

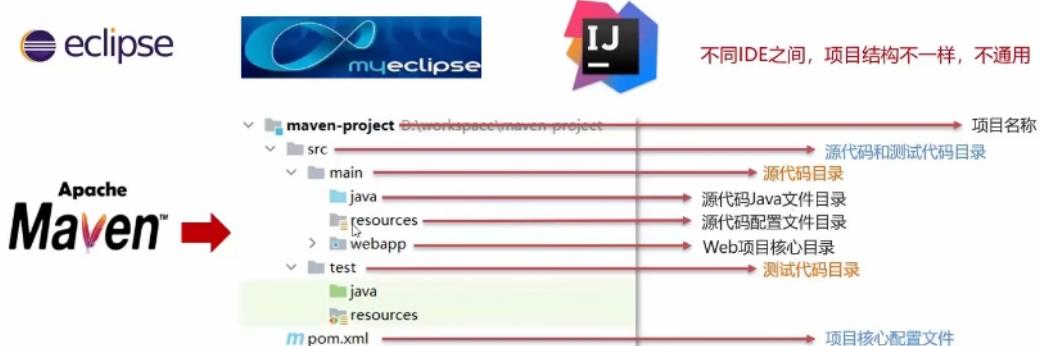
Maven

1、Maven概述



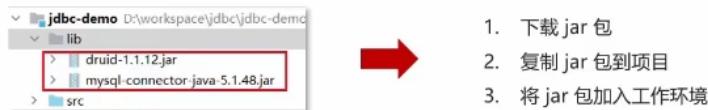
- ◆ Maven 简介
- ◆ Maven 安装配置
- ◆ Maven 基本使用
- ◆ IDEA 配置 Maven
- ◆ 依赖管理

- Maven是专门用于管理和构建Java项目的工具，它的主要功能有：
 - 提供了一套标准化的项目结构
 - 提供了一套标准化的构建流程（编译，测试，打包，发布.....）
 - 提供了一套依赖管理机制
- 标准化的项目结构

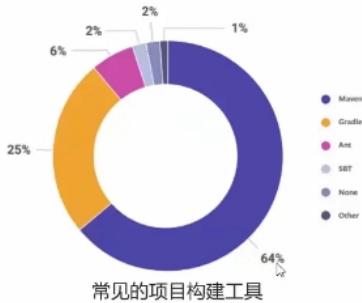


Maven提供了一套标准化的项目结构，所有IDE使用Maven构建的项目结构完全一样，所有IDE创建的Maven项目可以通用

- Maven是专门用于管理和构建Java项目的工具，它的主要功能有：
 - 提供了一套标准化的项目结构
 - 提供了一套标准化的构建流程（编译，测试，打包，发布.....）
 - 提供了一套依赖管理机制
- 依赖管理
 - 依赖管理其实就是管理你项目所依赖的第三方资源（jar包、插件...）



- Maven是专门用于管理和构建Java项目的工具，它的主要功能有：
 - 提供了一套标准化的项目结构
 - 提供了一套标准化的构建流程（编译，测试，打包，发布.....）
 - 提供了一套依赖管理机制
- 依赖管理
 - 依赖管理其实就是管理你项目所依赖的第三方资源（jar包、插件...）



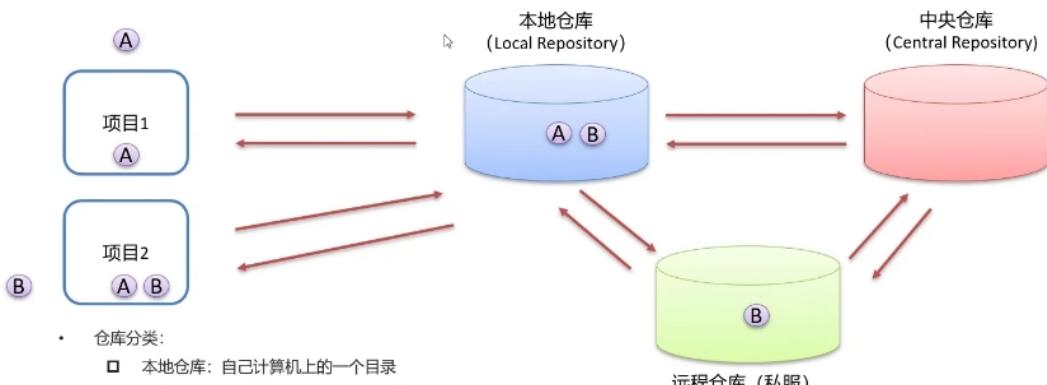
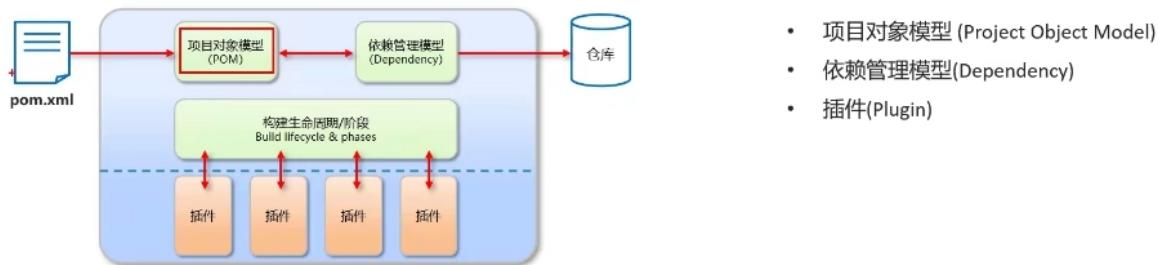
2、Maven简介

- Apache Maven 是一个项目管理和构建工具，它基于项目对象模型(POM)的概念，通过一小段描述信息来管理项目的构建、报告和文档
- 官网：<http://maven.apache.org/>

