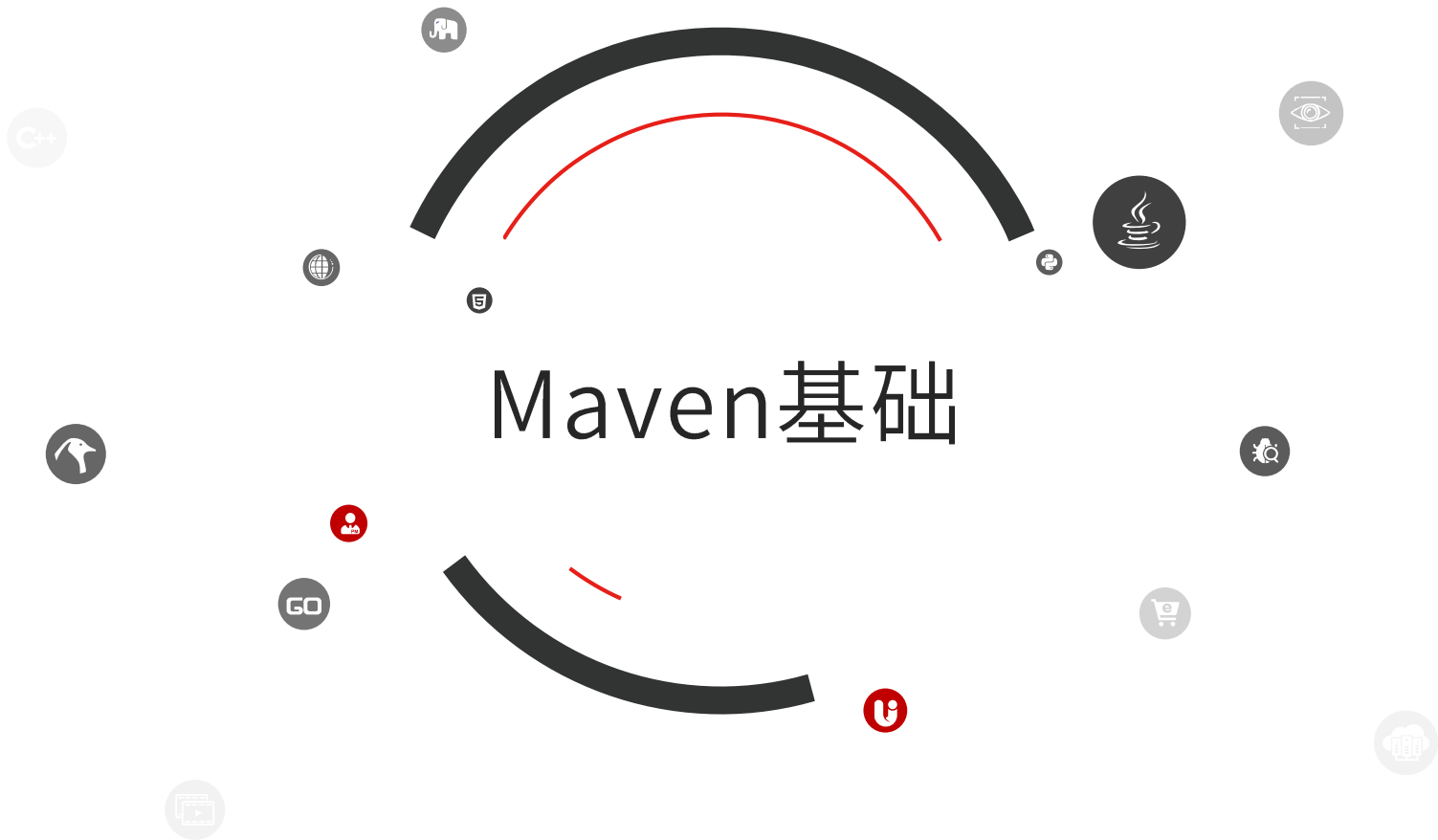



Maven基础





目录 Contents

- ◆ Maven简介
- ◆ 下载与安装
- ◆ Maven基础概念
- ◆ 第一个Maven项目（手工制作）
- ◆ 第一个Maven项目（IDEA生成）
- ◆ 依赖管理
- ◆ 生命周期与插件

目录 Contents

- ◆ Maven简介
- ◆ 下载与安装
- ◆ Maven基础概念 (重点)
- ◆ 第一个Maven项目 (手工制作) (重点)
- ◆ 第一个Maven项目 (IDEA生成) (重点)
- ◆ 依赖管理 (重点)
- ◆ 生命周期与插件

资料格式

- 配置文件

```
<groupId>com.itheima</groupId>
```

- Java代码

```
Statement stat = con.createStatement();
```

- 示例

```
<groupId>com.itheima</groupId>
```

- 命令

```
mvn test
```

目录 Contents

- ◆ Maven简介
- ◆ 下载与安装
- ◆ Maven基础概念
- ◆ 第一个Maven项目（手工制作）
- ◆ 第一个Maven项目（IDEA生成）
- ◆ 依赖管理
- ◆ 生命周期与插件

