupId>
10         <artifactId>spring-boot-starter-test</artifactId>
11         <scope>test</scope>
12     </dependency>

```

- <repositories>

用于标明远程仓库地址，在这个新库中下载jar包，覆盖了默认的远程仓库

需要标注的远程仓库信息：

- <name>
- <id>
- <url>
- <releases>稳定版
- <snapshots>快照版

```

1  <repositories>
2      <repository>
3          <id>aliyun</id>

```



```

4         <name>Aliyun Maven</name>
5         <url>https://maven.aliyun.com/repository/public</url>
6         <releases>
7             <enabled>true</enabled> <!--启用Releases版本下载 -->
8         </releases>
9         <snapshots>
10            <enabled>false</enabled><!--禁用Snapshots 版本下载-->
11        </snapshots>
12    </repository>
13 </repositories>

```

3.2.3 构建配置

指定编译、测试、打包等构建流程的规则。

- <packaging>

打包方式，默认为jar，如果需要多模块管理的话需要改为pom形式

```
<packaging>pom</packaging>
```

- <build>

管理项目的构建过程（如编译、打包），内容包括

- 插件配置：<plugins> 定义构建过程中使用的插件
- 资源管理：<resource> 控制源码资源（如配置文件）的复制与过滤规则
- 目录管理：<directory> 自定义输出目录（如字节码文件位置、打包文件存放路径等）

```

1  <build>
2      <plugins>
3          <plugin>
4              <groupId>org.springframework.boot</groupId>
5              <artifactId>spring-boot-maven-plugin</artifactId>
6          </plugin>
7      </plugins>
8
9      <resources>
10         <resource>
11             <directory>src/main/resources</directory>
12             <includes>
13                 <include>**/*</include>
14             </includes>
15             <filtering>false</filtering>

```

```
16         </resource>
17     </resources>
18 </build>
```

3.2.4 继承与聚合

大型项目拆分了多个子模块，各模块之间如何引用？

Maven支持多模块项目的父子关系、聚合关系管理。

- <parent>

```
1  <parent> <!--继承父项目-->
2      <groupId>org.springframework.boot</groupId>
3      <artifactId>spring-boot-starter-parent</artifactId>
4      <version>2.7.18</version>
5      <relativePath/>
6  </parent>
```

- 详见后文继承
 - 概括来说，就是“复用公共配置，统一管理多模块项目的依赖、插件、规则”，避免重复配置，降低复用率。
 - relativePath元素不是必须的，指定后会优先从指定的位置查找父pom
- <dependencyManagement>

写在父工程内，用于管理子工程所用依赖的版本，注意子工程还是需要引入依赖，只是不需写版本号了

```
1  <dependencyManagement>
2      <dependencies>
3          <!-- spring boot -->
4          <dependency>
5              <groupId>org.mybatis.spring.boot</groupId>
6              <artifactId>mybatis-spring-boot-starter</artifactId>
7              <version>${mybatis-spring-boot-starter.version}</version>
8          </dependency>
9      </dependencies>
10 </dependencyManagement>
```

- `<modules>`

用于聚合项目或者多模块项目（我们考核项目中的微服务的其中一种实现方式）

通过一个pom打包的项目可以将它们列为模块来聚合成一组项目进行构建

```
1  <modules>
2      <module>my-project</module>
3      <module>another-project</module>
4  </modules>
```

3.3 POM中的核心概念

3.3.1 坐标（Coordinates）

- 坐标是资源（jar包）的唯一标识，通过坐标可以定位该资源位置
- 用坐标来定义项目或引入项目中需要的依赖

```
1  <groupId>org.werun</groupId>
2  <artifactId>shike-songshuo</artifactId>
3  <version>1.0-SNAPSHOT</version>
```