Maven 作用：

- 标准化的项目结构
- 标准化的构建流程
- 方便的依赖管理

Maven 模型：



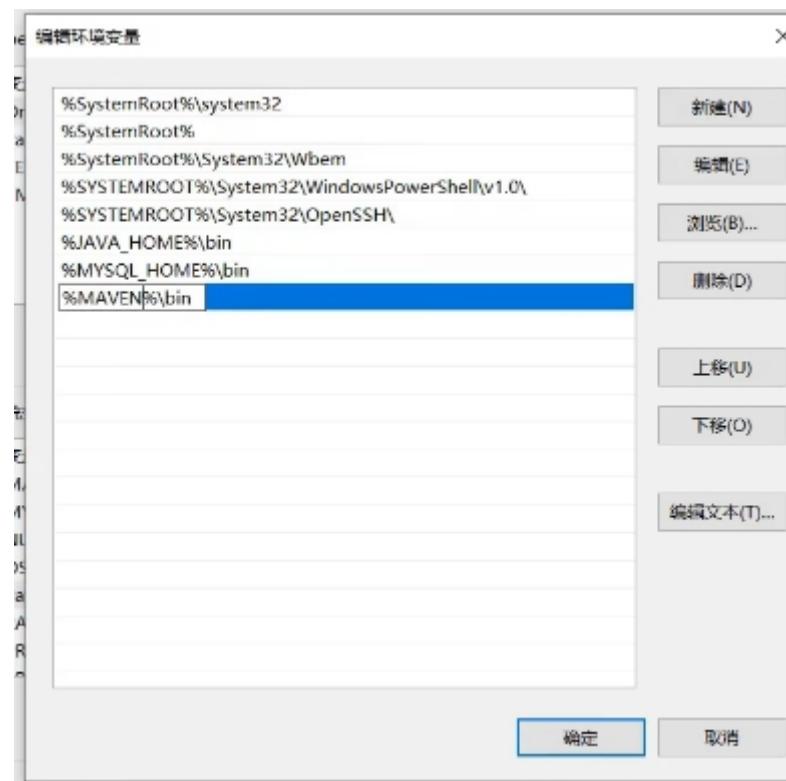
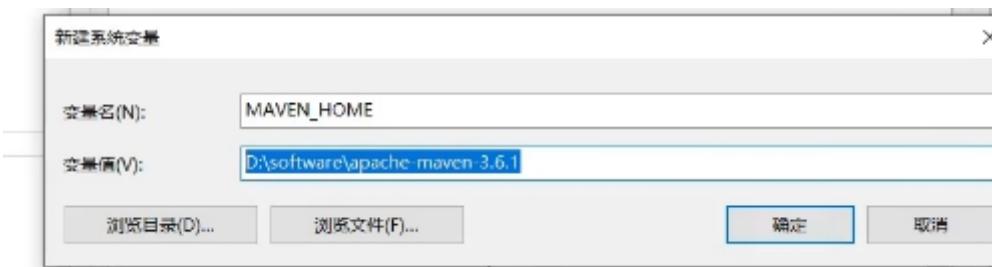
- 仓库分类：
 - 本地仓库：自己计算机上的一个目录
 - 中央仓库：由Maven团队维护的全球唯一的仓库
 - 地址：<https://repo1.maven.org/maven2/>
 - 远程仓库(私服)：一般由公司团队搭建的私有仓库
- 当项目中使用坐标引入对应依赖jar包后，首先会查找本地仓库中是否有对应的jar包：
 - 如果有，则在项目直接引用；
 - 如果没有，则去中央仓库中下载对应的jar包到本地仓库。
- 还可以搭建远程仓库，将来jar包的查找顺序则变为：
 - 本地仓库 → 远程仓库 → 中央仓库

3、Maven安装&配置及基本使用

如果要安装Maven配置就跟<https://www.bilibili.com/video/BV1Qf4y1T7Hx?p=44>

1. 解压 apache-maven-3.6.1.rar 既安装完成
2. 配置环境变量 MAVEN_HOME 为安装路径的bin目录
3. 配置本地仓库：修改 conf/settings.xml 中的 <localRepository> 为一个指定目录
4. 配置阿里云私服：修改 conf/settings.xml 中的 <mirrors>标签，为其添加如下子标签

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



```
<localRepository>D:\software\apache-maven-3.6.1\mvn_resp</localRepository>
<!-- interactiveMode -->
```

目录

Contents

- ◆ Maven 概念模型
- ◆ Maven 安装配置
- ◆ **Maven 基本使用**
- ◆ IDEA 配置 Maven
- ◆ 依赖管理



Maven 基本使用

- Maven 常用命令
- Maven 生命周期

Maven 常用命令

- `compile` : 编译
- `clean`: 清理
- `test`: 测试
- `package`: 打包
- `install`: 安装

Maven 生命周期

- Maven 构建项目生命周期描述的是一次构建过程经历经历了多少个事件
- 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, 安装等
 - site: 产生报告, 发布站点等

同一生命周期内, 执行后边的命令, 前边的所有命令会自动执行



default 构建生命周期

● 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 (JUnit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)	进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。

exp:

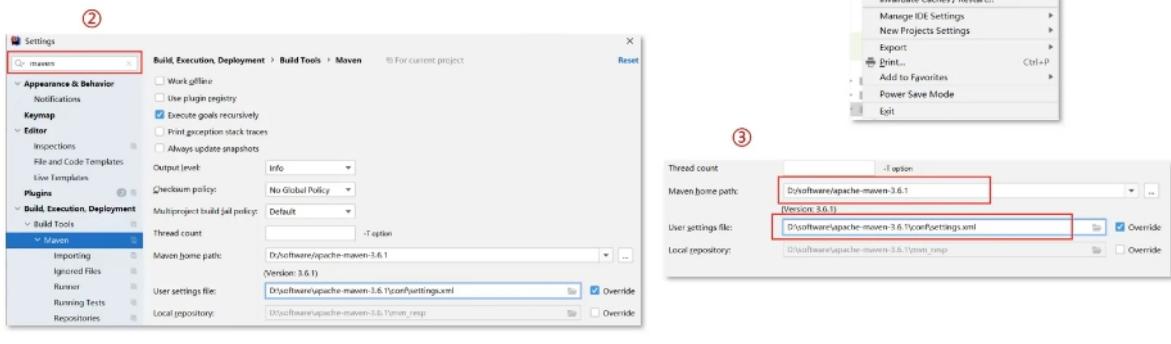
```
PS E:\2021JavaWeb\day04-Maven\代码\maven-project> mvn instal
```