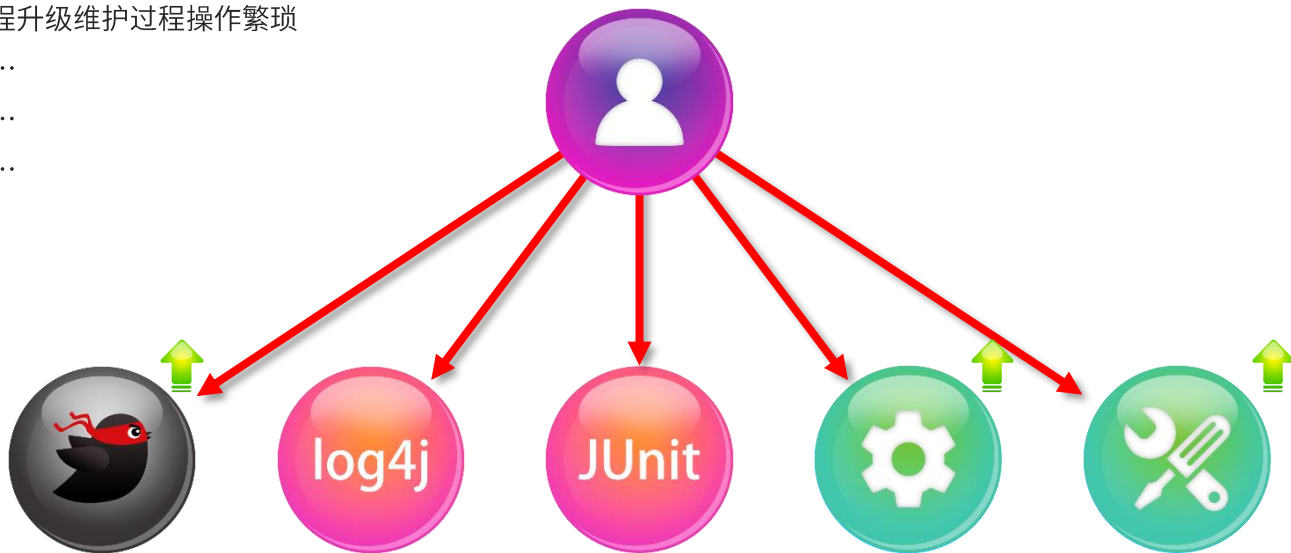
目录 Contents

◆ Maven简介

- ◆ Maven是什么
- ◆ Maven的作用

传统项目管理状态分析

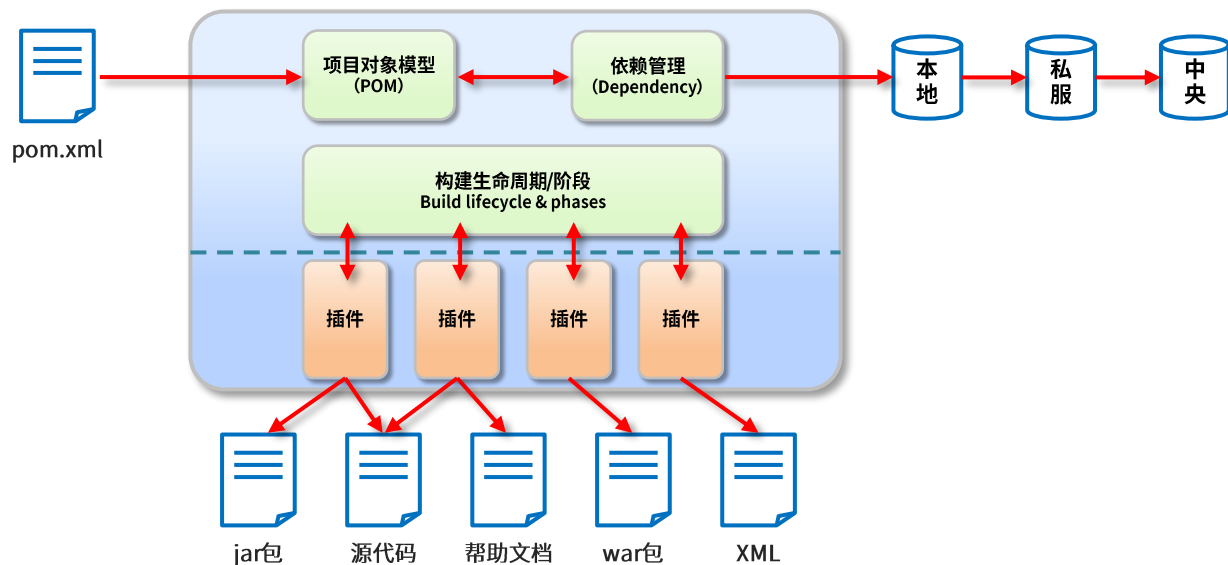
- jar包不统一，jar包不兼容
- 工程升级维护过程操作繁琐
-
-
-



Maven简介

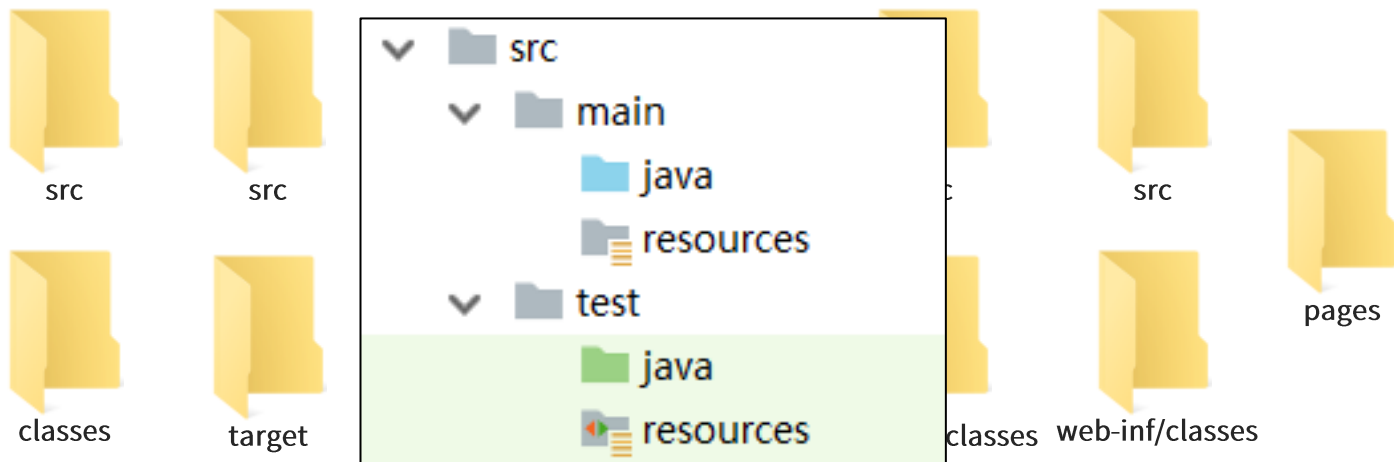
Maven是什么

- 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



Maven的作用

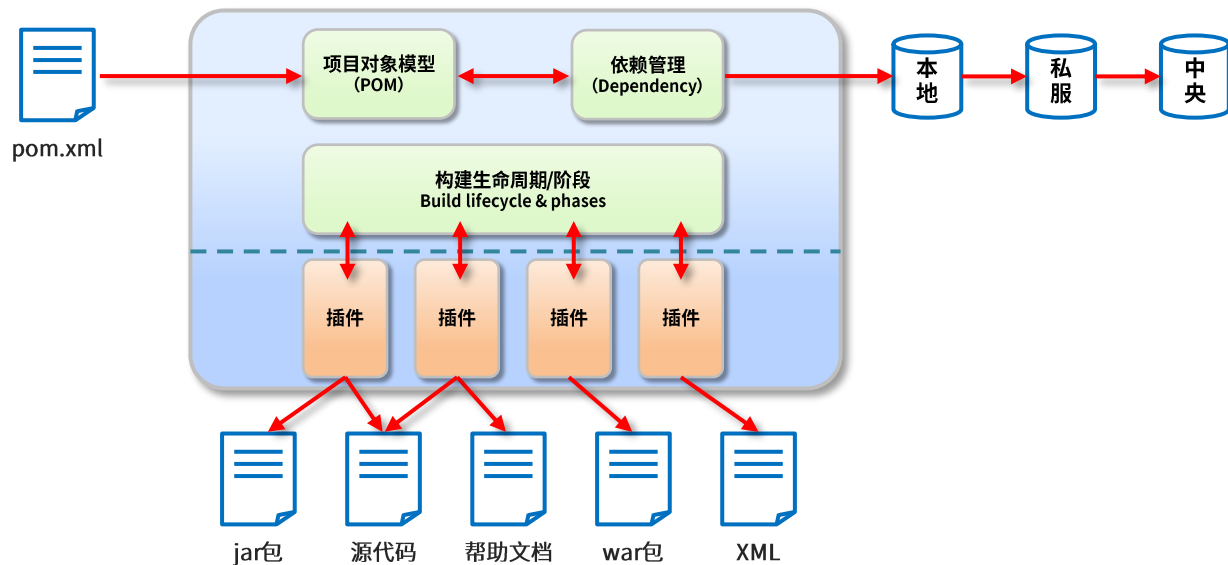
- 项目构建：提供标准的、跨平台的自动化项目构建方式
- 依赖管理：方便快捷的管理项目依赖的资源（jar包），避免资源间的版本冲突问题
- 统一开发结构：提供标准的、统一的项目结构



Maven简介

小节

- Maven是什么
- Maven的作用
- POM



目录 Contents

- ◆ Maven简介
- ◆ 下载与安装
- ◆ Maven基础概念
- ◆ 第一个Maven项目（手工制作）
- ◆ 第一个Maven项目（IDEA生成）
- ◆ 依赖管理
- ◆ 生命周期与插件