```
1  <dependencies>
2
3      <!-- SpringBoot框架 -->
4      <dependency>
5          <groupId>org.springframework.boot</groupId>
6          <artifactId>spring-boot-starter-web</artifactId>
7      </dependency>
8      <dependency>
9          <groupId>org.springframework.boot</groupId>
10         <artifactId>spring-boot-starter-test</artifactId>
11         <scope>test</scope>
12     </dependency>
```

- 组成

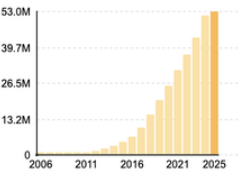
- groupId：公司或组织域名的倒序，通常也会加上项目名称。（如：com.lfc.mall）
- artifactId：模块的名称，一般为项目名或者模块名，将来作为Maven工程的工程名。
- version：模块的版本号，根据需要决定。（如：1.1-SNAPSHOT 或者 2.0-RELEASE）
 - SNAPSHOT 快照版，功能不稳定，持续开发中
 - RELEASE 发行版，功能稳定，已经停止更新

由这三个“向量”确定该jar包在仓库中的坐标

查找坐标

查找坐标


Indexed Artifacts (52.9M)



Popular Categories

- Testing Frameworks & Tools
- Android Packages
- Logging Frameworks
- JVM Languages
- Java Specifications
- JSON Libraries
- Core Utilities
- Mocking
- Annotation Libraries
- Web Assets
- Language Runtime
- HTTP Clients
- Logging Bridges
- Dependency Injection
- XML Processing
- Web Frameworks
- Android Platform
- Defect Detection Metadata

Home » com.mysql » mysql-connector-j



MySQL Connector/J

MySQL Connector/J is a JDBC Type 4 driver, which means that it is pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries. This driver supports auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for large update counts, support for local and offset date-time variants from the java.time package, support for JDBC-4.x XML processing, support for per connection client information and support for the NCHAR, NVARCHAR ...

Categories	JDBC Drivers
Tags	database sql jdbc driver connector rdbms mysql connection
HomePage	http://dev.mysql.com/doc/connector-j/en/
Ranking	#538 in MvnRepository (See Top Artifacts) #7 in JDBC Drivers
Used By	1,042 artifacts

Central (10)

Redhat GA (1)

Redhat EA (1)

	Version	Vulnerabilities	Repository	Usages	Date
9.2.x	9.2.0		Central	70	Jan 21, 2025
9.1.x	9.1.0		Central	164	Oct 15, 2024
9.0.x	9.0.0		Central	117	Jul 02, 2024
8.4.x	8.4.0		Central	154	Apr 30, 2024
8.3.x	8.3.0		Central	361	Jan 16, 2024
8.2.x	8.2.0		Central	204	Oct 25, 2023