4、IDEA配置Maven

步骤

IDEA 配置 Maven 环境

- 选择 IDEA 中 File --> Settings
- 搜索 maven
- 设置 IDEA 使用本地安装的 Maven，并修改配置文件路径



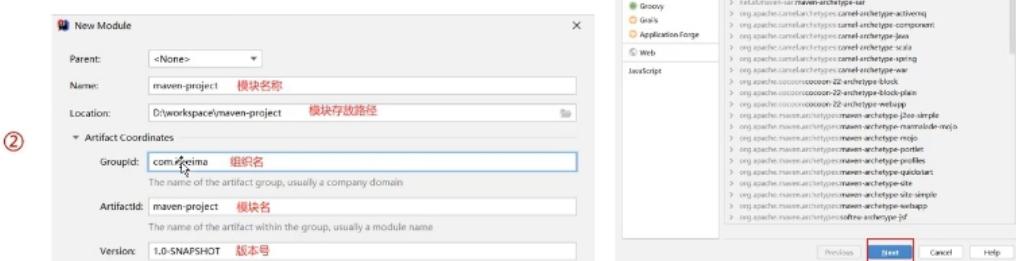
Maven 坐标详解



步骤

IDEA 创建 Maven 项目

- 创建模块，选择 Maven，点击 Next
- 填写模块名称，坐标信息，点击 finish，创建完成
- 编写 HelloWorld，并运行



04

IDEA 配置 Maven

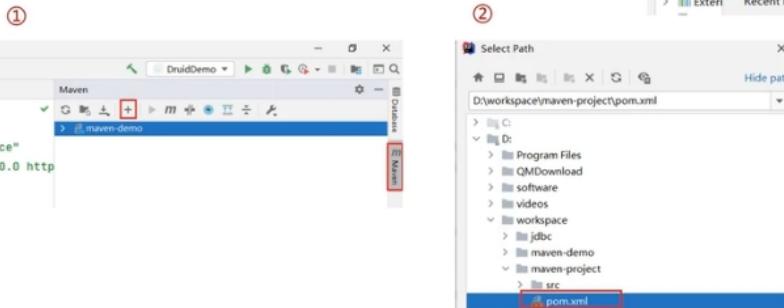
- IDEA 配置 Maven 环境
- Maven 坐标详解
- IDEA 创建 Maven 项目
- IDEA 导入 Maven 项目

步骤

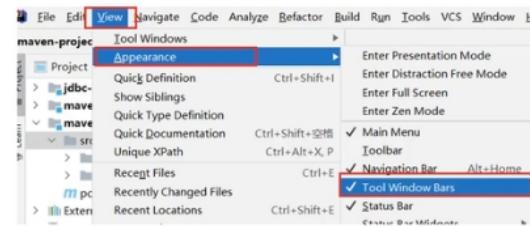
IDEA 导入 Maven 项目

1. 选择右侧Maven面板，点击 + 号
2. 选中对应项目的pom.xml文件，双击即可
3. 如果没有Maven面板，选择

View → Appearance → Tool Window Bars



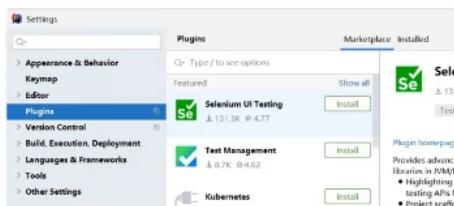
③



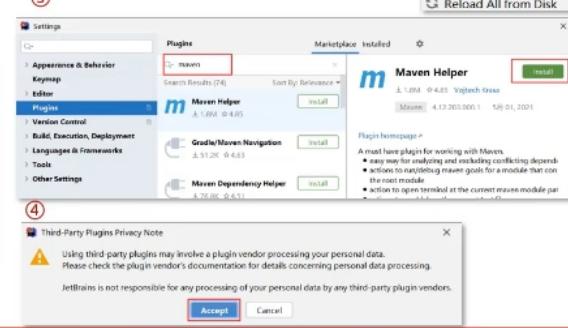
配置 Maven-Helper 插件

1. 选择 IDEA 中 File --> Settings
2. 选择 Plugins
3. 搜索 Maven，选择第一个 Maven Helper，点击 Install 安装，弹出面板中点击 Accept
4. 重启 IDEA

②



③



④

5、依赖管理&依赖范围

目录

Contents

- ◆ Maven 概念模型
- ◆ Maven 安装配置
- ◆ Maven 基本使用
- ◆ IDEA 配置 Maven
- ◆ 依赖管理



步骤 使用坐标导入 jar 包

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, 使坐标生效

①

```
<dependencies>
    <!-- mysql 坐标 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.32</version>
    </dependency>
</dependencies>
```

②

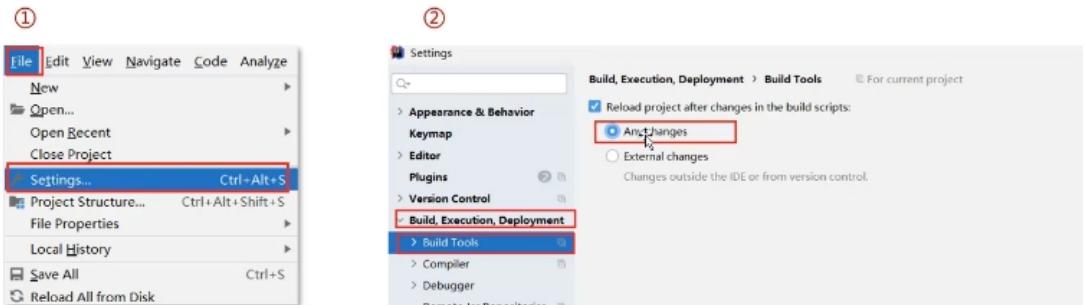




步骤

使用坐标导入 jar 包 – 自动导入

1. 选择 IDEA 中 File --> Settings
2. 在弹出的面板中找到 Build Tools
3. 选择 Any changes, 点击 ok 即可生效