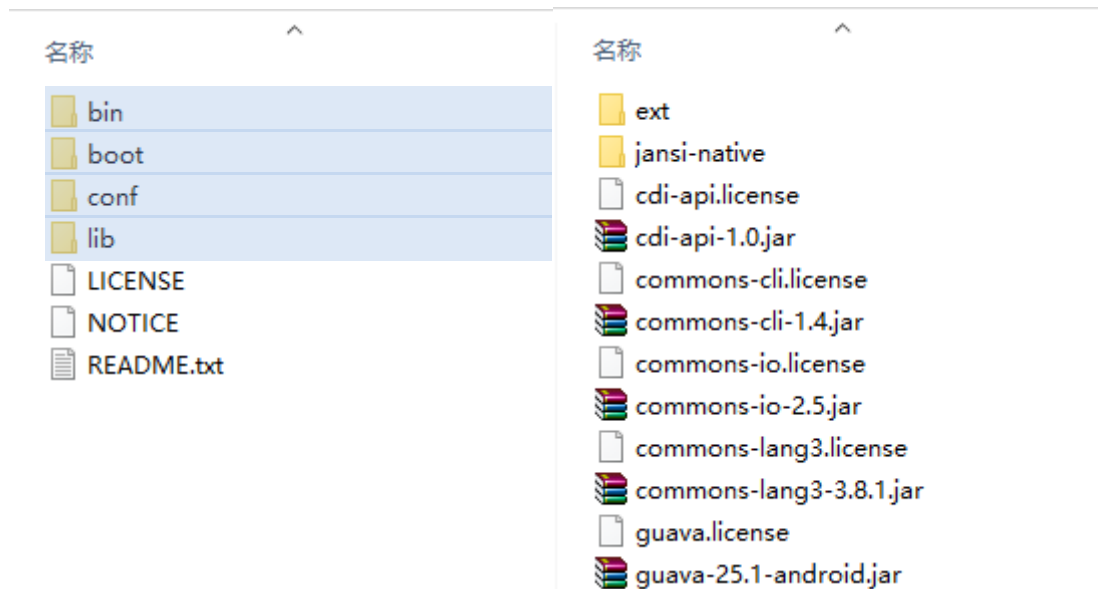
■ 下载与安装

Maven下载

- 官网: <http://maven.apache.org/>
- 下载地址: <http://maven.apache.org/download.cgi>

Maven安装

- Maven属于绿色版软件，解压即安装



Maven环境变量配置

- 依赖Java，需要配置JAVA_HOME
- 设置MAVEN自身的运行环境，需要配置MAVEN_HOME
- 测试环境配置结果

MVN



下载与安装

小节

- 下载与安装
- 环境变量配置

目录 Contents

- ◆ Maven简介
- ◆ 下载与安装
- ◆ Maven基础概念
- ◆ 第一个Maven项目（手工制作）
- ◆ 第一个Maven项目（IDEA生成）
- ◆ 依赖管理
- ◆ 生命周期与插件

Contents

目录

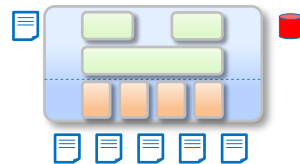
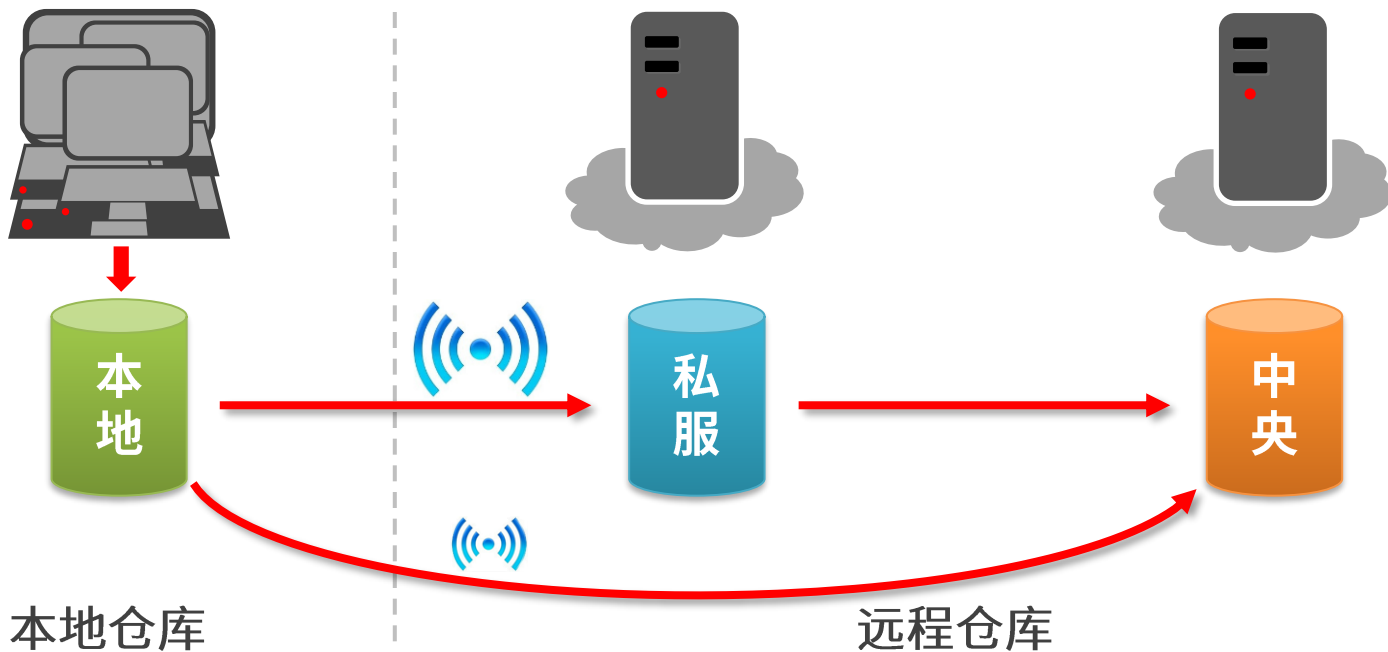
◆ Maven基础概念

- ◆ 仓库
- ◆ 坐标

Maven基础概念

仓库

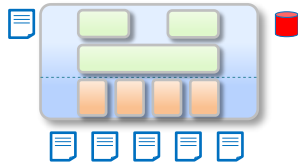
- 仓库：用于存储资源，包含各种jar包



Maven基础概念

仓库

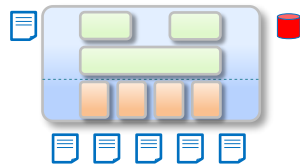
- 仓库：用于存储资源，包含各种jar包
- 仓库分类：
 - ◆ 本地仓库：自己电脑上存储资源的仓库，连接远程仓库获取资源
 - ◆ 远程仓库：非本机电脑上的仓库，为本地仓库提供资源
 - 中央仓库：Maven团队维护，存储所有资源的仓库
 - 私服：部门/公司范围内存储资源的仓库，从中央仓库获取资源
- 私服的作用：
 - ◆ 保存具有版权的资源，包含购买或自主研发的jar
 - 中央仓库中的jar都是开源的，不能存储具有版权的资源
 - ◆ 一定范围内共享资源，仅对内部开放，不对外共享



Maven基础概念

小节

- 仓库的概念与作用
- 仓库的分类
 - ◆ 本地仓库
 - ◆ 远程仓库
 - 中央仓库
 - 私服

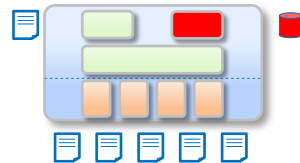


Maven基础概念

坐标



Jock的快递



Maven基础概念

坐标

- 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/>

- 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)

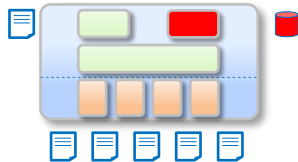
artifactId: 定义当前Maven项目名称 (通常是模块名称, 例如CRM、SMS)

version: 定义当前项目版本号

packaging: 定义该项目的打包方式

- Maven坐标的作用

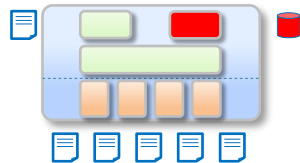
使用唯一标识, 唯一性定位资源位置, 通过该标识可以将资源的识别与下载工作交由机器完成



Maven基础概念

小节

- 坐标的概念与作用
- 坐标的组成
 - ◆ 组织ID
 - ◆ 项目ID
 - ◆ 版本号



Maven基础概念

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

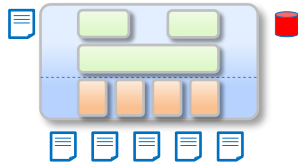
```
<localRepository>${user.home}/.m2/repository</localRepository>
```

