Maven 3知识图谱

目录结构

- bin — 存放命令脚本 — mvn等
- boot — 存放类加载器，负责加载maven的核心组件
- lib — 存放maven组件，超级POM也在这里
- conf
  - 全局配置文件settings.xml
  - 日志

坐标和依赖

- 依赖范围
  - Maven在三个阶段从三个classpath下引入依赖
    - 编译
    - 测试
    - 运行
  - 五种依赖范围
    - compile — 编译、测试、运行
    - test — 测试 — junit
    - provided — 编译、测试 — servlet-api
    - system
      - 编译、测试
      - 与其它依赖范围不同，此依赖范围表示从本机文件系统导入依赖
    - runtime — 测试、运行 — jdbc驱动
- 传递性依赖
  - Maven遵循两大原则，避免依赖冲突和重复依赖
    - 短路径优先
    - 声明顺序靠前优先
  - 直接依赖与间接依赖的依赖范围交叉影响当前项目感知到的间接依赖的依赖范围

	compile	test	provided	runtime
compile	compile			runtime
test	test			test
provided	provided		provided	provided
runtime	runtime			runtime

  - 黄色：第一依赖的依赖范围
  - 蓝色：第二依赖的依赖范围
- 依赖坐标和依赖主构件名称关系
  - artifactId-version.packaging
  - 特例 — 插件 — packaging是maven-plugin，但主构件扩展名为jar

聚合和继承

- 聚合是为了能够一次构建多个Maven构件
- 继承是为了消除重复的配置
- 所有Maven构件的构建顺序被称为反应堆 — 可以根据需要裁剪反应堆以达到部分构建的目的
  - 单独构件构件
  - 指定构件及其依赖的构件
  - 指定构件、其依赖的构件、依赖其的构件
  - ...

Maven测试

- 借助测试框架实现自动化测试
- 可以对测试用例范围进行操作
  - 跳过
  - 包含
  - 排除

持续集成

- 可与持续集成工具配合，自动签出代码并构建
- 以Hudson为代表

灵活构建

- <profile>灵活激活配置
  - 指定id激活
  - 触发属性激活
  - 依赖指定文件激活
- 资源过滤
  - 可以将资源目录下的配置文件经常发生变更的部分使用Maven属性替换
  - 属性有六种来源
    - pom属性
    - settings属性
    - 内置属性
      - ${version}
      - ${groupId}
      - ...
    - 自定义属性
    - Java系统属性
    - 环境变量属性

Archetype

- 选择合适的archetype
  - 从archetype列表中选择 — maven archetype项目没有特殊的打包方式，默认jar即可
  - 可以自己编写archetype
- archetype-catalog.xml — 存储了各种archetype的坐标，为archetype列表提供内容
- Maven读取来源
  - internal — 内置了一份archetype-catalog.xml
  - local — 用户目录下
  - remote — 中央仓库
  - 默认来源
  - file — 从文件系统路径下读取
  - http — 从指定远程仓库中获取

仓库

- 仓库类型
  - 本地仓库
  - 远程仓库
  - 私服仓库（代理）
    - 代表
      - 最早出现 — Archiva
      - 最早支持用户界面 — Articfactory
      - 最流行 — Nexus
    - 以Nexus为代表，将仓库又分为多种类型
      - 远程仓库代理
        - 发布版
        - 快照版
      - 组织/内部私有
      - 第三方依赖
      - 聚合其它仓库 — 仓库组
- 解析依赖的机制
  - 判定依赖范围是否是system
    - 如果依赖的依赖范围为system，则在指定的文件系统路径下寻找
    - 如果不是，则根据检索的依赖版本采取不同的行为
      - 不明晰的版本
        - SNAPSHOT — 在所有仓库检索元数据文件（groupId/artifactId/maven-metadata.xml）后与本地仓库元数据合并，挑选出最新的快照版本
        - RELEASE — 基本同SNAPSHOT，最终返回最新的发布版本
        - LATEST — 基本同SNAPSHOT，最终返回最新的版本（不关注是否为发布版还是快照版）
      - 明晰版本
        - 首先在本地仓库搜索
        - 如果有代理仓库，搜索代理仓库：没有，则搜索远程仓库
        - 远程仓库默认只配置了中央仓库，如果有多个远程仓库，则按声明顺序依次寻找
  - 如果找不到目标依赖，报错

生命周期和插件

- 生命周期分为三个阶段
  - clean阶段
    - pre-clean
    - clean
    - post-clean
  - default阶段
    - 2
      - validate
      - inialize
    - 6
      - generate-sources
      - process-sources
      - generate-resources
      - process-resources
      - compile
      - process-classes
    - 6
      - generate-test-sources
      - process-test-sources
      - generate-test-resources
      - process-test-resources
      - test-compile
      - process-test-classes
    - 1
      - test
    - 2
      - prepare-package
      - package
    - 3
      - pre-integration
      - integration
      - post-integration
    - 3
      - verify
      - install
      - deploy
  - site阶段
    - pre-site
    - site
    - post-site
    - site-deploy
- 插件目标与生命周期某阶段相绑定
  - Maven内置了部分绑定
  - 可以在pom.xml中自定义绑定
  - 也可以自己编写插件来绑定 — 插件的<package>值为maven-plugin
- 插件仓库 — 插件有专门的仓库，与依赖分别存储 — 主要有两大插件仓库
- 插件解析机制
  - 有版本号
    - 首先在本地仓库寻找
    - 其次向远程仓库仓库寻找
  - 无版本号
    - 核心插件 — 超级POM中已经声明了核心插件的版本，按明晰版本方法寻找即可
    - 非核心插件 — 在所有仓库搜索元数据文件（groupId/artifactId/maven-metadata.xml）后合并，返回最新的发布版本
- 解析插件前缀 — 搜索所有仓库artifactId/maven-metadata.xml，找到与artifactId相匹配的目标前缀
  - 先搜索两大主流插件仓库（即先尝试它们的groupId）
  - 其次搜索用户自定义groupId