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

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13</version>
    <scope>test</scope>
</dependency>
```

依赖范围	编译classpath	测试classpath	运行classpath	例子
compile	Y	Y	Y	logback
test	-	Y	-	Junit
provided	Y	Y	-	servlet-api
runtime	-	Y	Y	jdbc驱动
system	Y	Y	-	存储在本地的jar包
import	引入DependencyManagement			

- <scope>默认值： compile

6、分模块开发的意义



目录

Contents

◆ 分模块开发与设计

◆ 依赖管理



◆ 聚合与继承

◆ 属性管理

◆ 多环境配置与应用

◆ 私服



学习目标

Learning Objectives

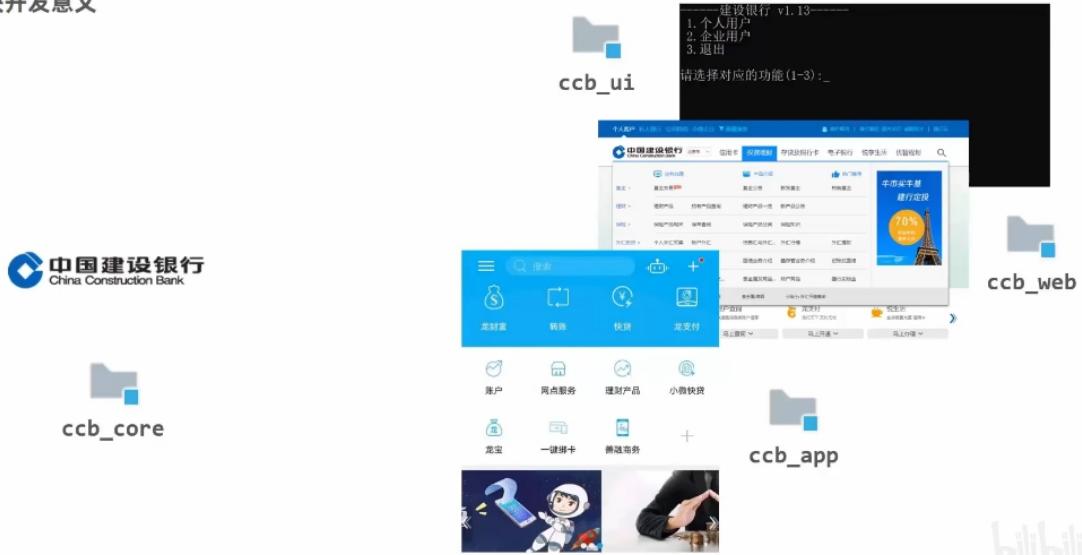
1. 理解分模块开发的意义
2. 能够使用聚合工程快速构建项目
3. 能够使用继承简化项目配置
4. 能够根据需求配置生产、开发、测试环境，并在各环境间切换运行



01

分模块开发与设计

分模块开发意义

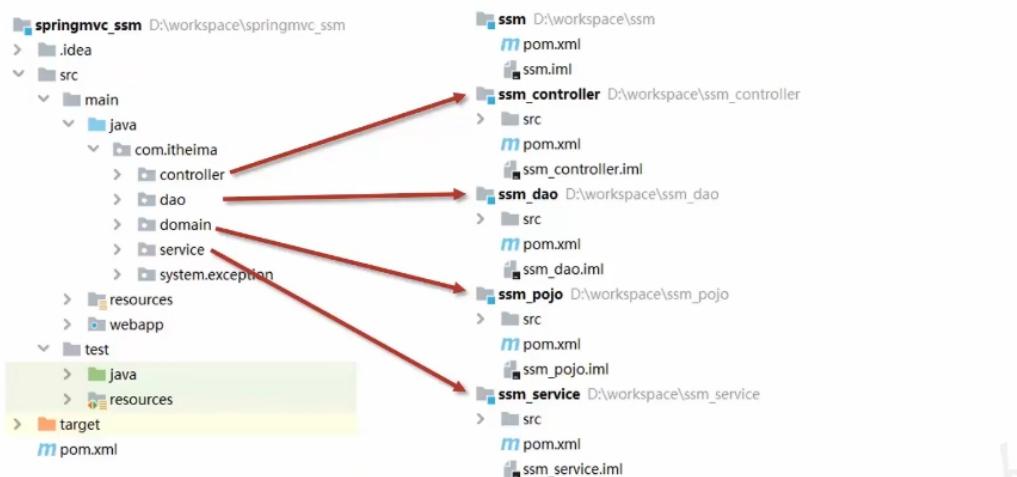


分模块开发意义



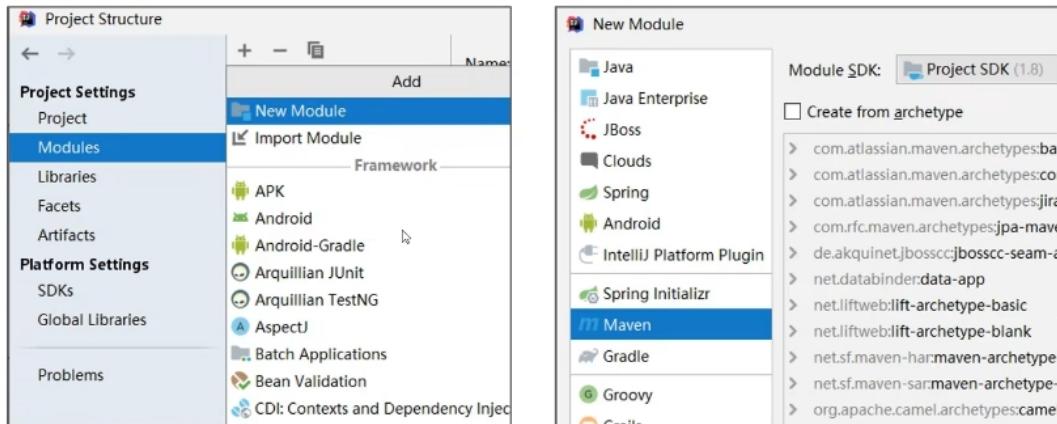
分模块开发意义

- 将原始模块按照功能拆分成若干个子模块，方便模块间的相互调用，接口共享



7、分模块开发与设计

①：创建Maven模块



②：书写模块代码

注意事项

分模块开发需要先针对模块功能进行设计，再进行编码。不会先将工程开发完毕，然后进行拆分

③：通过maven指令安装模块到本地仓库（install指令）

注意事项

团队内部开发需要发布模块功能到团队内部可共享的仓库中（私服）

注意：记得加依赖

