# Maven学习笔记

# Maven基础概念

## 仓库

概念：用于存储资源，包含各种jar包

## 坐标

Maven中的坐标用于描述仓库中资源的位置

1. 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

packing：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下载工作交由机器完成

### 本地仓库配置

* 默认位置

Default: ${user.home}/.m2/repository

自定义位置

<localRepository>E:\Mavenrepository\respository</localRepository>

### 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，使用空格隔开

|  |  |
| --- | --- |
| mvn compile | 编译 |
| mvn clean | 清理 |
| mvn test | 测试 |
| mvn package | 打包 |
| mvn install | 安装到本地仓库 |

### 构建插件

|  |
| --- |
| <!--构建的意思-->  <build>  <!--设置插件-->  <plugins>  <!--具体的插件-->  <plugin>  <!-- https://mvnrepository.com/artifact/org.apache.tomcat.maven/tomcat7-maven-plugin -->  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.1</version>  </plugin>  </plugins>  </build> |

### 在tomcat配置文件中，配置端口号以及相关虚拟路径

|  |
| --- |
| <configuration>  <port>80</port>  <path>/</path> </configuration> |

### Pom文件解析分析

|  |
| --- |
| <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"> <!--指定pom的模型版本-->  <modelVersion>4.0.0</modelVersion>  <!--打包方式，java为jar，web为war，-->  <packaging>war</packaging>   <name>web01</name>  <!--组织id，通常是域名反写，做作婊-->  <groupId>com.ly</groupId>  <!--项目id，-->  <artifactId>web01</artifactId>  <!--版本号：release：完成版SNAPSHOT：开发版-->  <version>1.0-SNAPSHOT</version>  <!--设置当前工程的所有依赖-->  <dependencies>  <dependency>  <!--代表具体的依赖-->  <groupId></groupId>  <artifactId></artifactId>  </dependency>  </dependencies>  <!--构建的意思-->  <build>  <!--设置插件-->  <plugins>  <!--具体的插件-->  <plugin>  <!-- https://mvnrepository.com/artifact/org.apache.tomcat.maven/tomcat7-maven-plugin -->  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>tomcat7-maven-plugin</artifactId>  <version>2.1</version>  <configuration>  <port>8080</port>  <path>/</path>  </configuration>  </plugin>  </plugins>  </build> |

## 依赖传递

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

## 依赖传递冲突问题

### 路径优先

当依赖关系中出现相同资源时，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>中的<optional></optional>标签，里面的值为true时，设置为不透明

## 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

|  |
| --- |
| <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.11</version>  <scope>test</scope>  <exclusions>  <exclusion>  <groupId></groupId>  <artifactId></artifactId>  </exclusion>  </exclusions>  </dependency> </dependencies> |

## 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）

## 依赖范围传递性

带有依赖范围的资源进行传递时，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 |

## 生命周期与插件

### Maven对项目构建的生命周期规划为3套

1. clean：清理工作
2. default：核心工作，例如编译，测试，打包，部署等
3. site：生产报告，发布站点等

#### clean生命周期

1. pre-clean执行一些需要在clean之前完成的工作
2. clean:移除所有上一次构建生成的文件
3. post-clean：执行一些需要在clean之后立刻完成的工作

#### 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1部署做工作

site-deploy：将生成的站点文档部署到特定的服务器上

### 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

通过插件可以在自定义其他功能

|  |
| --- |
| <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> |

# Maven高级部分

## 分模块开发

模块中今含当前模块对应的功能类与配置文件

Spring核心配置根据模块功能不同进行独立制作

当前模块所依赖的模块通过导入坐标的形式加入到当前模块后才可以使用

Web.xml需要加载所有的spring核心文件

## 模块构建管理模块

### 聚合

作用：聚合用于快速构建maven工程，一次性构建多个项目/模块

### 需要在pom.xml中加入一个配置文件

|  |
| --- |
| <!--定义定义该工程进行构建管理--> <packaging>pom</packaging>  <!--管理的工程列表--> <modules>  <!--具体的工程名称-->  <module>../ssm\_controller</module>  <module>../ssm\_service</module>  <module>../ssm\_pojo</module>  <module>../ssm\_dao</module> </modules> |