当前目录位置为登录用户名所在目录下的.m2文件夹中

- ◆ 自定义位置

```
<localRepository>D:\maven\repository</localRepository>
```

当前目录位置为D:\maven\repository文件夹中

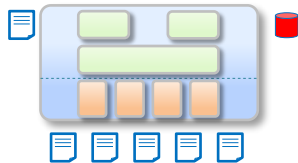


Maven基础概念

远程仓库配置

- Maven默认连接的仓库位置

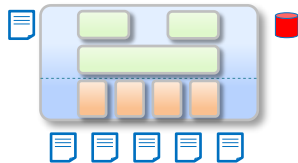
```
<repositories>
  <repository>
    <id>central</id>
    <name>Central Repository</name>
    <url>https://repo.maven.apache.org/maven2</url>
    <layout>default</layout>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </repository>
</repositories>
```



Maven基础概念

镜像仓库配置

- 在setting文件中配置阿里云镜像仓库

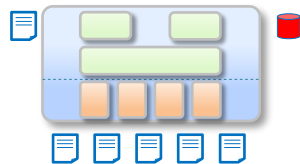


```
<mirrors>
  <!--配置具体的仓库的下载镜像 -->
  <mirror>
    <!-- 此镜像的唯一标识符，用来区分不同的mirror元素 -->
    <id>nexus-aliyun</id>
    <!-- 对哪种仓库进行镜像，简单说就是替代哪个仓库 -->
    <mirrorOf>central</mirrorOf>
    <!-- 镜像名称 -->
    <name>Nexus aliyun</name>
    <!-- 镜像URL -->
    <url>http://maven.aliyun.com/nexus/content/groups/public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

■ Maven基础概念

全局setting与用户setting区别

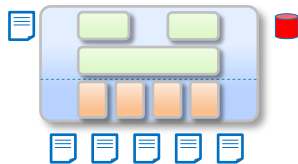
- 全局setting定义了当前计算机中Maven的公共配置
- 用户setting定义了当前用户的配置



■ Maven基础概念

小节

- 配置本地仓库（资源下到哪）
- 配置阿里镜像仓库（资源从哪来）
- setting文件的区别



目录 Contents

- ◆ Maven简介
- ◆ 下载与安装
- ◆ Maven基础概念
- ◆ 第一个Maven项目（手工制作）
- ◆ 第一个Maven项目（IDEA生成）
- ◆ 依赖管理
- ◆ 生命周期与插件

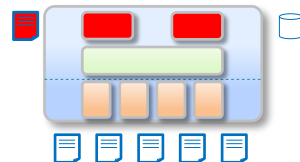
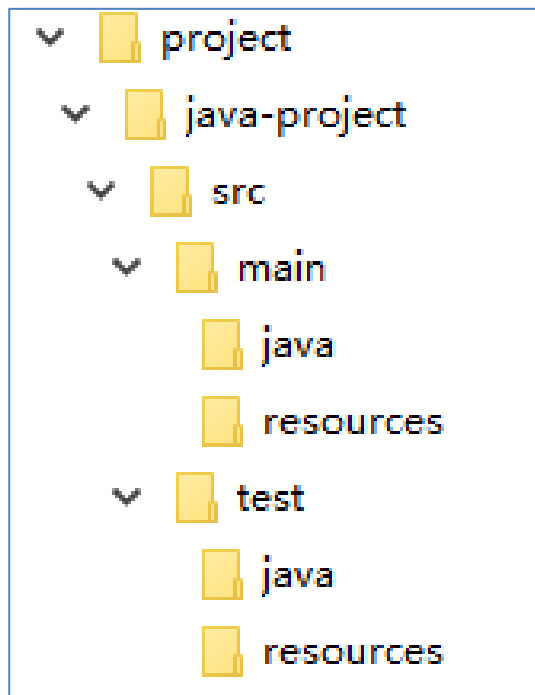
目 录 Contents

◆ 第一个Maven项目（手工制作）

- ◆ Maven工程目录结构
- ◆ 构建命令
- ◆ 插件创建工程

■ 第一个Maven项目（手工制作）

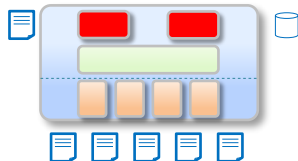
Maven工程目录结构



■ 第一个Maven项目（手工制作）

Maven工程目录结构

- 在src同层目录下创建pom.xml



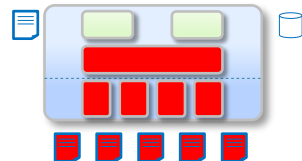
```
<?xml version="1.0" encoding="UTF-8"?>
<project
  xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.itheima</groupId>
  <artifactId>project-java</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
    </dependency>
  </dependencies>
</project>
```


■ 第一个Maven项目（手工制作）

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

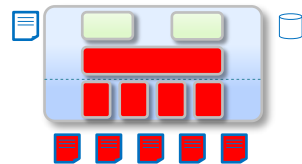
<code>mvn compile</code>	#编译
<code>mvn clean</code>	#清理
<code>mvn test</code>	#测试
<code>mvn package</code>	#打包
<code>mvn install</code>	#安装到本地仓库



■ 第一个Maven项目（手工制作）

小节

- Maven工程目录结构
- Maven常用项目构建指令



■ 第一个Maven项目（手工制作）

插件创建工程

- 创建工程

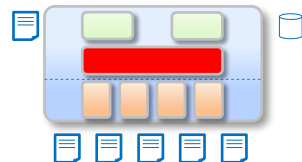
```
mvn archetype:generate
    -DgroupId={project-packaging}
    -DartifactId={project-name}
    -DarchetypeArtifactId=maven-archetype-quickstart
    -DinteractiveMode=false
```

- 创建java工程

```
mvn archetype:generate -DgroupId=com.itheima -DartifactId=java-project -
DarchetypeArtifactId=maven-archetype-quickstart -Dversion=0.0.1-snapshot -
DinteractiveMode=false
```

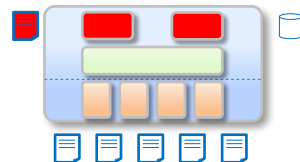
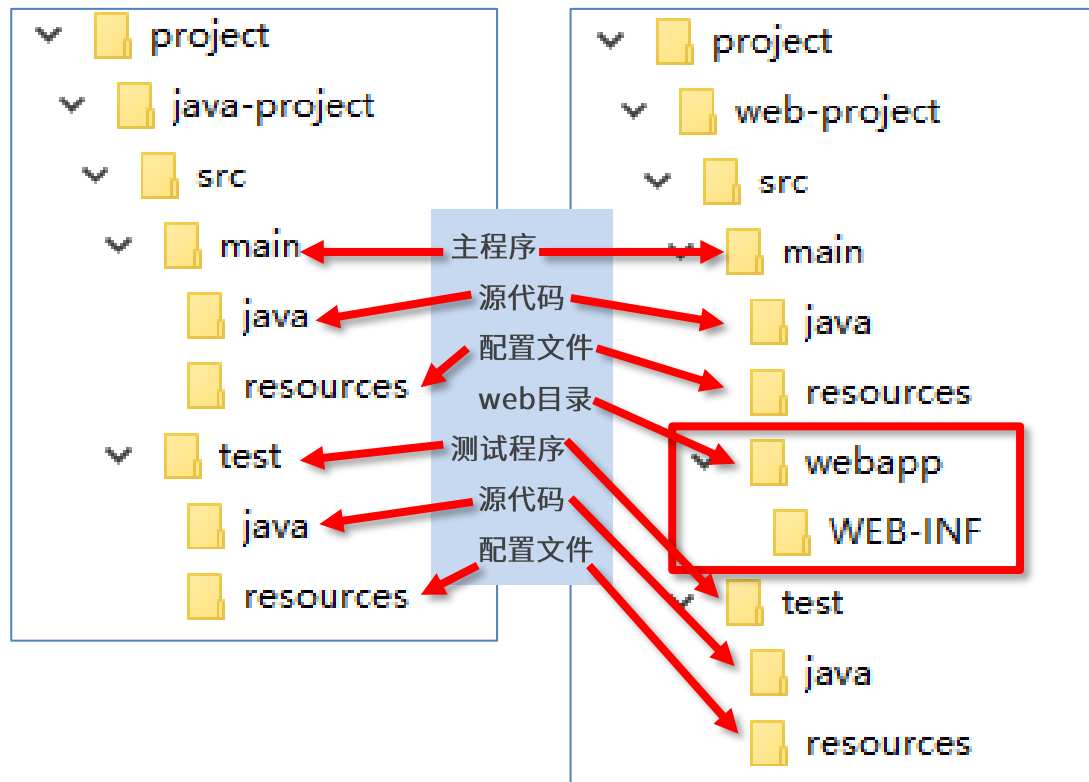
- 创建web工程