```
<dependency>
    <groupId>com.itheima</groupId>
    <artifactId>maven_03_pojo</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```



1. 分模块开发意义

2. 分模块开发（模块拆分）

↓

8、依赖传递

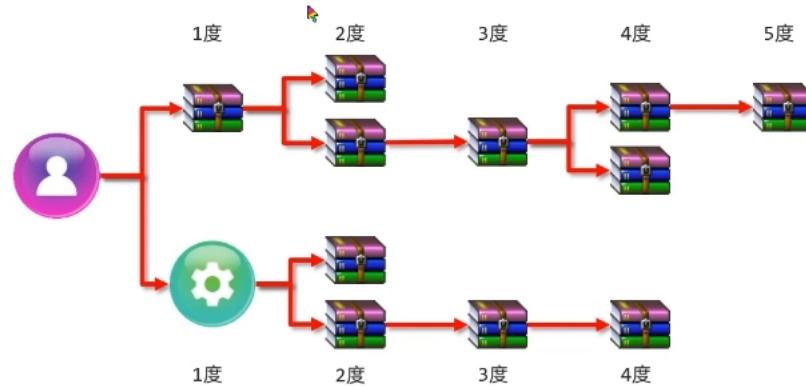
依赖管理

- 依赖指当前项目运行所需的jar，一个项目可以设置多个依赖
- 格式：

```
<!-- 设置当前项目所依赖的所有jar-->
<dependencies>
    <!--设置具体的依赖-->
    <dependency>
        <!--依赖所属群组id-->
        <groupId>org.springframework</groupId>
        <!--依赖所属项目id-->
        <artifactId>spring-webmvc</artifactId>
        <!--依赖版本号-->
        <version>5.2.10.RELEASE</version>
    </dependency>
</dependencies>
```

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时，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



9、可选依赖与排除依赖

可选依赖

- 可选依赖指对外隐藏当前所依赖的资源——不透明

```
<dependency>
    <groupId>com.itheima</groupId>
    <artifactId>maven_03_pojo</artifactId>
    <version>1.0-SNAPSHOT</version>
    <!-- 可选依赖是隐藏当前工程所依赖的资源，隐藏后对应资源将不具有依赖传递性-->
    <optional>false</optional>
</dependency>
```

排除依赖

- 排除依赖指主动断开依赖的资源，被排除的资源无需指定版本——不需要

```
<dependency>
    <groupId>com.itheima</groupId>
    <artifactId>maven_04_dao</artifactId>
    <version>1.0-SNAPSHOT</version>
    <!-- 排除依赖是隐藏当前资源对应的依赖关系-->
    <exclusions>
        <exclusion>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

- 排除依赖资源仅指定GA即可，无需指定V



1. 依赖管理
2. 依赖传递
3. 可选依赖（不透明）
4. 排除依赖（不需要）

10、聚合

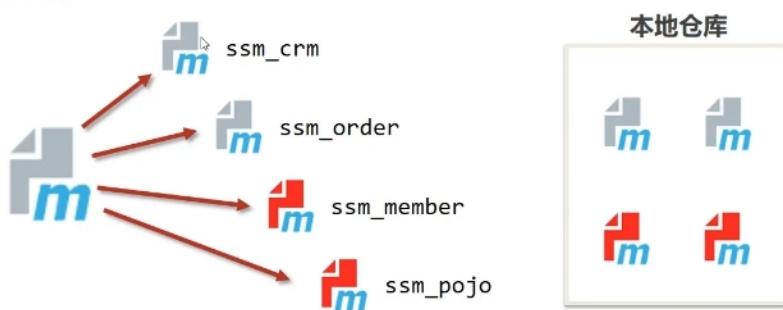


继承与聚合

- 聚合
- 继承

聚合

- 聚合：将多个模块组织成一个整体，同时进行项目构建的过程称为聚合
- 聚合工程：通常是一个不具有业务功能的“空”工程（有且仅有一个pom文件）
- 作用：使用聚合工程可以将多个工程编组，通过对聚合工程进行构建，实现对所包含的模块进行同步构建
 - 当工程中某个模块发生更新（变更）时，必须保障工程中与已更新模块关联的模块同步更新，此时可以使用聚合工程来解决批量模块同步构建的问题



exp:

```
<?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.itheima</groupId>
    <artifactId>maven_01_parent</artifactId>
    <version>1.0-RELEASE</version>
    <packaging>pom</packaging>

    <!-- 设置管理的模块名称-->
    <modules>
        <module>../maven_02_ssm</module>
        <module>../maven_03_pojo</module>
        <module>../maven_04_dao</module>
    </modules>

</project>
```

这就是聚合工程的pom文件

11、继承

继承

The image shows three separate dependency trees, each represented by a blue icon with a white 'm' and a yellow rounded rectangle containing XML code. The first tree is for 'ssm.crm', the second for 'ssm.goods', and the third for 'ssm.order'. Each tree contains three dependency blocks, all pointing to 'org.springframework' version '5.2.10.RELEASE'.