## 继承

作用：通过继承可以实现在子工程中沿用父工程的配置

Maven中的继承与java中的继承相似，在子工程中配置继承关系

### 进行所有的依赖管理

|  |
| --- |
| <!--声明此处进行依赖管理--> <dependencyManagement>  <!--具体的依赖--> <dependencies>  <!--spring环境-->  <!--spring环境-->  <!--spring环境-->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>5.1.9.RELEASE</version>  </dependency>    <!--mybatis环境-->  <!--mybatis环境-->  <!--mybatis环境-->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.5.3</version>  </dependency>  <!--mysql环境-->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.47</version>  </dependency>  <!--spring整合jdbc-->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>5.1.9.RELEASE</version>  </dependency>  <!--spring整合mybatis-->  <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>2.0.3</version>  </dependency>  <!--druid连接池-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.16</version>  </dependency>  <!--分页插件坐标-->  <dependency>  <groupId>com.github.pagehelper</groupId>  <artifactId>pagehelper</artifactId>  <version>5.1.2</version>  </dependency>    <!--springmvc环境-->  <!--springmvc环境-->  <!--springmvc环境-->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>5.1.9.RELEASE</version>  </dependency>  <!--jackson相关坐标3个-->  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId>  <version>2.9.0</version>  </dependency>  <!--<dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-core</artifactId>  <version>2.9.0</version>  </dependency>  <dependency>  <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-annotations</artifactId>  <version>2.9.0</version>  </dependency>-->  <!--servlet环境-->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>3.1.0</version>  <scope>provided</scope>  </dependency>    <!--其他组件-->  <!--其他组件-->  <!--其他组件-->  <!--junit单元测试-->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  </dependency>  <!--spring整合junit-->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>5.1.9.RELEASE</version>  </dependency> </dependencies> </dependencyManagement> |

### 定义父工程依赖的引入

在子工程中声明其父工程坐标与对应的位置

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

## 继承和聚合

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那些

继承是在子模块中配置关系，父模块无法感知那些子模块继承了自己

## 属性

### 自定义属性

|  |
| --- |
| 定义自定义的属性 <properties>  <spring.version>5.1.9.RELEASE</spring.version> </properties> |

### 使用属性

|  |
| --- |
| <version>${spring.version}</version> |

### 属性类型

1. 自定义属性
2. 内置属性
3. Settings属性
4. Java系统属性
5. 环境变量属性

## 版本管理

### 工程版本（快照版本）

* 项目开发过程中，为方便团队成员合作，解决模块相互依赖和时时更新的问题，开发者对每一个模块进行构建时候，输出的临时性版本叫快照版本（测试阶段版本）
* 快照版本随着开发的进程不断更新

### 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成文件是稳定的，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

### 工程版本号约定

约定规范：

* <主版本><次版本><增量版本><里程版本>
* 主版本：表示项目重大构架的变更，如：spring5相较于spring4的迭代
* 次版本：表示较大的功能增加和变化，或者全面系统地修复漏洞
* 增量版本：表示有重大漏洞的修复
* 里程碑版本：表明一个版本的里程碑版本（版本内部）。这样的版本同下一个正式版本相比，相对来说不是很稳定，有待更多的测试

范例：5.1.9.RELEASE

## 资源加载属性值

### 配置资源文件的信息

|  |
| --- |
| <!--配置资源文件的信息--> <build>  <resources>  <resource>  //${project.basedir}表示所有的通配文件  <directory>${project.basedir}/src/main/resources</directory>  <directory>ssm\_dao/src/main/resources</directory>  <filtering>true</filtering>  </resource>  </resources> </build> |

### 配置测试资源文件的配置文件

|  |
| --- |
| <build>  <testResources>  <testResource>  <directory>${project.basedir}/src/main/resources</directory>  <filtering>true</filtering>  </testResource>  </testResources> </build> |

### 小节

配置文件中读取pom属性值

* 在pom配置文件中设定配置文件路径
* 开启加载pom属性过滤功能
* 使用${属性名}格式引用pom属性

## 多环境开发设置

### 多环境兼容

|  |
| --- |
| <!--创建多环境--> <profiles>  <!--定义具体点的环境：生产环境-->  <profile>  <!--定义环境对应的唯一名称-->  <id>pro\_env</id>  <!--定义环境中的属性值-->  <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>dep\_env</id>  <jdbc.url>jdbc:mysql://127.2.2.2:3306/ssm\_db</jdbc.url>  </profile> </profiles> |

开发环境打包

执行对应的文件名：

Install -P dep\_env

加载指定环境：

作用：加载指定环境配置

调试格式：

Mvn 指令 -p 环境定义id

范例：

mvn -p install -p pro\_env

## 跳过测试

|  |
| --- |
| <!--配置跳过测试--> <plugin>  <groupId>org.apache.tomcat.maven</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>2.22.1</version>  <configuration>  <!--设置跳过测试-->  <skipTests>true</skipTests>  </configuration> </plugin> |

### 配置指定的测试和排除测试

|  |
| --- |
| <!--设置指定测试--> <configuration>  <!--设置指定的模块-->  <includes>  <include>\*\*/UserServiceTest.java</include>  </includes>  <!--设置跳过的测试-->  <excludes>  <exclude>   </exclude>  </excludes> </configuration> |

使用命令跳过测试

* 命令

mvn 指令 -D skipTest

注意事项

* 执行的指令生命周期必须包含测试环节

## 私服

### 私服资源获取

### 仓库分类

* 宿主仓库hosted：

保存无法从中央仓库获取的资源

自主研发

第三方非开源项目

* 代理仓库proxy

代理远程仓库，通过nexus访问其他公共仓库，例如中央仓库

* 仓库组group

将若干个仓库组成一个群组，简化配置

仓库不能保存资源，属于设计型仓库

### 资源上传

上传资源时提供对应的信息

* 保存的位置
* 资源文件
* 对应坐标

### 发布的配置文件设置

|  |
| --- |
| <!--发布的配置管理--> <distributionManagement>  <repository>  <id>heima-release</id>  <url>http://localhost:8081/repository/lyy-release/</url>  </repository> </distributionManagement> |