```
mvn archetype:generate -DgroupId=com.itheima -DartifactId=web-project -
DarchetypeArtifactId=maven-archetype-webapp -Dversion=0.0.1-snapshot -
DinteractiveMode=false
```



■ 第一个Maven项目（手工制作）

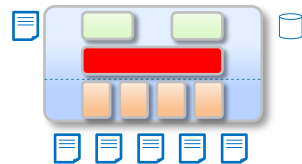
Maven工程目录结构



■ 第一个Maven项目（手工制作）

小节

- 插件创建Maven工程
- Maven web工程目录结构



目录 Contents

- ◆ Maven简介
- ◆ 下载与安装
- ◆ Maven基础概念
- ◆ 第一个Maven项目（手工制作）
- ◆ 第一个Maven项目（IDEA生成）
- ◆ 依赖管理
- ◆ 生命周期与插件

目录 Contents

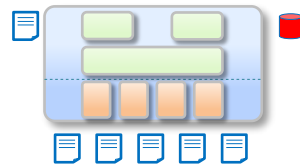
◆ 第一个Maven项目（IDEA生成）

- ◆ 配置Maven
- ◆ 手工创建Java项目
- ◆ 原型创建Java项目
- ◆ 原型创建Web项目
- ◆ 插件

■ 第一个Maven项目（IDEA生成）

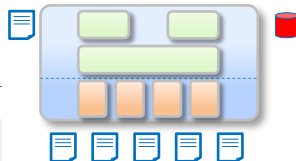
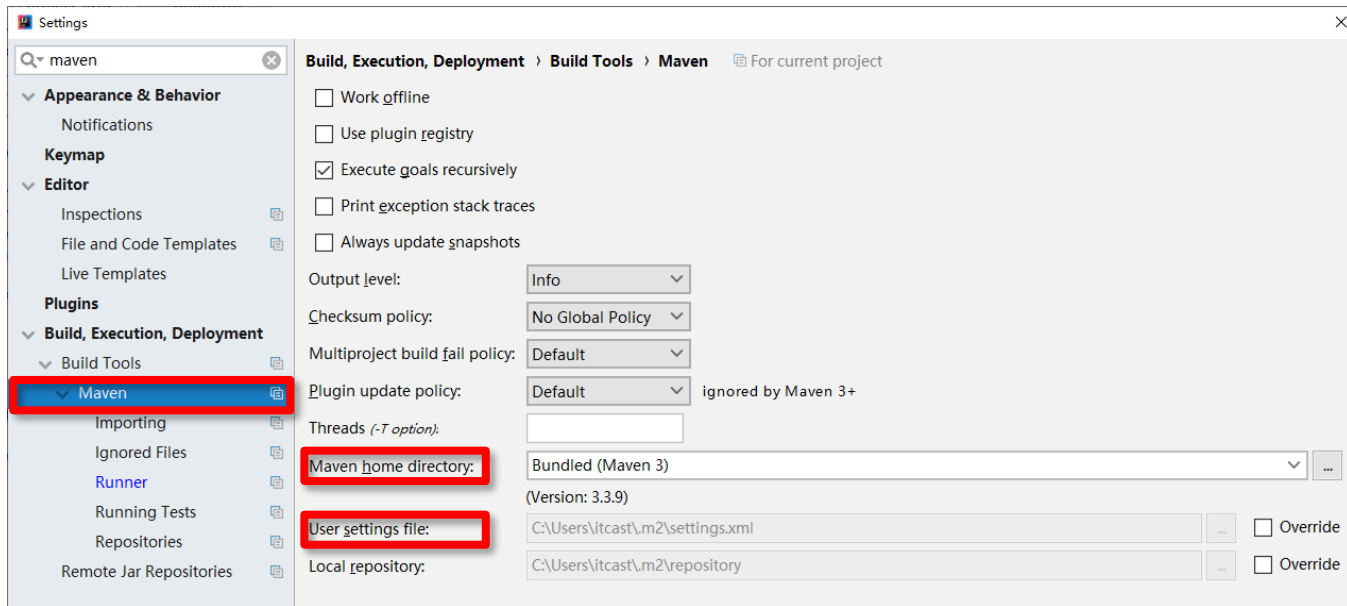
配置Maven

- Idea对3.6.2及以上版本存在兼容性问题，为避免冲突，IDEA中安装使用3.6.1版本



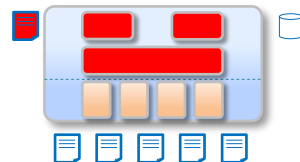
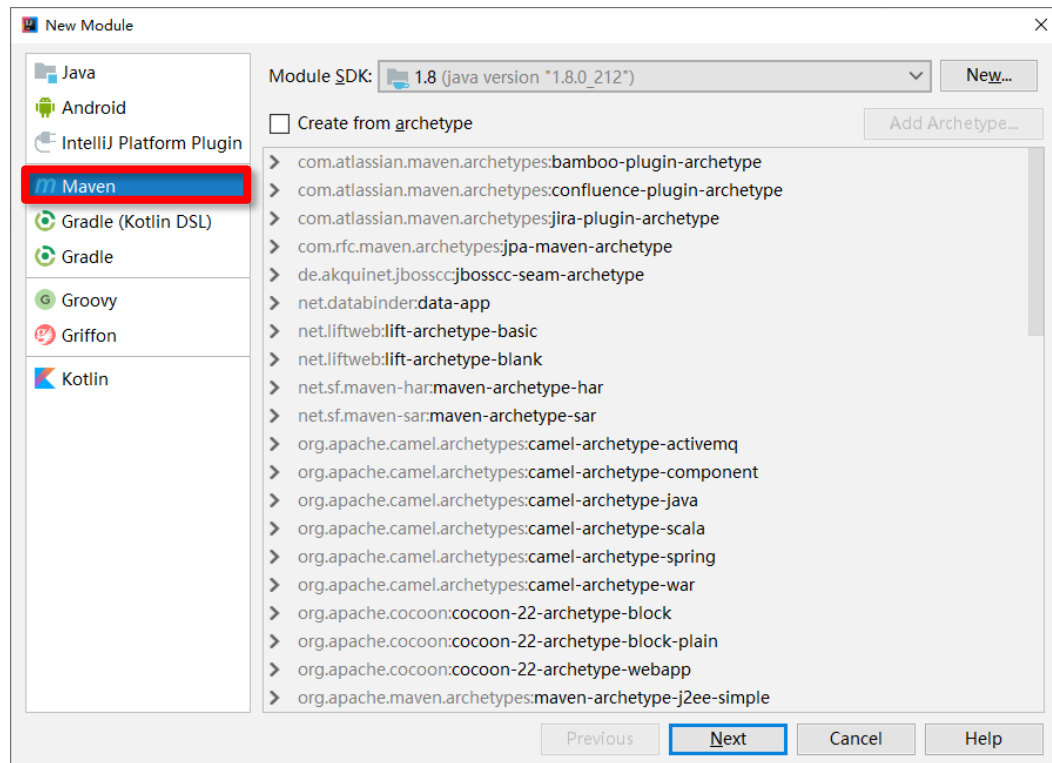
■ 第一个Maven项目 (IDEA生成)

配置Maven



■ 第一个Maven项目 (IDEA生成)

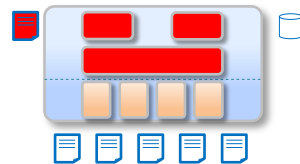
手工创建Java项目



■ 第一个Maven项目（IDEA生成）

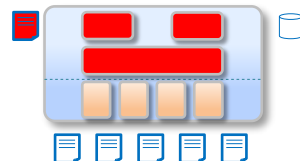
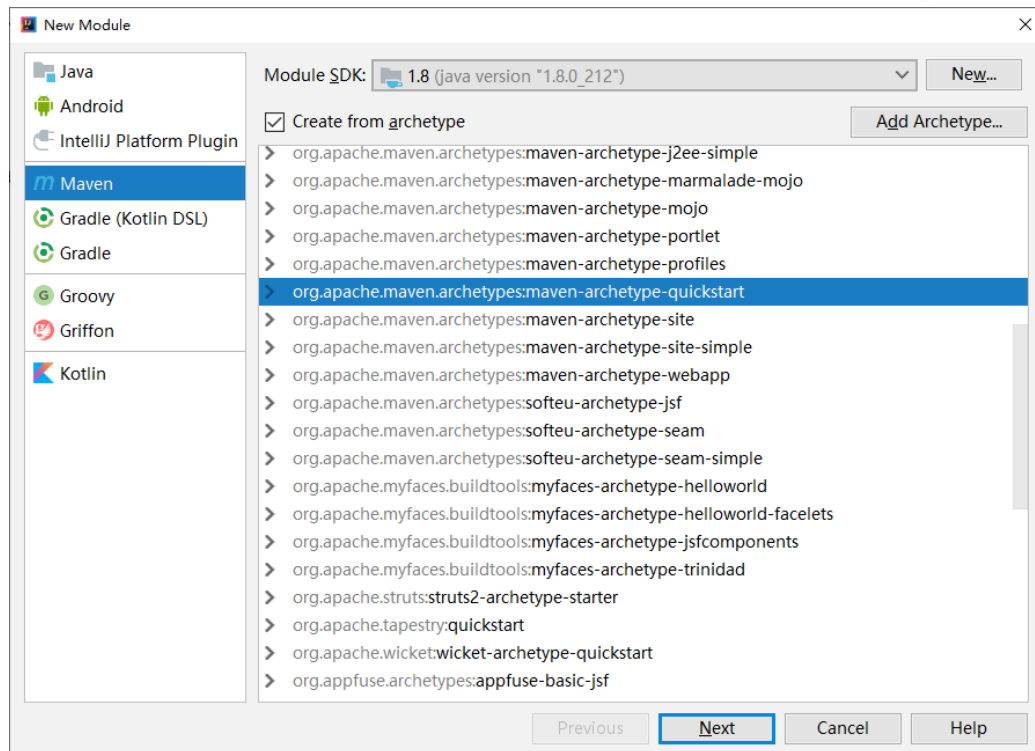
小节

- Maven环境配置
- Maven项目创建
- Maven命令执行



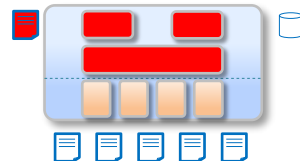
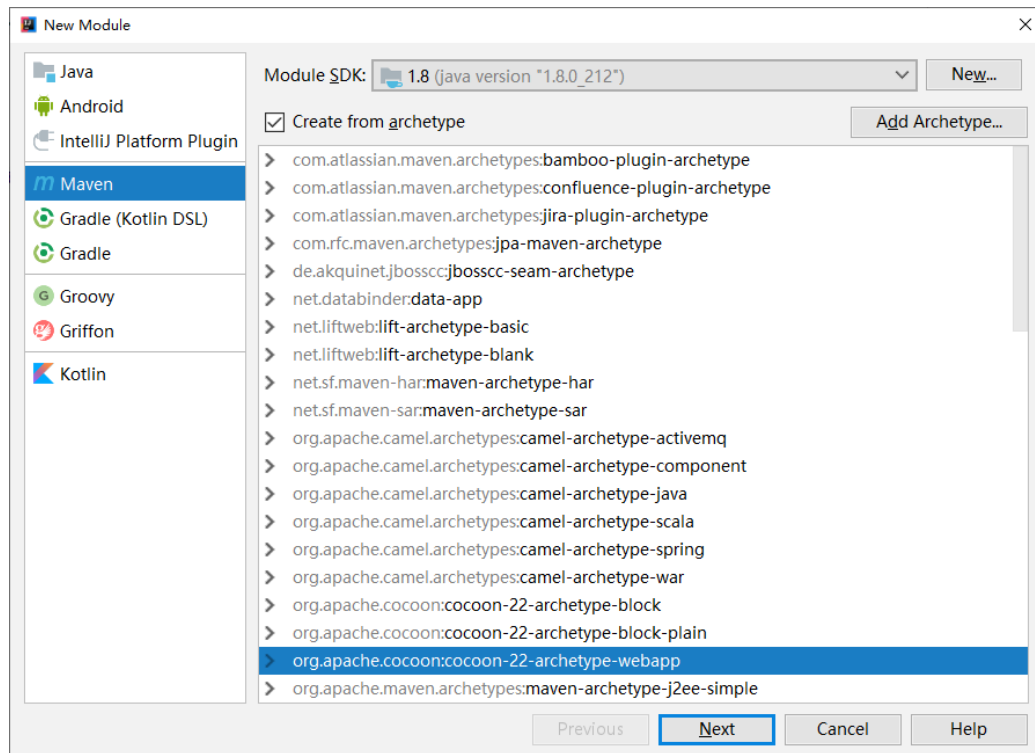
■ 第一个Maven项目 (IDEA生成)

原型创建Java项目



■ 第一个Maven项目 (IDEA生成)

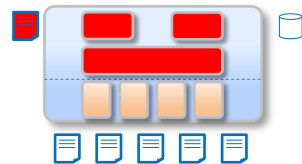
原型创建Web项目



■ 第一个Maven项目（IDEA生成）

小节

- 使用原型创建java项目
- 使用原型创建web项目

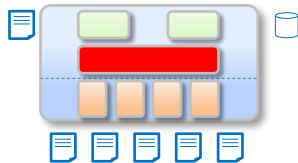


■ 第一个Maven项目（IDEA生成）

插件

- Tomcat7运行插件

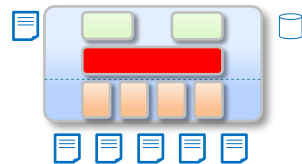
```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.tomcat.maven</groupId>
      <artifactId>tomcat7-maven-plugin</artifactId>
      <version>2.1</version>
      <configuration>
        <port>80</port>
        <path>/</path>
      </configuration>
    </plugin>
  </plugins>
</build>
```




■ 第一个Maven项目（IDEA生成）

小节

- tomcat7插件安装
- 运行web项目





目录 Contents

- ◆ Maven简介
- ◆ 下载与安装
- ◆ Maven基础概念
- ◆ 第一个Maven项目（手工制作）
- ◆ 第一个Maven项目（IDEA生成）
- ◆ 依赖管理
- ◆ 生命周期与插件

目录 Contents

◆ 依赖管理

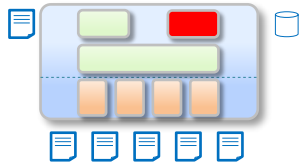
- ◆ 依赖配置
- ◆ 依赖传递
- ◆ 可选依赖
- ◆ 排除依赖
- ◆ 依赖范围

■ 依赖管理

依赖配置

- 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>junit</groupId>
    <!--依赖所属项目id-->
    <artifactId>junit</artifactId>
    <!--依赖版本号-->
    <version>4.12</version>
  </dependency>
</dependencies>
```