```
ssm.crm dependencies:
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.10.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.10.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.2.10.RELEASE</version>
</dependency>

ssm.goods dependencies:
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.10.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.10.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>5.2.10.RELEASE</version>
</dependency>

ssm.order dependencies:
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.10.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.2.10.RELEASE</version>
</dependency>

<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.16</version>
</dependency>
```

继承

- 概念：继承描述的是两个工程间的关系，与java中的继承相似，子工程可以继承父工程中的配置信息，常见于依赖关系的继承
- 作用：
 - 简化配置
 - 减少版本冲突

①：创建Maven模块，设置打包类型为pom

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

注意事项

建议父工程打包方式设置为pom

②：在父工程的pom文件中配置依赖关系（子工程将沿用父工程中的依赖关系）

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.2.10.RELEASE</version>
  </dependency>
  .....
</dependencies>
```

③：配置子工程中可选的依赖关系

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.16</version>
        </dependency>
        .....
    </dependencies>
</dependencyManagement>
```

④：在子工程中配置当前工程所继承的父工程

```
<!-- 定义该工程的父工程-->
<parent>
    <groupId>com.itheima</groupId>
    <artifactId>maven_parent</artifactId>
    <version>1.0-SNAPSHOT</version>
    <!-- 填写父工程的pom文件-->
    <relativePath>../maven_parent/pom.xml</relativePath>
</parent>
```

⑤：在子工程中配置使用父工程中可选依赖的坐标

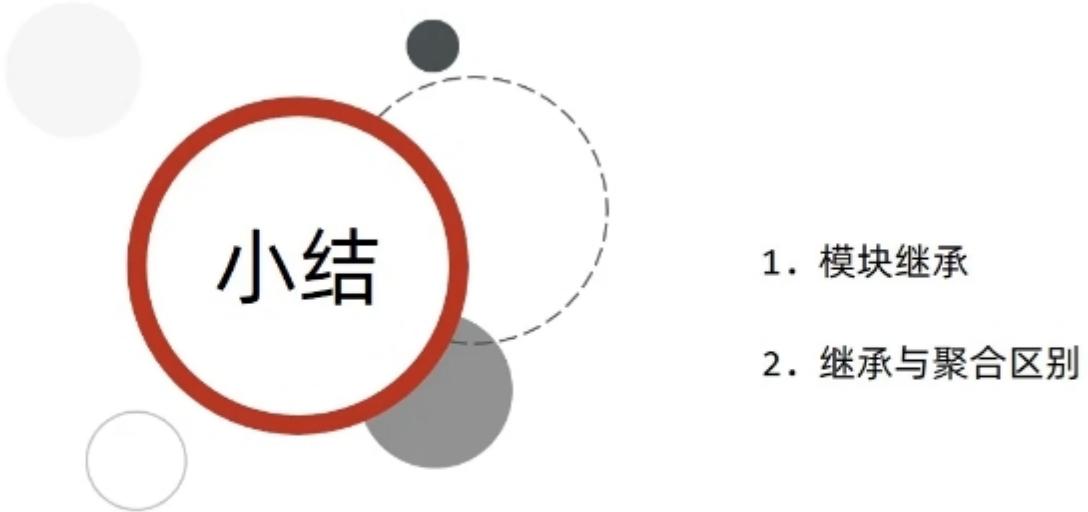
```
<dependencies>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
    </dependency>
</dependencies>
```

注意事项

子工程中使用父工程中的可选依赖时，仅需要提供群组id和项目id，无需提供版本，版本由父工程统一提供，避免版本冲突
子工程中还可以定义父工程中没有定义的依赖关系

聚合与继承的区别

- 作用
 - 聚合用于快速构建项目
 - 继承用于快速配置
- 相同点：
 - 聚合与继承的pom.xml文件打包方式均为pom，可以将两种关系制作到同一个pom文件中
 - 聚合与继承均属于设计型模块，并无实际的模块内容
- 不同点：
 - 聚合是在当前模块中配置关系，聚合可以感知到参与聚合的模块有哪些
 - 继承是在子模块中配置关系，父模块无法感知哪些子模块继承了自己



总结

1. 聚合

2. 继承

12、属性

属性

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>【spring_version】</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>【spring_version】</version>
</dependency>

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-test</artifactId>
    <version>【spring_version】</version>
</dependency>
```

String spring_version = "5.2.10.RELEASE";

①：定义属性

```
<!-- 定义自定义属性-->
<properties>
    <spring.version>5.2.10.RELEASE</spring.version>
        <junit.version>4.12</junit.version>
</properties>
```

②：引用属性

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

13、配置文件加载属性

①：定义属性

```
<!-- 定义自定义属性-->
<properties>
    <spring.version>5.2.10.RELEASE</spring.version>
    <junit.version>4.12</junit.version>
    <jdbc.url>jdbc:mysql://127.0.0.1:3306/ssm_db</jdbc.url>
</properties>
```

②：配置文件中引用属性

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=${jdbc.url}
jdbc.username=root
jdbc.password=root
```

3、开启资源文件目录加载属性的过滤器

```
<build>
    <resources>
        <!-- 设置资源目录，并设置能够解析${}-->
        <resource>
            <directory>${project.basedir}/src/main/resources</directory>
            <filtering>true</filtering>
        </resource>
    </resources>
</build>
```

④：配置maven打war包时，忽略web.xml检查

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-war-plugin</artifactId>
<version>3.2.3</version>
<configuration>
<failOnMissingWebXml>false</failOnMissingWebXml>
</configuration>
</plugin>
```

exp:

```
<build>
<plugins>
<plugin>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.12.4</version>
<configuration>
<skipTests>false</skipTests>
<!--排除掉不参与测试的内容-->
<excludes>
<exclude>**/BookServiceTest.java</exclude>
</excludes>
</configuration>
</plugin>
</plugins>
</build>
```

其他属性（了解）

属性分类	引用格式	示例
自定义属性	<code> \${自定义属性名}</code>	<code> \${spring.version}</code>
内置属性	<code> \${内置属性名}</code>	<code> \${basedir} \${version}</code>
Setting属性	<code> \${setting.属性名}</code>	<code> \${settings.localRepository}</code>
Java系统属性	<code> \${系统属性分类.系统属性名}</code>	<code> \${user.home}</code>
环境变量属性	<code> \${env.环境变量属性名}</code>	<code> \${env.JAVA_HOME}</code>