8.1.x	8.1.0	1 vulnerability	Central	176	Jul 18, 2023
	8.0.33	1 vulnerability	Central	384	Apr 18, 2023
8.0.x	8.0.32	1 vulnerability	Central	154	Jan 18, 2023
	8.0.31	1 vulnerability	Central	115	Oct 14, 2022

Logging Frameworks

JVM Languages

Java Specifications

JSON Libraries

Core Utilities

Mocking

Annotation Libraries

Web Assets

Language Runtime

HTTP Clients

Logging Bridges

Dependency Injection

XML Processing

Web Frameworks

Android Platform

Defect Detection Metadata

I/O Utilities

Code Generators

Configuration Libraries

Concurrency Libraries

OSGi Utilities

JDBC Drivers

Reflection Libraries

Date and Time Utilities

HomePage

http://dev.mysql.com/doc/connector-j/en/

Date

Apr 18, 2023

Files

pom (3 KB)

jar (2.4 MB)

View All

Repositories

Central

AlessioDP

Alfresco

Auxilor

Fit2Cloud

Kyligence Public

Mulesoft

WSO2 Public

Ranking

#538 in MvnRepository (See Top Artifacts)

#7 in JDBC Drivers

Used By

1,042 artifacts

Vulnerabilities

Direct vulnerabilities:

CVE-2023-22102

Vulnerabilities from dependencies:

CVE-2024-7254

Note: There is a new version for this artifact

New Version

9.2.0

Maven

Gradle

SBT

Ivy

Grape

Leiningen

Buildr

Scope:

Compile

<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

☒ Include backlinks

3.3.2 依赖（Dependencies）

3.3.2.1 依赖范围

标签位置：<dependencies> <dependency> <scope>

可选值：compile / test / runtime / provided / system / import

- compile：默认值，在编译、测试、运行中都需要的依赖
- test：只在测试中用到，例如Junit
- runtime：只在运行时需要，例如JDBC驱动
- provided：在编译与测试中用到，运行不需要
- system：在本地下载的依赖（不建议使用，会造成移植的不稳定性）
- import：只在<dependencyManegement>中用到，导入其他项目的依赖管理配置

3.3.2.2 依赖传递

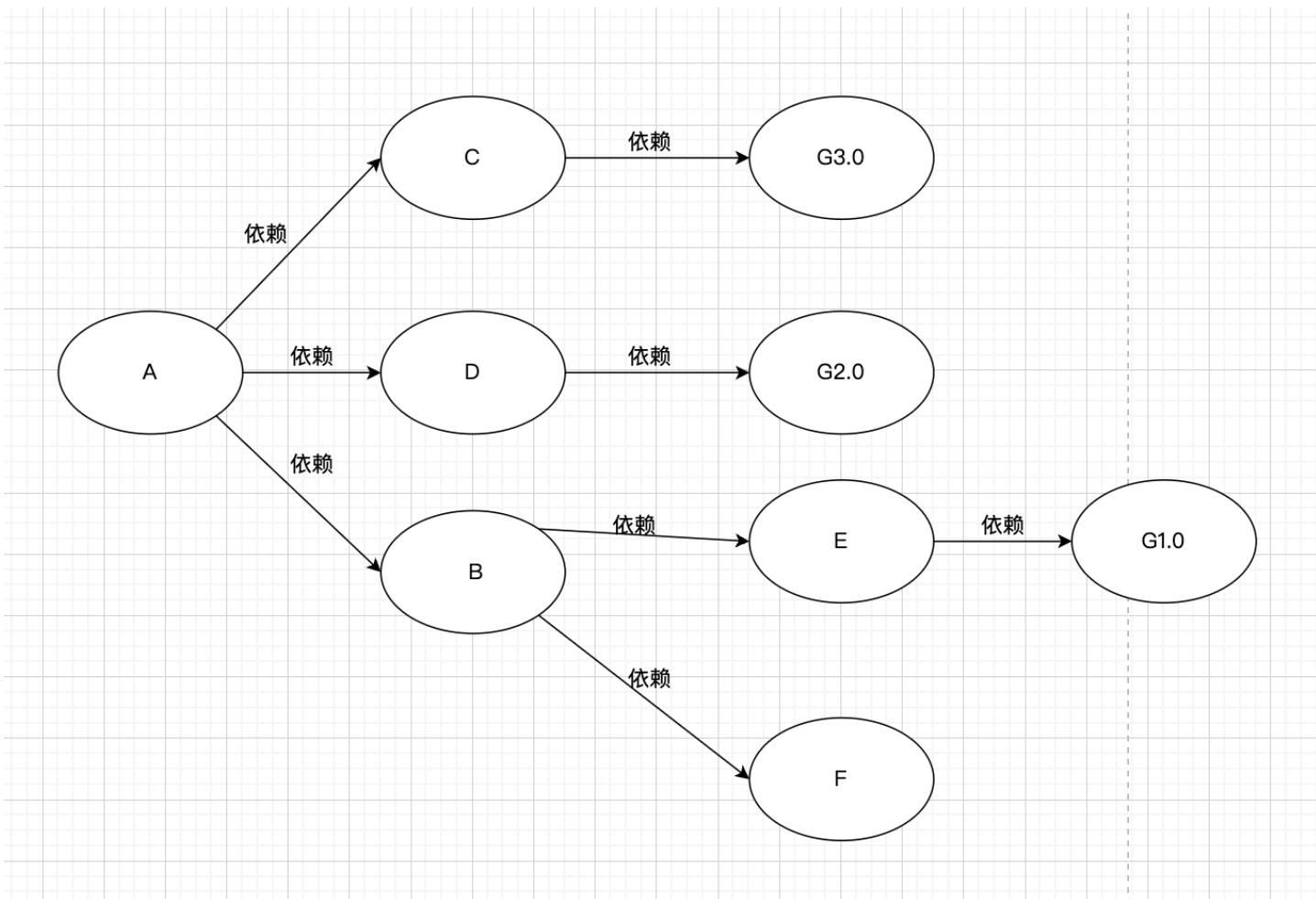
- 依赖传递：A项目依赖B，B依赖C
- 是否可以进行依赖传递取决于依赖范围的设置

- B依赖C时使用的是compile范围，可以依赖传递
- B依赖C时使用的test或provided范围，不可以依赖传递，此时需要手动引入C

3.3.2.3 依赖冲突

- 依赖冲突的两种情况

1. 如下图的依赖关系，A多次间接依赖了G，由于多个G版本不同造成的依赖冲突



2. 引入的这些模块中具有相同路径的类（包名+类名完全相同）造成的依赖冲突

- 解决依赖冲突

1. 版本冲突：由Maven自动进行依赖仲裁

- 最短路径优先：优先使用间接依赖路径最短的版本，如上图G1.0和G2.0，优先使用2.0版
- 先声明优先：如果间接依赖路径一样长，使用先声明的依赖版本，如上图G2.0和G3.0，在pom文件中先声明的C，故优先使用G3.0

利用这两条规则可以选择自己想保留的版本

2. 相同路径冲突：手动解决

- a. 可以借助 Maven 的生命周期插件 maven-enforcer-plugin，找出类路径相同的模块
- b. 用<exclusions>排除依赖传递