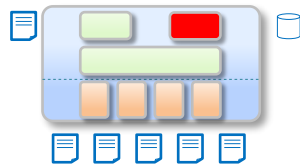
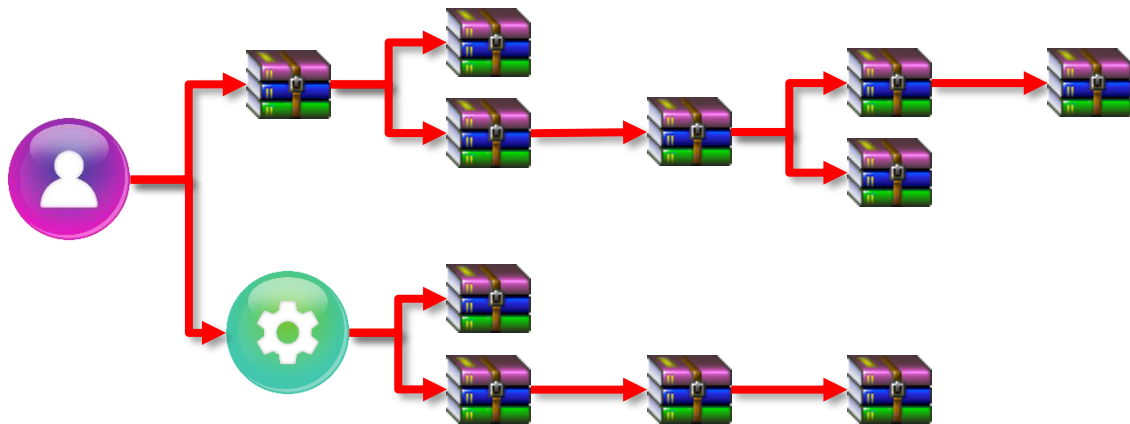


