链滴

Maven 入门知识点 个人学习笔记

作者：PeterChu

原文链接：https://ld246.com/article/1598617486176

来源网站：链滴

1. Maven 项目的创建必须使用 Maven 规定好的目录结构样式。

```
Project
   |--src(源码包)
      |--main（正常的源码包）
         |--java（.java 文件的目录）
         |--resources（资源文件的目录）
      |--test（测试的源码包）
         |--java
         |--resources
   |--target（class 文件、报告等信息存储的地方）
   |--pom.xml（maven 工程的描述文件）
```

2. 在 Project 文件夹下可以使用命令窗口运行 maven 命令，实际是需要在 pom.xml 文件所在的目录中执行。

3. Maven 的两大功能：管理依赖、构建工程。

Maven 是通过 pom.xml 进行依赖管理。

4. test 包中存放的是对 java 中的源码的测试类。每次使用

5. Maven 的命令：

第一次使用 Maven 命令时需要下载相关的 maven 命令使用时需要的 jar 依赖。

a. mvn -v　打印 Maven 的版本信息

b. mvn compile　编译源码命令（不包含 test 目录下的源码）。会将源码包中的源码编译后，将生

的 class 文件、报告信息（如 test 测试报告）等文件放入 target 目录中。

c. mvn clean 清除命令，清除已经编译好的 class 文件，具体来说是清除的是 target 目录中的文件。

d. mvn test 测试命令，该命令会将 test 目录中的源码进行编译。（会包含 main 目录中的源码）

e. mvn package 打包项目（可以打包为 jar、war、pom 三种，pom.xml 中设置，默认为 jar）

f. mvn install 安装命令。会将打好的 jar 包安装到本地仓库。

打包的文件会自动放到本地 maven 的仓库中（如.m2文件夹下），此时可以像使用其他 jar 一样，在他地方依赖该项目 jar 包。

g. 组合命令： mvn clean compile 先清空，再编译，通常应用于上线前执行，清除测试类

mvn clean test 先清空，再执行测试，通常应用于测试环节

mvn clean package 先执行clean，再执行package，将项目打包，通常应用于发布前

执行过程：清理、编译、测试、打包

mvn clean install 先执行clean，再执行install，将项目打包，通常应用于发布前

执行过程：清理、编译、测试、打包、部署

6. 依赖范围

<scope>compile<scope> 设置依赖生效范围。缺省默认为 compile .

compile 对主代码classpath 有效，对test 有效，打包后对 classpath 有效，如 log4j

test 对主代码classpath 无效，对测试代码 classpath 有效，打包后对 classpath 无效，如 junit

provided 对主代码 classpath 有效，对测试代码 classpath 有效，打包后对 classpath 无效，如 servlet-api （开发时需要使用，打包后容器 tomcat 会提供，如果对打包后有效，则会可能冲突）

runtime 只对打包后对 classpath 有效，如 JDBC Driver ,Implementation.

7. 依赖具有传递性。

被依赖的工程中依赖的工程，可以直接被传递给最终的依赖工程。

如，B依赖A（A为C的第二直接依赖），C依赖B（B为C的第一直接依赖），则若依赖范围都为默认 compile 时，A会同时在C依赖B时会被C依赖。否则：

- 当第二依赖的范围是 compile 时，传递性依赖的范围与第一直接依赖的范围一致。 当第二直接依的范围是 test 时，依赖不会传递。
- 当第二依赖的范围是 provided 时，只传递第一直接依赖范围也为 provided 的依赖，且传递性依的范围同样为 provided。
- 当第二直接依赖的范围是 runtime 时，传递性依赖的范围与第一直接依赖的范围一致，但 compile 例外，此时传递传递的依赖范围为 runtime.