小结

1. 自定义属性

2. 其他属性（了解）

14、版本管理

版本管理

	spring-context	junit	mysql-jdbc
	5.2.1.RELEASE	4.13-rc-2	6.0.6
	5.2.0.RELEASE	4.13-rc-1	6.0.5
	5.1.20.RELEASE	4.13-beta-3	6.0.4
	5.1.19.RELEASE	4.13-beta-2	6.0.3
	5.1.18.RELEASE	4.13-beta-1	6.0.2
	5.1.17.RELEASE	4.12	5.1.49
	5.1.16.RELEASE	4.12-beta-3	5.1.48
	5.1.15.RELEASE	4.12-beta-2	5.1.47
	5.1.14.RELEASE	4.12-beta-1	5.1.46
	5.1.13.RELEASE	4.11	5.1.45
	5.1.12.RELEASE	4.11-beta-1	5.1.44
	5.1.11.RELEASE	4.10	5.1.43
5.1.x	5.1.10.RELEASE	4.9	5.1.42
	5.1.9.RELEASE	4.8.2	5.1.41
	5.1.8.RELEASE	4.8.1	5.1.40

版本管理

- 工程版本：
 - SNAPSHOT（快照版本）
 - ◆ 项目开发过程中临时输出的版本，称为快照版本
 - ◆ 快照版本会随着开发的进展不断更新
 - RELEASE（发布版本）
 - ◆ 项目开发到进入阶段里程碑后，向团队外部发布较为稳定的版本，这种版本所对应的构件文件是稳定的，即便进行功能的后续开发，也不会改变当前发布版本内容，这种版本称为发布版本
- 发布版本
 - alpha版
 - beta版
 - 纯数字版

小结

1. SNAPSHOT
2. RELEASE

总结

1. 属性
2. 版本管理

15、多环境开发

①：定义多环境

```
<!-- 定义多环境-->
<profiles>
    <!-- 定义具体的环境：生产环境-->
    <profile>
        <!-- 定义环境对应的唯一名称-->
        <id>env_dep</id>
        <!-- 定义环境中专用的属性值-->
        <properties>
            <jdbc.url>jdbc:mysql://127.0.0.1:3306/ssm_db</jdbc.url>
        </properties>
        <!-- 设置默认启动-->
        <activation>
            <activeByDefault>true</activeByDefault>
        </activation>
    </profile>
    <!-- 定义具体的环境：开发环境-->
    <profile>
        <id>env_pro</id>
        .....
    </profile>
</profiles>
```

②：使用多环境（构建过程）

```
mvn 指令 -P 环境定义id
```

范例：

```
mvn install -P pro_env
```



1. 多环境开发

16、跳过测试

跳过测试

- 跳过测试

```
mvn 指令 -D skipTests
```

- 范例：

```
mvn install -D skipTests
```

注意事项

执行的项目构建指令必须包含测试生命周期，否则无效果。例如执行compile生命周期，不经过test生命周期

跳过测试

- 细粒度控制跳过测试

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.22.1</version>
  <configuration>
    <skipTests>true</skipTests><!-- 设置跳过测试-->
    <includes> <!-- 包含指定的测试用例-->
      <include>**/*User*Test.java</include>
    </includes>
    <excludes><!-- 排除指定的测试用例-->
      <exclude>**/*User*TestCase.java</exclude>
    </excludes>
  </configuration>
</plugin>
```



1. 跳过测试（了解）

总结

1. 多环境配置: profile
2. 跳过测试 (了解)

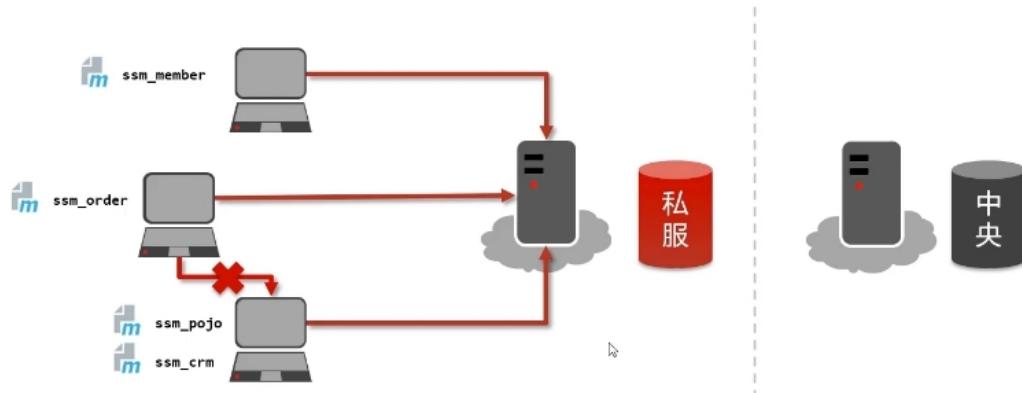
17、私服简介与安装



私服

- 私服简介
- 私服仓库分类
- 资源上传与下载

团队开发现状分析



私服简介

- 私服是一台独立的服务器，用于解决团队内部的资源共享与资源同步问题
- Nexus
 - Sonatype公司的一款maven私服产品
 - 下载地址：<https://help.sonatype.com/repomanager3/download>

启动位置



Nexus安装与启动

- 启动服务器（命令行启动）