■ 依赖管理

依赖传递

- 依赖具有传递性

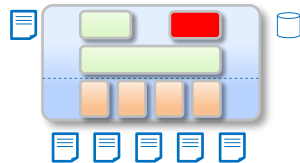
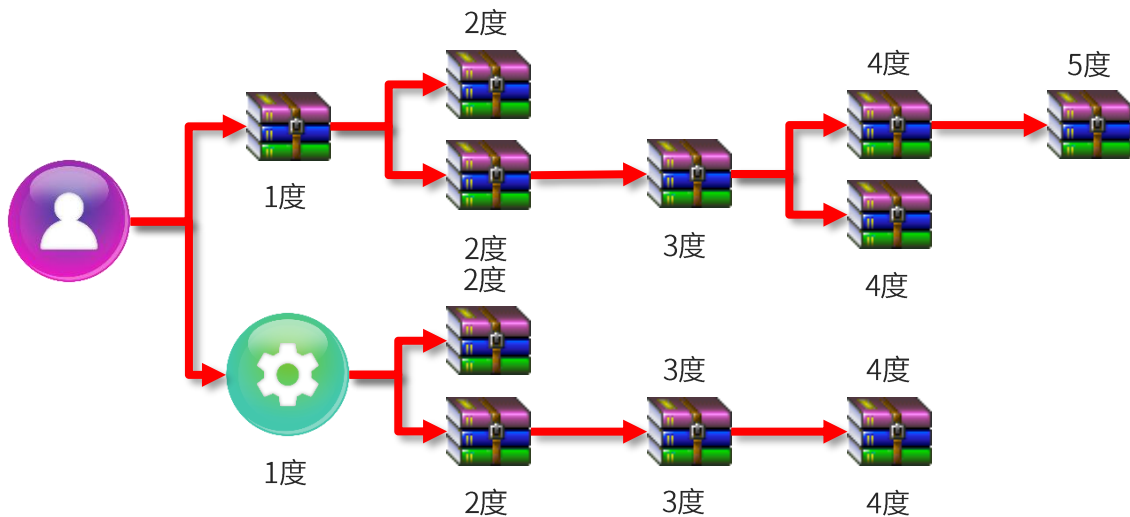
- ◆ 直接依赖：在当前项目中通过依赖配置建立的依赖关系
- ◆ 间接依赖：被资源的资源如果依赖其他资源，当前项目间接依赖其他资源



■ 依赖管理

依赖传递冲突问题

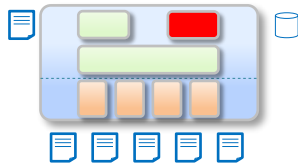
- 路径优先：当依赖中出现相同的资源时，层级越深，优先级越低，层级越浅，优先级越高
- 声明优先：当资源在相同层级被依赖时，配置顺序靠前的覆盖配置顺序靠后的
- 特殊优先：当同级配置了相同资源的不同版本，后配置的覆盖先配置的



可选依赖

- 可选依赖指对外隐藏当前所依赖的资源——不透明

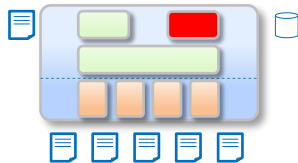
```
<dependency>  
  <groupId>junit</groupId>  
  <artifactId>junit</artifactId>  
  <version>4.12</version>  
  <optional>true</optional>  
</dependency>
```



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

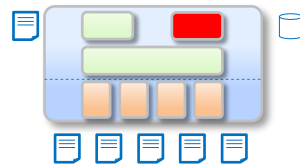
```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <exclusions>
    <exclusion>
      <groupId>org.hamcrest</groupId>
      <artifactId>hamcrest-core</artifactId>
    </exclusion>
  </exclusions>
</dependency>
```



■ 依赖管理

小节

- 依赖管理
- 依赖传递
- 可选依赖（不透明）
- 排除依赖（不需要）

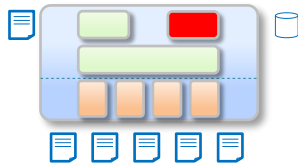


■ 依赖管理

依赖范围

- 依赖的jar默认情况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



■ 依赖管理

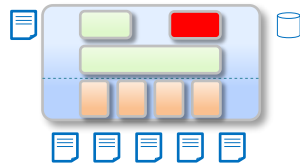
依赖范围传递性

- 带有依赖范围的资源在进行传递时，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

↑
间接依赖

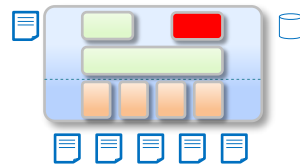
← 直接依赖



■ 依赖管理

小节

- 依赖范围
- 依赖范围传递性（了解）



目录 Contents

- ◆ Maven简介
- ◆ 下载与安装
- ◆ Maven基础概念
- ◆ 第一个Maven项目（手工制作）
- ◆ 第一个Maven项目（IDEA生成）
- ◆ 依赖管理
- ◆ 生命周期与插件

目录 Contents

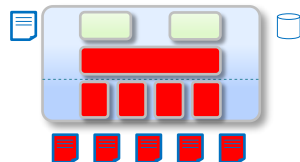
◆ 生命周期与插件

- ◆ 构建生命周期
- ◆ 插件

■ 生命周期与插件

项目构建生命周期

- Maven构建生命周期描述的是一次构建过程经历经历了多少个事件



compile

test-compile

test

package

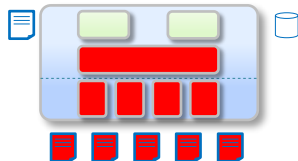
install



■ 生命周期与插件

项目构建生命周期

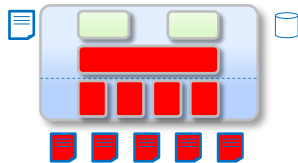
- Maven对项目构建的生命周期划分为3套
 - ◆ clean: 清理工作
 - ◆ default: 核心工作, 例如编译, 测试, 打包, 部署等
 - ◆ site: 产生报告, 发布站点等



■ 生命周期与插件

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



■ 生命周期与插件

default构建生命周期

- validate (校验)
- initialize (初始化)
- generate-sources (生成源代码)
- process-sources (处理源代码)
- generate-resources (生成资源文件)
- process-resources (处理资源文件)
- **compile** (编译)
- process-classes (处理类文件)
- generate-test-sources (生成测试源代码)
- process-test-sources (处理测试源代码)
- generate-test-resources (生成测试资源文件)
- process-test-resources (处理测试资源文件)
- **test-compile** (编译测试源码)
- process-test-classes (处理测试类文件)
- **test** (测试)
- prepare-package (准备打包)
- **package** (打包)
- pre-integration-test (集成测试前)
- integration-test (集成测试)
- post-integration-test (集成测试后)
- verify (验证)
- **install** (安装)
- deploy (部署)