8. 依赖冲突

Maven 解决依赖冲突的方式为 就近原则。

- 第一直接依赖与第二依赖中出现的依赖版本冲突后（跨 pom 文件），则会采用第一直接依赖的版。即第一依赖的权重更高。
- 同一 pom.xml 文件中重复引用同一 artifactId 的不同版本，则最下面的权重更高。

9. 排除依赖（可选依赖）

● 可选依赖是通过在被依赖工程的 pom.xml 中设置自己为是否为可选依赖。

<optional> true/false 是否可选，也可以理解为是否向下传递。

在依赖中添加optional选项决定此依赖是否向下传递，如果是true则不传递，如果是false就传递，默为false。

```
<dependencies>
      <dependency>
         <groupId>com.itheima.maven</groupId>
         <artifactId>MavenFirst</artifactId>
         <version>0.0.1-SNAPSHOT</version>
<!--           <optional>true</optional> -->
      </dependency>
      ...
```

该方式不够灵活，当多个工程依赖某个工程时，不能处理不同的依赖情况。

● 为解决多个工程依赖同一工程时，不同的依赖情况，可以选择排除依赖方式。

排除依赖包中所包含的依赖关系，不需要添加版本号。

如果在本次依赖中有一些多余的jar包也被传递依赖过来，如果想把这些jar包排除的话可以配置exclusions进行排除。

```
<dependencies>
      <!-- 第一直接依赖 -->
      <dependency>
         <groupId>com.itheima.maven</groupId>
         <artifactId>MavenSecond</artifactId>
         <version>0.0.1-SNAPSHOT</version>

         <exclusions>
            <exclusion>
               <groupId>com.itheima.maven</groupId>
               <artifactId>MavenFirst</artifactId>
            </exclusion>
         </exclusions>
      </dependency>
   </dependencies>
```

10. Maven 的坐标

groupId：定义当前Maven组织名称

artifactId：定义实际项目名称

version：定义当前项目的当前版本

依赖管理就是对项目中jar 包的管理。可以在pom文件中定义jar包的GAV坐标，管理依赖。

依赖声明主要包含如下元素：

```
<dependencies>
   <dependency>
      <groupId>junit</groupId>
```

```
        <artifactId>junit</artifactId>
        <version>4.10</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

11. 生命周期

Maven 有三个生命周期： clean 生命周期、default 生命周期、site 生命周期。

● 生命周期可以理解为项目构建的步骤集合。

● 生命周期是有多个阶段（Phase）组成。每个阶段都是一个完整的功能，比如 mvn clean 中的 clean 就是一个阶段。

● 在 maven 中，只要是在同一个生命周期中，执行后面的阶段，那么前面的阶段也会被执行，不需额外去输入前面阶段的命令。

clean 清理项目

pre-clean 执行清理前的工作

clean 清理上一次构建生成的所有文件

post-clean 执行清理后的文件

default 构建项目（最核心的一部分）

compile、test、package、install等等。

site 生成项目站点

pre-site 在生成项目站点前要完成的工作

site 生成项目的站点文档

post-site 在生成项目站点后要完成的工作

site-deploy 发布生成的站点到服务器上

在这几个clean、compile、test、package、install命令中，当我们运行package的时候，clean、compile、test将会在在package之前依次运行。

● Clean 生命周期

mvn clean 命令，等同于 mvn pre-clean 、clean 命令集合。只要执行后面的命令，则前面的命令会执行，不需要再重新去输入命令。

有 Clean 生命周期，在生命周期又有 clean 阶段。

● Default：构建项目

Default生命周期是Maven生命周期中最重要的一个，绝大部分工作都发生在这个生命周期中。这里只解释一些比较重要和常用的阶段：

validate
generate-sources
process-sources
generate-resources
process-resources 复制并处理资源文件，至目标目录，准备打包。

compile 编译项目的源代码。
process-classes
generate-test-sources
process-test-sources
generate-test-resources
process-test-resources 复制并处理资源文件，至目标测试目录。
test-compile 编译测试源代码。
process-test-classes
test 使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。
prepare-package
package 接受编译好的代码，打包成可发布的格式，如 JAR 。
pre-integration-test
integration-test
post-integration-test
verify
install 将包安装至本地仓库，以让其它项目依赖。
deploy 将最终的包复制到远程的仓库，以让其它开发人员与项目共享。

运行任何一个阶段的时候，它前面的所有阶段都会被运行，这也就是为什么我们运行mvn install 的候，代码会被编译，测试，打包。此外，Maven的插件机制是完全依赖Maven的生命周期的，因此解生命周期至关重要。

● Site：生成项目站点

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

12. 继承

采用父工程进行统一管理依赖的 jar 包，统一管理依赖的版本号。

基本上不管哪一个 maven 项目，都会用到 junit，而且每一个项目的 pom.xml 文件，都会对其进行置。这个时候就会出现很多重复的配置代码。

所以在 maven 中，我们可以像 Java 一样，将共用的部分，写成一个父类。

具体使用方法：

新建一个 maven 项目（假设命名为 parent），packaging 改为 pom，将要写入的依赖，写在 depe dencyManagement 中。如下所示：