```
nexus.exe /run nexus
```

- 访问服务器（默认端口：8081）

```
http://localhost:8081
```

- 修改基础配置信息
 - 安装路径下etc目录中nexus-default.properties文件保存有nexus基础配置信息，例如默认访问端口

- 修改服务器运行配置信息
 - 安装路径下bin目录中nexus.vmoptions文件保存有nexus服务器启动对应的配置信息，例如默认占用内存空间

账号：admin

密码：admin

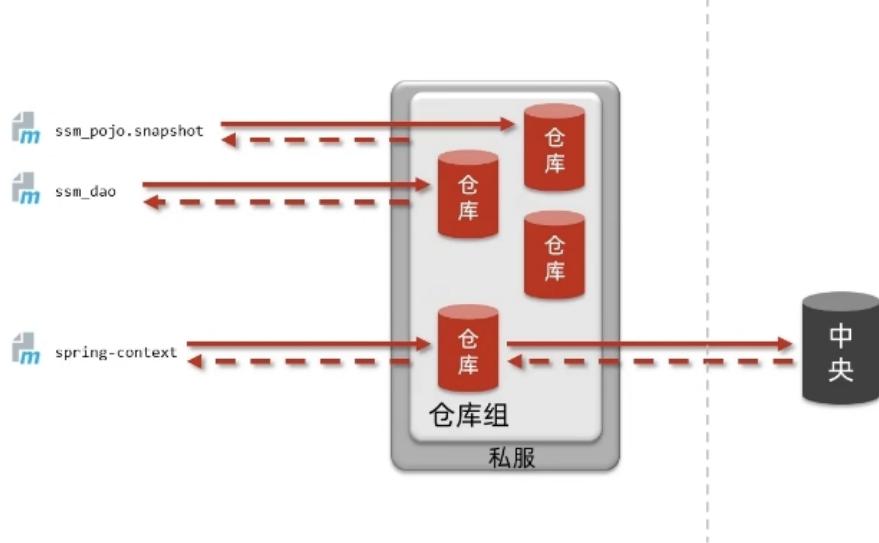


1. 私服作用

2. Nexus

18、私服仓库分类

私服资源操作流程分析



私服仓库分类

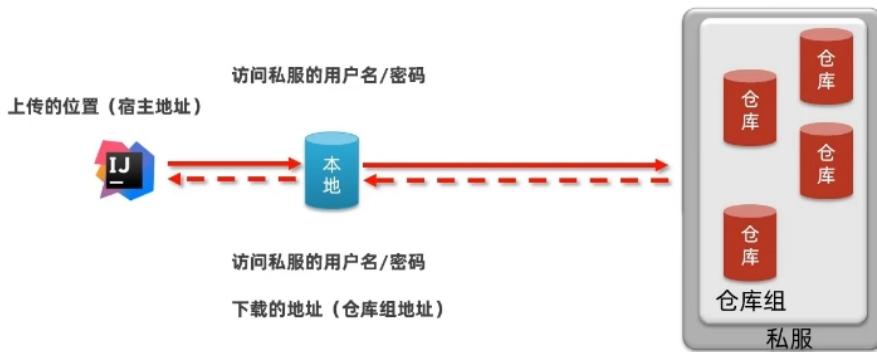
仓库类别	英文名称	功能	关联操作
宿主仓库	hosted	保存自主研发+第三方资源	上传
代理仓库	proxy	代理连接中央仓库	下载
仓库组	group	为仓库编组简化下载操作	下载

小结

1. 仓库分类

19、本地仓库访问私服配置

资源上传与下载



修改setting文件

The screenshot shows the Windows File Explorer interface. The path is '此电脑 > 本地磁盘 (D:) > soft > apache-maven-3.6.1 > conf'. A file named 'settings.xml' is selected. Below the file list, the XML content of 'settings.xml' is displayed:

```
<!--配置访问字符的权限-->
<servers>
    <id>私服中的服务端Id名称</id>
    <username>repouser</username>
    <password>repopwd</password>
</servers>
```

At the bottom, there is a table mapping repository names to their details:

仓库名	类型	地址	状态
itheima-release	hosted	maven2	Online
itheima-snapshot	hosted	maven2	Online

Below the table, the XML content for the 'itheima-release' repository is shown:

```
<!--配置访问字符的权限-->
<servers>
    <id>itheima-release</id>
    <username>admin</username>
    <password>admin</password>
</servers>
```

映射关系

```

<mirrors>
    <!-- mirror
    | Specifies a repository mirror site to use instead of a given repository. The repository tha-
    | this mirror serves has an ID that matches the mirrorOf element of this mirror. IDs are used
    | for inheritance and direct lookup purposes, and must be unique across the set of mirrors.
    |
<mirror>
    <id>mirrorId</id>
    <mirrorOf>repositoryId</mirrorOf>
    <name>Human Readable Name for this Mirror.</name>
    <url>http://my.repository.com/repo/path</url>
</mirror>
-->
<!--私服的访问路径-->
<mirror>
    <id>mirrorId</id>
    <mirrorOf>repositoryId</mirrorOf>
    <url>http://my.repository.com/repo/path</url>
</mirror>
</mirrors>

```

20、私服资源上传与下载

工程上传到私服服务器设置

- 配置位置（工程pom文件中）

```

<distributionManagement>
    <repository>
        <id>heima-release</id>
        <url>http://localhost:8081/repository/heima-release/</url>
    </repository>
    <snapshotRepository>
        <id>heima-snapshots</id>
        <url>http://localhost:8081/repository/heima-snapshots/</url>
    </snapshotRepository>
</distributionManagement>

```

- 发布命令

```
mvn deploy
```

私服访问中央服务器设置

- 配置位置（nexus服务器页面设置）

The screenshot shows the Nexus Repository Manager interface. On the left, there's a sidebar with 'Administration' and 'Repositories' sections. Under 'Repositories', 'Repositories' is selected, and a list of repositories is shown: Blob Stores, maven-central (which is highlighted with a red box), maven-public, maven-releases, maven-snapshots, nuget-group, nuget-hosted, and nuget.org-proxy. On the right, the main panel shows the 'Repositories' settings for 'maven-central'. It includes fields for 'Remote storage' (set to 'https://repo1.maven.org/maven2/'), 'Use the Nexus truststore', and two buttons at the bottom: 'Save' and 'Discard'.

小结

1. 服务器配置
2. 资源上传与下载

总结

1. 私服简介
2. 私服仓库分类
3. 资源上传与下载