```
1  <exclusions>
2      <exclusion>
3          <groupId>要弃用的模块ID</groupId>
4          <artifactId>要弃用的模块artifactId</artifactId>
5      </exclusion>
6  </exclusions>
```

3.3.3 仓库 (Repositories)

- 本地仓库

缓存下载的依赖与插件，避免重复下载

默认路径：用户目录/.m2/repository

- 远程私有仓库

企业或团队搭建的内部仓库，托管私有依赖和加速访问

- 远程中央仓库

Maven官方维护的公共仓库，包含大多数开源依赖

[Maven官方仓库](#)

- 镜像仓库

代替某个或所有远程仓库，能够优化下载速度

国内仓库镜像（Mirror）[阿里云 Maven 镜像仓库地址](#)

在settings.xml中修改仓库地址

```
1  <mirrors>
2      <mirror>
3          <id>aliyun</id>
4          <name>Aliyun Mirror</name>
5          <url>https://maven.aliyun.com/repository/public</url>
6          <mirrorOf>*</mirrorOf><!--匹配所有仓库-->
```

```
7     </mirror>
8 </mirrors>
```

3.3.4 继承与聚合 (Inheritance and Aggregation)

3.3.4.1 继承 (Inheritance)

- 通过父项目中的pom文件管理子项目/模块的依赖版本、插件配置和属性
- 子项目中用<parent>标签声明继承关系，该标签的使用详见上文
- 子项目继承父项目中的依赖
 - 父项目<dependencies>中声明的依赖，子项目可以直接使用无需再次声明
 - 父项目<dependencyManagement>中声明的依赖，子项目仍需再次声明，但是无需声明版本号，版本号与父项目保持一致。

3.3.4.2 聚合 (Aggregation)

- 多个子模块聚合到父模块中，在父模块中用<modules>声明子模块
- 用于统一构建多个模块，聚合之后，在总工程执行mvn install 可以一键完成安装，而且会自动按照正确的顺序执行
- 没有配置上的关联与继承，各个子模块独立性高

可以联合使用依赖继承与依赖聚合，便于统一管理配置与简化多模块构建过程

3.3.5 Target目录

- 在该目录下储存的文件：
 - classes (编译后的字节码文件)
 - test-classes (测试代码编译后的字节码文件)
 - surefire- reports (单元测试报告，哪些测试通过哪些失败)
 - 项目名.jar/war(最终打包好的文件)
- 该目录由Maven自动创建，一般不提交到代码库中
- 有些时候代码已经改动但运行结果没变，是因为仍然使用原来的target目录中的编译结果，需要执行mvn clean命令来清空该目录