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。

初始化构建状态，比如设置属性值。

生成包含在编译阶段中的任何源代码。

处理源代码，比如说，过滤任意值。

生成将会包含在项目包中的资源文件。

复制和处理资源到目标目录，为打包阶段做好准备。

编译项目的源代码。

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。

生成包含在编译阶段中的任何测试源代码。

处理测试源代码，比如说，过滤任意值。

为测试创建资源文件。

复制和处理测试资源到目标目录。

编译测试源代码到测试目标目录。

处理测试源码编译生成的文件。

使用合适的单元测试框架运行测试 (JUnit是其中之一)。

在实际打包之前，执行任何的必要的操作为打包做准备。

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。

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。

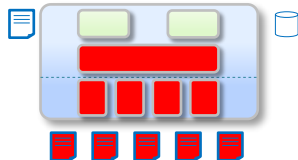
处理和部署项目到可以运行集成测试环境中。

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。

运行任意的检查来验证项目包有效且达到质量标准。

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。

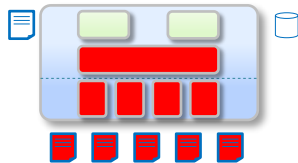
将最终的项目包复制到远程仓库中与其他开发者和项目共享。



■ 生命周期与插件

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



■ 生命周期与插件

default构建生命周期

- validate (校验)
- initialize (初始化)
- generate-sources (生成源代码)
- process-sources (处理源代码)
- generate-resources (生成资源文件)
- process-resources (处理资源文件)
- compile (编译)
- process-classes (处理类文件)
- generate-test-sources (生成测试源代码)
- process-test-sources (处理测试源代码)
- generate-test-resources (生成测试资源文件)
- process-test-resources (处理测试资源文件)
- test-compile (编译测试源码)
- process-test-classes (处理测试类文件)
- test (测试)
- prepare-package (准备打包)
- package (打包)
- pre-integration-test (集成测试前)
- integration-test (集成测试)
- post-integration-test (集成测试后)
- verify (验证)
- install (安装)
- deploy (部署)

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。

初始化构建状态，比如设置属性值。

生成包含在编译阶段中的任何源代码。

处理源代码，比如说，过滤任意值。

生成将会包含在项目包中的资源文件。

复制和处理资源到目标目录，为打包阶段最好准备。

编译项目的源代码。

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。

生成包含在编译阶段中的任何测试源代码。

处理测试源代码，比如说，过滤任意值。

为测试创建资源文件。

复制和处理测试资源到目标目录。

编译测试源代码到测试目标目录。

处理测试源码编译生成的文件。

使用合适的单元测试框架运行测试（JUnit是其中之一）。

在实际打包之前，执行任何的必要的操作作为打包做准备。

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。

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。

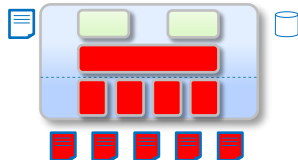
处理和部署项目到可以运行集成测试环境中。

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。

运行任意的检查来验证项目包有效且达到质量标准。

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。

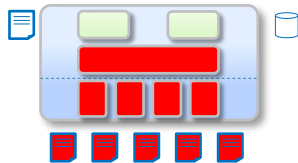
将最终的项目包复制到远程仓库中与其他开发者和项目共享。



■ 生命周期与插件

插件

- 插件与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



■ 生命周期与插件

插件

- 插件与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

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-source-plugin</artifactId>
      <version>2.2.1</version>
      <executions>
        <execution>
          <goals>
            <goal>jar</goal>
          </goals>
          <phase>generate-test-resources</phase>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```



■ 生命周期与插件

default构建生命周期

- validate (校验)
- initialize (初始化)
- generate-sources (生成源代码)
- process-sources (处理源代码)
- generate-resources (生成资源文件)
- process-resources (处理资源文件)
- compile (编译)
- process-classes (处理类文件)
- generate-test-sources (生成测试源代码)
- process-test-sources (处理测试源代码)
- generate-test-resources (生成测试资源文件)
- process-test-resources (处理测试资源文件)
- test-compile (编译测试源码)
- process-test-classes (处理测试类文件)
- test (测试)
- prepare-package (准备打包)
- package (打包)
- pre-integration-test (集成测试前)
- integration-test (集成测试)
- post-integration-test (集成测试后)
- verify (验证)
- install (安装)
- deploy (部署)

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。

初始化构建状态，比如设置属性值。

生成包含在编译阶段中的任何源代码。

处理源代码，比如说，过滤任意值。

生成将会包含在项目包中的资源文件。

复制和处理资源到目标目录，为打包阶段最好准备。

编译项目的源代码。

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。

生成包含在编译阶段中的任何测试源代码。

处理测试源代码，比如说，过滤任意值。

为测试创建资源文件。

复制和处理测试资源到目标目录。

编译测试源代码到测试目标目录。

处理测试源码编译生成的文件。

使用合适的单元测试框架运行测试（JUnit是其中之一）。

在实际打包之前，执行任何的必要的操作作为打包做准备。

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。

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。

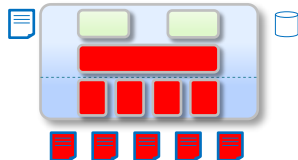
处理和部署项目到可以运行集成测试环境中。

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。

运行任意的检查来验证项目包有效且达到质量标准。

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。

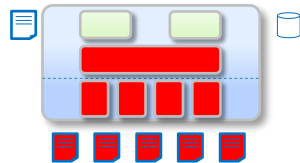
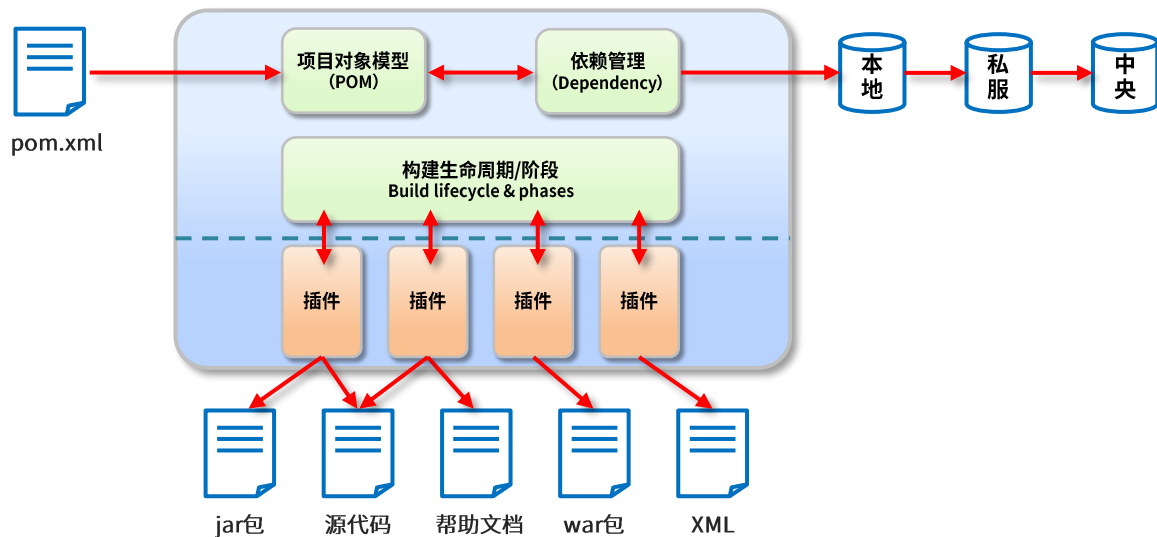
将最终的项目包复制到远程仓库中与其他开发者和项目共享。



■ 生命周期与插件

小节

- 生命周期
- 插件





Maven基础

- Maven简介
- 下载与安装
- Maven基础概念
- 第一个Maven项目（手工制作）
- 第一个Maven项目（IDEA生成）
- 依赖管理
- 生命周期与插件