```
<properties>
   <junit.version>4.12</junit.version>  <!-- 抽取管理版本号 -->
  </properties>

  <dependencyManagement>
   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <scope>test</scope>
```

```
        <version>${junit.version}</version>
      </dependency>
    </dependencies>
  </dependencyManagement>
```

在其他的 maven 项目中，去继承 parent。代码如下所示：

在 pom.xml 文件中：

```
<!-- 引入父工程依赖 -->
  <parent>
    <groupId>parent的groupId</groupId>
    <artifactId>parent的artifactId</artifactId>=
    <version>parent的version</version>
  </parent>

  <dependencies>
    <dependency>
      <!-- 只需要 GA 即可引入父工程中的依赖 -->
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
    </dependency>
  </dependencies>
```

如上所示，继承 parent 之后，再引用 junit，已经不需要进行过多的配置了，只需要定义它的坐标即
。

13. 聚合

● 聚合工程必须为 pom 打包方式。一般聚合工程和父工程合并为一个工程。聚合工程可以继承父工
。

● 创建聚合工程后，其所使用到的其他模块如业务层、持久层、表现层分别创建为 Maven Module
式。其中表现层必须设置打包方式为 war 方式打包。

● 其他模块创建需要建立在之前聚合工程基础上。（IDE中一般表现为选中聚合工程目录后右键 new
project）

● 聚合之后，聚合工程中的 pom 文件内容中会包含 <modules> 标签，标注被聚合的模块工程。

● 运行聚合工程，可以使用 maven-tomcat 插件，或在 IDE 中配置 tomcat 运行。注意：运行之前
需要将父工程安装到本地仓库中。（如果聚合工程继承了父工程。install）

14. 仓库

● 用来统一存储所有Maven共享构建的位置就是仓库。根据Maven坐标定义每个构建在仓库中唯一
储路径大致为：groupId/artifactId/version/artifactId-version.packaging

● 分类：

本地仓库（~/.m2/repository 每个用户只有一个本地仓库）

远程仓库（中央仓库、私服仓库）。

● 私服仓库 Nexus

访问URL: http://localhost:8080/nexus-2.7.0-06/

默认账号:

用户名： admin

密码： admin123

15. 部署构建到 Nexus 私服

第一步：Nexus的访问权限控制

在本地仓库的setting.xml中配置如下：

```
<server>
          <id>releases</id>
          <username>admin</username>
          <password>admin123</password>
        </server>
        <server>
          <id>snapshots</id>
          <username>admin</username>
          <password>admin123</password>
        </server>
```

第二步：配置pom文件

在需要构建的项目中修改pom文件

```
<distributionManagement>
          <repository>
            <id>releases</id>
            <name>Internal Releases</name>
            <url>http://localhost:8080/nexus-2.7.0-06/content/repositories/releases/</url>
          </repository>
          <snapshotRepository>
            <id>snapshots</id>
            <name>Internal Snapshots</name>
            <url>http://localhost:8080/nexus-2.7.0-06/content/repositories/snapshots/</url

          </snapshotRepository>
        </distributionManagement>
```

第三步：执行maven的deploy命令。（）

---

archetype 骨架

IDEA中创建maven web项目的详细部署

Hello,Maven 黑客派@liumapp

Maven settings配置中的mirrorOf

maven配置多仓库镜像